UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GEAN MICHEL CERETTA

OTIMIZAÇÃO DO CLIMA AVÍCOLA PARA MAIOR PRODUTIVIDADE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2021

GEAN MICHEL CERETTA

OTIMIZAÇÃO DO CLIMA AVÍCOLA PARA MAIOR PRODUTIVIDADE

Optimization of poultry climate for greater productivity

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dalcimar Casanova, Dr.
Universidade Tecnológica Federal do Paraná

Coorientador: Prof. Marcelo Teixeira, Dr.
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2021

GEAN MICHEL CERETTA

OTIMIZAÇÃO DO CLIMA AVÍCOLA PARA MAIOR PRODUTIVIDADE

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Data de aprovação: 11 de Novembro de 2021

Dalcimar Casanova
Doutor
Universidade Tecnológica Federal do Paraná

Marcelo Teixeira
Doutor
Universidade Tecnológica Federal do Paraná

Darlan Felipe Klotz
Mestre
Instituto Federal Catarinense

Richardson Ribeiro
Doutor
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2021

RESUMO

Métodos de aprendizagem de máquina, busca e mutação foram empregados para encontrar parâmetros de ambiência (também chamados de planos de ação) para controladores de aviários de frango de corte, capazes de proporcionar maiores ganhos de peso por consumo de alimento, com o potencial de aumentar a produtividade. Configurar parâmetros de temperatura e umidade que sejam capazes de trazer o conforto térmico necessário para o bom desenvolvimento dos frangos tradicionalmente depende do conhecimento empírico de um especialista em manejo. O estudo aqui realizado baseia-se em literaturas anteriores de aplicação métodos de aprendizagem de máquina, busca e otimização utilizando algoritmos genéticos, porém empregados a uma base de dados reais de empresa de automação do setor avícola, coletados de forma automática, abordando desde as etapas de engenharia de dados, seleção das características relevantes para aprendizagem, treinamento do modelo de aprendizagem, previsão da produtividade baseado em parâmetros de ambiência, busca e mutação utilizando algoritmos genéticos para obter candidados a planos de ação para maior produtividade. Ao final, a produtividade prevista para o plano de ação proposto é comparada com amostra da base de dados e com a literatura. Também são aprensentados aspectos que poderiam melhorar as previsões para trabalhos futuros na área.

Palavras-chave: Aprendizagem De Máquina. Ambiência. Produtividade. Avicultura. Frangos de Corte. Engenharia de Dados. Algoritmos Genéticos.

ABSTRACT

Machine learning, search and mutation methods were applied to find ambience parameters (also called action plans) for broiler house controllers, capable of providing greater weight gains by feed consumption, with the potential to increase productivity. Setting temperature and humidity parameters that are capable of providing the necessary thermal comfort for the good development of broilers traditionally depends on the empirical knowledge of a poultry specialist. The study carried out here is based on previous literature on the application of machine learning, search and optimization methods using genetic algorithms, but applied to a real database of an automation company in the poultry sector, collected automatically, covering the stages of data engineering, selection of relevant learning characteristics, learning model training, productivity prediction based on ambience parameters, search and mutation using genetic algorithms to obtain candidates for action plans for higher productivity. At the end, the expected productivity for the proposed action plan is compared with a sample from the database and with the literature. Aspects that could improve forecasts for future work in the area are also presented.

Keywords: Machine Learning. Thermal Comfort. Feed Conversion Rate. Productivity. Broiler. Poultry Houses. Data Engineering. Genetic Algorithms.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A carne de frango, segundo a Associação Brasileira de Proteína Animal (ABPA), possui o maior índice de consumo *per capita* no Brasil, sendo considerada fundamental para a garantia da segurança alimentar. Em seu Relatório Anual (ABPA, 2020), destaca o clima favorável, a utilização de energias renováveis e a qualidade das instalações como fatores para que o Brasil consiga produzir carne de frango com 45% a 50% menos emissão de CO_2 do que países como Reino Unido e França e cita um dos Objetivos de Desenvolvimento Sustentável da Organização das Nações Unidas como sendo focar na eficiência e uso de recursos para melhorar a sustentabilidade e beneficiar os aspectos de saúde e bem-estar animal. O número de aves alojadas teve um crescimento de quase 7% em 2019 se comparado com o ano anterior, com uma produção recorde de 13,245 milhões de toneladas, 68% destinada ao mercado interno e 32% para exportações. O estado do Paraná é lider nacional em abate, responsável por 34% do total de frangos, sendo o Brasil líder nas exportações e terceiro maior produtor mundial desta carne (ABPA, 2020).

A busca por melhoras na produtividade deste setor se iniciou nos anos 70, com mudanças tecnológicas e na estrutura produtiva, com objetivos de aumentar o peso médio das aves, reduzir a idade média ao abate e reduzir a taxa de conversão alimentar (ESPíNDOLA, 2012).

A Taxa de Conversão é um dos principais índices de produtividade em uma granja (SKINNER-NOBLE; TEETER, 2004), dado pelo consumo total de ração dividido pelo produto entre o peso médio da aves e a quantidade de aves vivas.

Sendo a taxa de conversão um dos principais indicativos de produtividade, e sendo os frangos muito sensíveis à temperatura e umidade, principalmente nos primeiros dias de vida, manter estes parâmetros de ambiência (também chamados de receitas ou planos de controle) dentro das faixas consideradas confortáveis para as aves é crucial para uma boa produtividade (COBB-VANTRESS, 2018).

Empresas especializadas em genética avícola comercializam aos produtores avícolas os pintainhos[1], geralmente com 1 dia de vida, que são então alojados nos aviários para que seja iniciado o manejo até o peso médio dos frangos atingir o peso desejado pelas empresas do setor de proteína animal como agroindústrias e frigoríficos, que comprarão estes animais para abate. Produtores e técnicos são assessorados com manuais de manejo produzidos pelas empresas de genética avícola. Tais manuais contêm, entre outas informações, os parâmetros de ambiência recomendados para uma melhor produtividade (ROSS-AVIAGEN, 2018), (COBB-VANTRESS, 2018).

O controle da ambiência em aviários modernos é feito de maneira automatizada, utilizando controladores que após serem configurados com parâmetros de ambiência para cada idade das aves, acionam exaustores, entradas de ar, aquecedores e umidificadores (ROSS-AVIAGEN, 2018).

1.1 CONSIDERAÇÕES INICIAIS

A construção desse estudo está fundamentada nas recomendações presentes nos manuais de manejo avícola, elaborados pelas empresas de genética animal e disponibilizados aos produtores (COBB-VANTRESS, 2018)(ROSS-AVIAGEN, 2018).

[1] Filhotes de frango

A motivação para este estudo segue as constatações feitas por Klotz et al. (2020) e Ribeiro et al. (2019) em suas pesquisas que empregam aprendizado de máquina na otimização das receitas de ambiência para frangos de corte, obtendo melhora significativa na produtividade.

Apesar da temperatura e da umidade serem muito importantes para um bom desenvolvimento dos frangos, os manuais de manejo deixam claro ao longo de todo o seu texto que muitos outros parâmetros são importantes, como qualidade da ração, programas de iluminação, níveis de gás carbônico e amônia, entre outros. Todos estes parâmetros deveriam ser considerados como entrada do sistema de aprendizagem a fim de obter uma maior confiabilidade dos resultados.

1.2 PROBLEMA

Os manuais de manejo avícola Cobb-Vantress (2018), Ross-Aviagen (2018) indicam que as técnicas de manejo devem ser ajustadas de acordo com as condições climáticas da região onde o aviário está situado, e com a experiência de manejo dos produtores.

Klotz et al. (2020, p. 3), Ribeiro et al. (2019, p. 1) pontuam que o ajuste das receitas de ambiência feito pelos produtores, baseados em conhecimento empírico levam a decisões que muitas vezes diferem radicalmente das recomendações dos manuais de manejo, induzindo a alimentação insuficiente, resfriamento ou aquecimento desnecessários, e ajuste da umidade de acordo com a sensação humana, resultando em uma maior taxa de conversão, indicativo de baixa produtividade.

1.3 OBJETIVO

1.3.1 Objetivo Geral

Construir e treinar um sistema de aprendizagem de máquina capaz de aprender utilizando leituras de sondas instaladas nos aviários e armazenadas em nuvem. Gerar receitas de ambiência através do conhecimento obtido pelo sistema de aprendizagem. As receitas geradas por estes sistemas serão comparados com as sugeridas nos manuais de manejo. Tal comparação tem o propósito de verificar teóricamente se tais soluções podem propiciar melhor produtividade às granjas.

1.3.2 Objetivos Específicos

1. Preparar dados contendo informações coletadas em aviários de frango de corte e armazenados em nuvem para serem utilizados por algoritmos de aprendizagem de máquina.
2. Empregar algoritmos de aprendizagem de máquina para estimar taxas de conversão alimentar baseados em receitas de ambiência.
3. Empregar algoritmos de busca para encontrar receitas de ambiência que minimizem a taxa de conversão alimentar.
4. Implementar as etapas de preparação de dados, aprendizagem e busca de forma modular, possibilitando que sejam executadas individualmente

1.4 JUSTIFICATIVA

Sistemas de aprendizagem de máquina alimentados com dados coletados automaticamente serão capazes otimizar as receitas de ambiência com o potencial de propiciarem ganhos de produtividade.

A automação da utilização de novos dados e do processo de aprendizagem tornará possível que as receitas de ambiência sejam continuamente aprimoradas, acompanhando os impactos que os esforços no campo da genética animal (TAVáREZ; SANTOS, 2016) possam ter sobre a taxa de conversão alimentar e as necessidades de ambiência das aves.

1.5 ORGANIZAÇÃO DO TEXTO

O chapter 2 inicia em sua Seção 2.1 com uma descrição geral das técnicas de manejo avícola e características mais relevantes do ponto de vista de produtividade. Os equipamentos comumente empregados no controle de um aviário de frangos de corte serão abordados na Seção 2.2.

Na Seção 2.3, será explorado como a cobertura de internet no campo contribui para a utilização em larga escala de sistemas como o proposto por este trabalho, bem como, para a viabilidade em disponibilizar as leituras de dados em nuvem para serem processados e consumidos pela cadeia de profissionais da gestão das granjas e seus sistemas de análise.

As técnicas de aprendizagem e busca, que são objeto desta pesquisa, constam na Seção 2.4 e Seção 2.5. Os estudos relacionados que inspiram esta pesquisa são analisados na Seção 2.6

O chapter 3, Seção 3.1, delimita o escopo desta pesquisa. Seção 3.2 e Seção 3.3 dão detalhes acerca da base de dados utilizada na aprendizagem e validação, incluindo características e disposição dos dados coletados pela Inobram Automações[2]. Os cuidados que precisam ser tomados ao organizar e selecionar estes dados para treinamento estão na Seção 3.4.

A Seção 3.5 detalha os procedimentos de aprendizagem, busca e otimização a serem desenvolvidos. Os meios de validação e avaliação dos resultados estão na Seção 3.6

Resultados deste trabalho, que incluem desde a engenharia de dados, aprendizagem de máquina, busca e otimização, podem ser encontrados no chapter 4

Discussões finais sobre os resultados e conclusões que puderam ser tiradas deste trabalho estão na chapter 5.

[2]Empresa de assessoria e serviços em automações, especializada em avicultura, suínocultura e bovinocultura, situada na cidade de Pato Branco, Paraná. www.inobram.com.br

2 REVISÃO DE LITERATURA

A seguir, serão apresentadas informações essenciais sobre o manejo de aviários de frangos de corte e principais características do animal que impactam em um ganho de produtividade, uma visão geral dos sistemas de controle de ambiência em uso nos aviários e como a internet está difundida no campo. Também, será pontuado como a computação em nuvem viabiliza este trabalho através da aprendizagem de máquina.

Serão aprensentados os tópicos de Aprendizagem de Máquina mais relevantes para este estudo. Problemas de otimização, como o das receitas de ambiência, são resolvidos através de algoritmos de busca e otimização, também serão apresentados.

2.1 AVICULTURA DE FRANGOS DE CORTE

Os aviários de frango de corte utilizam a taxa de conversão como um dos principais indicativos da produtividade do lote (LIMA; SIQUEIRA; ARAúJO, 1995). Lote corresponde a um período completo de alojamento das aves, medido em dias, do recebimento dos pintaínhos ao carregamento para o abate, com duração aproximada de 42 dias. Considera-se o término do lote quando as aves atingem um peso alvo para abate. Ao término do lote as aves são transportadas aos frigoríficos e o aviário é preparado para receber um novo lote(ROSS-AVIAGEN, 2018), (COBB-VANTRESS, 2018).

A taxa de conversão é a razão entre a ração consumida e o peso de aves vivas, segundo Ross-Aviagen (2018), quanto menor for a taxa de conversão, mais eficientes as aves foram em converter ração em peso vivo. É de interesse dos produtores que a taxa de conversão seja a menor possível, o que indica que o frango está ganhando peso adequadamente, além do fator comercial (ROSS-AVIAGEN, 2018), já que a carne é vendida por peso e quanto menor foi o investimento em alimentação e maior o peso final, maiores os lucros.

Além da genética animal, os parâmetros de ambiência como temperatura, umidade e pressão são fatores de grande impacto na taxa de conversão (COBB-VANTRESS, 2018). Estes parâmetros variam de acordo com a idade do lote, medida em dias. Em geral, as receitas - valores de temperatura, umidade e pressão para cada idade - são determinadas de forma empírica acompanhando a produtividade de lotes anteriores e baseado na experiência em manejo das empresas e pessoas envolvidas.

Todavia, manuais de manejo como os citados acima, recomendam que estas receitas sejam ajustadas de acordo com o clima local e experiência de manejo da equipe. Equívocos ou mesmo a falta de experimentação com variações nas receitas podem fazer com que o lote deixe de atingir melhores resultados. Na Table 1 estão as recomendações da Cobb-Vantress (2018), que acima de uma densidade de $28Kg/m^2$ indica a utilização de uma outra tabela baseada na densidade de aves alojadas. Esta densidade pode ser calculada levando em consideração o peso total aproximado das aves alojadas e as dimensões do galpão. Nesta tabela podemos perceber que nos primeiros dias de vida as aves necessitam de temperaturas maiores, porém no decorrer do lote estas temperaturas devem ser diminuídas.

2.2 AUTOMAÇÃO DO CONTROLE DOS AVIÁRIOS E ACESSO A INFORMAÇÕES DE AMBIÊNCIA

Equipamentos que facilitem o manejo dos aviários estão em contínuo desenvolvimento, dentre eles podemos citar os exaustores, que desempenham a ventilação forçada, renovando

Tabela 1 – Temperatura e Umidade Relativa.

Idade (dias)	30%	40%	50%	60%	70%
0	34°	33°	32°	31°	30°
7	32°	31°	30°	29°	28°
14	29°	28°	27°	26°	25°
$28kg/m^2$	25°	24°	23°	22°	21°

Fonte: Cobb-Vantress (2018)

o ar no interior dos galpões, removendo os excessos de amônia e gás carbônico produzidos; aquecedores, para que em dias frios a temperatura seja mantida agradável às aves; nebulizadores, para baixar a temperatura nos dias quentes, e umidificar o ar, entre muitos outros, criados para atender requisitos de manejo indicados em manuais da área avícola, como Ross Broiler Management Handbook (ROSS-AVIAGEN, 2018) e Cobb Broiler Management Guide (COBB-VANTRESS, 2018).

Inicialmente estes equipamentos, apesar de serem acionados manualmente pelos responsáveis pelo manejo das granjas, já contribuíram reduzindo o esforço necessário no manejo, por exemplo: Acionar manualmente um motor elétrico para abertura das cortinas. Ou seja, a ação não era executada por uma lógica de controle automatizada.

Conforme os microcontroladores/microprocessadores foram se tornando econônicamente acessíveis, surgiram equipamentos que monitoram sondas de leitura de ambiência (leituras de temperatura, umidade, entre outras) e com base nas configurações[1] definidas pelo produtor, atuam acionando os equipamentos mencionados. Com isto, não é mais necessário estar constantemente atento ao estado[2] das aves, basta estar atento às sirenes e alarmes que soam quando algo sai dos invervalos de segurança definidos, como por exemplo quando a temperatura ultrapassa determinado valor, circunstância na qual uma intervenção humana pode vir a ser necessária (ABREU; ABREU, 2011).

De modo a ter melhor controle sobre a troca de ar dentro do galpão, utiliza-se a técnica conhecida como 'pressão negativa', que consiste em permitir menos vasão de entrada de ar (passiva) do que a retirada pelos exaustores (forçada). Ao empregar pressão negativa, deve-se atentar para que a diferença de pressão com o meio externo não seja muito grande, pois isto pode causar problemas de saúde nas aves (BLANES-VIDAL; FITAS; TORRES, 2007).

2.3 INTERNET DAS COISAS E COMPUTAÇÃO EM NUVEM

Conceitualmente, o termo internet das coisas se refere a equipamentos (ou coisas) que possuem a capacidade de se conectarem à internet, tal conexão torna possível a troca de informações ou instruções entre sistemas e equipamentos.

Já as chamadas nuvens são serviços de computação distribuída onde a localização física dos servidores fica transparente ao usuário, pois o acesso aos serviços é disponibilizado via internet.

Nos últimos anos, a internet deixou de ser cara e restrita a grandes centros, tornando-se de melhor qualidade e menor custo de instalação, sendo instalada até nas propriedades rurais

[1] A variar de acordo com o controlador, geralmente incluem temperatura e umidade desejada para cada idade do lote(INOBRAM, 2020)

[2] Quando não há conforto térmico ou umidade adequada, as aves demonstram físicamente, ficando ofegantes, amontoadas, entre outros aspectos comportamentais(COBB-VANTRESS, 2018),(ROSS-AVIAGEN, 2018)

mais remotas (CETIC.BR, 2018).

Os custos de armazenamento e processamento centralizado dos dados, que outrora poderiam exigir altos investimentos tanto em equipamentos quanto em pessoal qualificado, hoje podem ser feitos em núvens como Google Cloud Platform, Microsoft Azure, Amazon Web Services, entre outros, a custos baixos, proporcionais à utilização e sem investimentos iniciais altos (AHMAD; ANDRAS, 2019), (SANTOS et al., 2018), (QUINTINO, 2019), (POP, 2016).

Utilizando controladores modernos que possuem conexão com a internet, os dados dos galpões não estão mais restritos ao espaço físico da granja, e podem ser enviados através da internet para nuvens processamento de dados, estando acessíveis a integradores, veterinários e produtores de proteína animal (INOBRAM, 2020).

Além disto, muitas destas nuvens já possuem implementações altamente eficientes para aprendizagem de máquina(POP, 2016), (AMAZON, 2020), (GOOGLE, 2020), (MICROSOFT, 2020).

2.4 APRENDIZAGEM DE MÁQUINA

Segundo (GéRON, 2019, p. 4), "Aprendizado de Máquina é a ciência (e a arte) da programação de computadores para que eles possam aprender com os dados". A definição mais clássica foi feita por (MITCHELL, 1997, p. 2): "Um programa de computador é dito aprender a partir de uma experiência E com respeito a uma classe de tarefas T e medida de desempenho P, se seu desempenho nas tarefas em T, segundo a medida P, melhora com a experiência E"

A área de aprendizagem de máquina é bastante ampla e em constante crescimento. Lenz (2020) utiliza o diagrama mostrado na Figure 1 para classificar a área de aprendizagem de máquina.

Conforme abordado mais adiante na Seção 3.3, a base de dados da empresa Inobram Automações, que será utilizada nesta pesquisa, consiste em um conjunto de séries temporais, cada uma contendo informações de um lote completo de produção, já rotulados[3] com parâmetros de saída como pesagens, mortalidade e consumo de ração, necessários para o cálculo da taxa de conversão alimentar, rótulo este contínuo.

Considerando estas características, os métodos de aprendizagem mais apropriados ao escopo desta pesquisa são do tipo Supervisionado (GéRON, 2019, p. 8) e será utilizada Regressão (GéRON, 2019, p. 9) para estimar a taxa de conversão do lote.

Nos estudos feitos por Klotz et al. (2020), realizados em dados cujas características eram similares e os objetivos os mesmos, a utilização de aprendizado supervisionado e regressão apresentaram resultados bastante satisfatórios e, por este motivo, serão empregados neste trabalho, para que baseado em outro conjunto de dados, consiga-se aproximar dos resultados encontrados por Klotz et al. (2020)

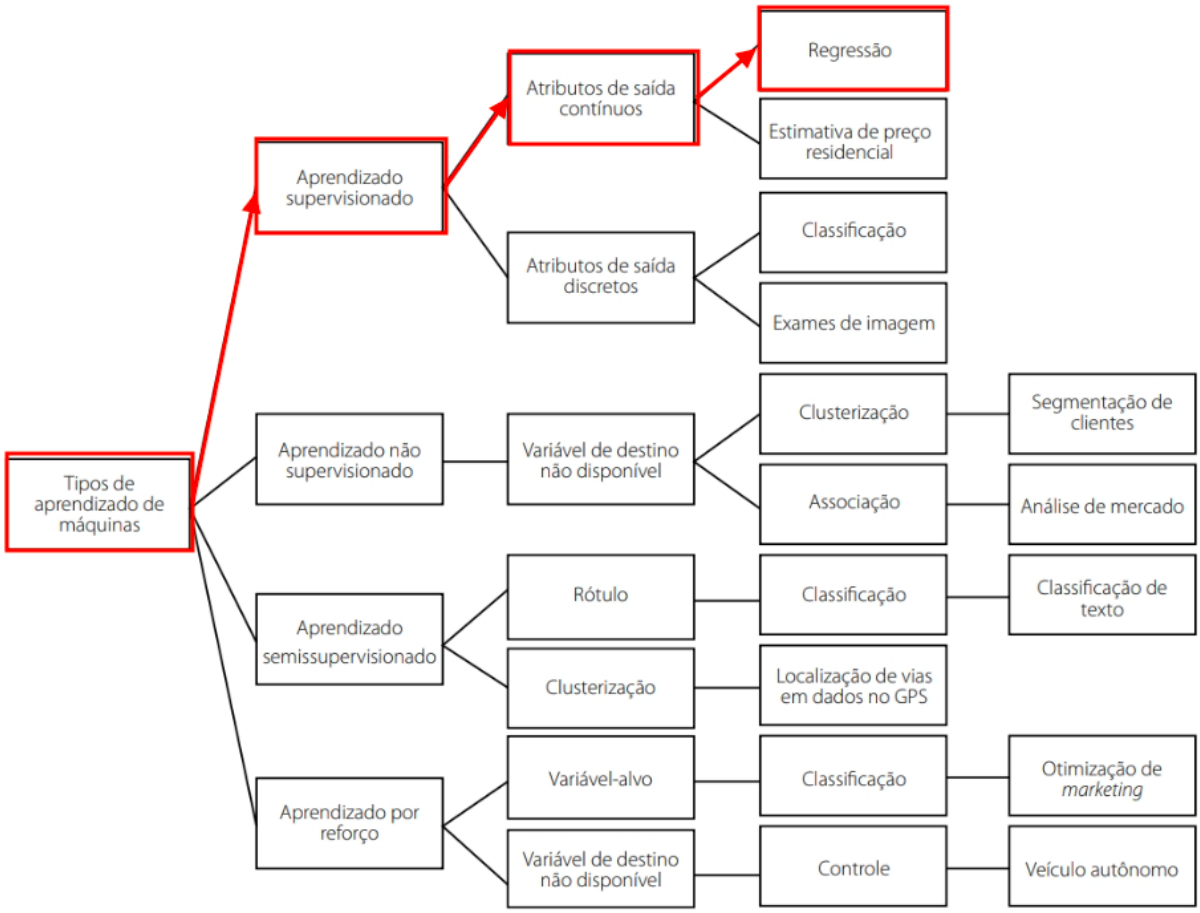
2.4.1 APRENDIZADO SUPERVISIONADO E REGRESSÃO

No aprendizado supervisionado, os dados de treinamento fornecidos ao algoritmo incluem a solução (também chamada de rótulo) (FACELLI et al., 2011). Deste modo, durante o processo de aprendizagem, a solução é conhecida.

Prever uma solução de valor contínuo é uma tarefa típica de aprendizado supervisionado, de acordo com Géron (2019, p. 9), que cita como exemplo a previsão do preço de um carro a partir de um conjunto de características, como quilometragem, idade, marca, etc. As caracterísiticas são chamadas de *previsores* e este tipo de tarefa, de fazer previsões, é chamada de *regressão*.

[3] Rótulo, em aprendizagem de máquina, é a solução desejada (GéRON, 2019)

Figura 1 – Classificações de aprendizagem de máquina



Fonte: Lenz (2020, p. 31)

O problema de prever rótulos contínuos como o valor do carro do exemplo anterior, é um problema de *regressão multivalorada*, visto que múltiplas características são necessárias para fazer a previsão (GéRON, 2019, p. 39).

2.4.2 REDES NEURAIS ARTIFICIAIS, REDES NEURAIS RECORRENTES E CÉLULAS LSTM

As Redes Neurais Artificiais são empregadas para fazer com que os sistemas computacionais adquiram a capacidade de aprender, foram inspiradas na arquitetura do cérebro para construir máquinas inteligentes, compostas por neurônios artificiais que ativam sua saída quando mais de um certo número de entradas estão ativas, podendo ser combinadas para calcular expressões lógicas complexas. (GéRON, 2019, p. 259, 263).

Redes Neurais Recorrentes são capazes de analisar dados em séries temporais e possibilitam antecipar resultados. Em seu livro, Géron (2019, 397) ilustra esta capacidade da seguinte forma: "Você pode pedir para prever quais serão as próximas notas mais possíveis de uma melodia".

Em relação às Redes Neurais Artificiais tradicionais, as Redes Neurais Recorrentes possuem em seus neurônios uma saída adicional que gera uma entrada para si mesmos, que pode ser considerada uma memória. Este efeito de memória se deve ao fato de que tal entrada recebe a saída da etapa de tempo anterior (GéRON, 2019).

Todavia, Redes Neurais Recorrentes não são capazes de aprender padrões de longo prazo devido, principalmente, a dois problemas:

1. Gradientes *vanishing/exploding*: Ocorrem quando as funções de erro das etapas anteriores decrescem ou aumentam muito rápido, fazendo tais etapas serem insignificantes, ou o algoritmo divergir da solução (GéRON, 2019, p. 283)
2. Desaparecimento da memória das primeiras entradas: "Depois de um tempo, o estado da Rede Neural Recorrente praticamente não contém traços das primeiras entradas"(GéRON, 2019, p. 419)

A célula LSTM (*Long Short-Term Memory*) resolve os dois problemas presentes nos neurônios das Redes Neurais Recorrentes (RNN[4]) citados anteriormente, através da adição de um segundo vetor de estado, correspondente a memória de longo prazo, enquanto o neurônio de uma RNN possui apenas um vetor de estado de curto prazo. (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 404), (GéRON, 2019, p. 421)

Estas características, de acordo com Goodfellow, Bengio e Courville (2016, p. 407) conferem às LSTM a capacidade de aprender mais fácilmente dependências de longo prazo.

2.5 ALGORITMOS GENÉTICOS PARA BUSCA E OTIMIZAÇÃO

Algoritmos genéticos são inspirados na biologia, para Mitchell (1996), Algoritmos Genéticos são aqueles compostos ao menos por: População de cromossomos, seleção de acordo com uma função de *fitness* (avaliação), cruzamento para produzir novas populações e mutação aleatória das novas gerações.

A função de avaliação determina o quanto aquele elemento da população é bom em atingir um objetivo.

De maneira simplista, o algoritmo genético é composto por três operadores:

- Seleção: Dentre uma população de elementos, seleciona os mais bem avaliados.
- *Crossover*: Cruza características entre dois elementos da população para gerar um novo elemento.
- Mutação: Modifica aleatóriamente características dos elementos da população

O algoritmo genético executa sequencialmente os operadores acima, cada execução da sequência é chamada de geração. A população resultante de uma geração é a entrada para a próxima geração.

Contextualizando com o tema deste trabalho, a população é composta por diversos planos de ação de ambiência. A avaliação deve definir quanto cada plano de ação é capaz de produzir uma baixa taxa de conversão alimentar. A otimização se dá em transformar os planos de ação para que sejam mais bem avaliados a cada nova geração.

Após a obtenção de um modelo capaz de prever a taxa de conversão alimentar de um lote a partir de valores de ambiência, sendo objetivo deste trabalho obter a menor taxa de conversão, gera-se a necessidade de encontrar combinações destes valores de ambiência capazes de resultar na menor taxa de conversão.

Consideradas temperatura e umidade como variáveis de entrada, quantidade de aves vivas, peso médio das aves e consumo de ração como variáveis de saída, a otimização se dará a partir de multiplas varáveis de entrada (multi-valorada) e múltiplos objetivos (muitiobjetivo).

Klotz et al. (2020, p. 5), ao analisar este problema, considerou algoritmos genéticos os melhores métodos, devido a sua capacidade de busca simultânea em diferentes regiões do espaço de soluções. Por este motivo, o presente trabalho adotará também algoritmos genéticos para busca e otimização.

[4]Do inglês, *Recurrent Neural Networks*

Todavia, comparativos de desempenho em problemas multiobjetivo como os conduzidos por Chiandussi et al. (2012, p. 941) atentam para o fato de que algoritmos genéticos tendem a exigir um grande esforço computacional.

O esforço computacional poderá futuramente motivar a substituição ou melhoramento dos algoritmos genéticos através da adição de técnicas como o operador transgênico proposto por Viana, Junior e Contreras (2020, p. 465), ou simplificar os objetivos prevendo a conversão alimentar diretamente ao invés dos elementos necessários para calculá-la

2.6 TRABALHOS RELACIONADOS

Em seu trabalho, Klotz et al. (2020) indica e põe à prova métodos de aprendizagem baseados em LSTM e métodos de busca e otimização utilizando algoritmos genéticos, obtendo uma melhora expressiva de 5% sobre os melhores resultados empíricos de manejo do seu grupo de estudo (KLOTZ et al., 2020, p. 14). Estes métodos serão utilizados como base no desenvolvimento desta pesquisa, uma vez que seu trabalho comparou o desempenho com uma grande variação de parâmetros e, sendo recente, contempla o que há de mais atual na área. Os métodos automatizados de obtenção e aprendizagem com novos dados serão um diferencial em relação a Klotz et al. (2020).

Ribeiro et al. (2019, p. 8), ao propor uma solução para o mesmo problema, obteve uma melhora de 2.6% na taxa de conversão em relação à receita gerada pelo especialista em manejo. O presente trabalho considera que a automação da aprendizagem é um dos passos necessários para justificar a construção de uma integração direta entre a saída dos sistemas de aprendizagem de máquina e os controladores de aviário, apontado por Ribeiro et al. (2019, p. 8) como uma forma de tornar mais prática sua solução de aprendizagem.

Siami-Namini, Tavakoli e Siami Namin (2018) comparou um dos métodos de aprendizagem mais tradicionais utilizados na previsão do tempo, ARIMA[5], com o método baseado em aprendizagem profunda (LSTM), concluindo que este último supera em performance os métodos tradicionais com até 87% de redução nas taxas de erros(Siami-Namini; Tavakoli; Siami Namin, 2018, p. 6). A utilização de LSTM será objeto de estudo do presente trabalho.

Lorencena et al. (2020, p. 15) ao propor um *framework* para modelar, controlar e supervisionar aviários, em suas considerações aponta a tendência de integração possível do sistema proposto em seu trabalho com a IoT[6]

[5]Do inglês, *AutoRegression Integrated Moving Average*

[6]Internet das coisas. Do inglês, *Internet of Things (IoT)*

3 PLANO DE AÇÃO PARA MAIOR PRODUTIVIDADE

Será apresentado a seguir um detalhamento de como a pesquisa e execução deste trabalho serão conduzidas de modo a obter parâmetros de ambiência capazes de melhorar a produtividade das granjas, uma visão geral da base de dados que alimentará a aprendizagem, bem como sugestões de como os resultados, caso satisfatórios, poderão ser utilizados. Serão mencionados na chapter 6 alguns aspectos relacionados não cobertos neste trabalho, porém com potencial de contribuírem para o objetivo de ganho de produtividade nas granjas.

3.1 DELINEAMENTO DA PESQUISA

Dadas as características citadas em Seção 2.1, e sua aplicação na Seção 2.4, o treinamento receberá leituras de temperatura, umidade e pressão aferidas durante lotes de diferentes galpões e, para o objetivo desta pesquisa não serão levadas em consideração as receitas configuradas nos controladores, mas apenas os reflexos das ações de controle nas variáveis mencionadas.

Será assumido que as ações de controle de ambiência são suficientes para neutralizar as interferências do clima externo no ambiente controlado, de modo a ignorar as sazonalidades climáticas. O clima externo tem um grande impacto nas ações de controle e consequentemente no estresse das aves, o que poderá ser explorado em trabalhos futuros. Pelo mesmo motivo, será assumido que os modelos específicos de atuadores utilizados em cada galpão (exaustores, umidificadores, etc) são irrelevantes para as medidas de ambiência.

Os resultados esperados desta pesquisa serão validados utilizando projeções das taxas de conversão e comparadas com recomendações de manuais de manejo e com dados de lotes anteriores.

Serão considerados também, nos métodos de aprendizagem, quais podem ser automaticamente treinados com novos dados, sem necessitar que todo o conjunto de dados seja reprocessado.

3.2 OBTENÇÃO DOS DADOS E AMBIENTE DE DESENVOLVIMENTO

Controladores modernos capazes de coletar os dados e enviar via internet, para um sistema em nuvem, automatizam a coleta dos parâmetros de ambiência, conforme visto na Seção 2.2 e Seção 2.3. Para esta pesquisa a empresa Inobram Automações disponibilizou o acesso a sua base de dados de ambiência, contendo os dados que detalharemos a seguir, na Seção 3.3.

Esta pesquisa irá cobrir o acesso a base de dados já em nuvem e a algumas técnicas necessárias para extrair da base apenas as informações de interesse.

Os dados encontram-se em um banco de dados PostgreSQL, serão acessados em um ambiente de desenvolvimento Python através do adaptador 'psycopg2', utilizado na forma de biblioteca.

A base de dados utilizada nesta pesquisa encontra-se em desenvolvimento e portanto algumas estruturas de dados poderão sofrer ajustes e modificações ao longo deste trabalho, parâmetros adicionais poderão ser considerados.

3.3 VISÃO GERAL DOS DADOS

As características, como temperatura, umidade, pressão, etc. são muito importantes na escolha dos métodos de aprendizagem a serem utilizados. Diferentes características de cada base de dados deverão ser consideradas ao replicar este trabalho.

O banco de dados que será empregado contém leituras de 26 galpões (também chamados de aviários), com uma média de 3 lotes por galpão, totalizando 78 séries de dados, dispostos da seguinte forma:

- Parâmetros de ambiência: Série temporal contendo leituras de minuto a minuto da temperatura média, umidade média, pressão, peso do silo de ração, data e hora de aferição, idade do lote e identificação do galpão.
- Análises de pesagem: Série temporal contendo cálculos diários de peso médio, ganho e desvio padrão das pesagens, além da data à qual a análise se refere.
- Mortalidade diária de cada galpão.
- Intervalo de datas que caracteriza cada lote para cada galpão.
- Quantidade de aves alojadas de cada galpão no início de cada lote.

3.4 PREPARANDO OS DADOS PARA APRENDIZAGEM DE MÁQUINA

Esta pesquisa considera que a preparação dos dados para serem utilizados nos sistemas de aprendizagem são uma etapa fundamental para se automatizar a realimentação do sistema com novos dados. Assim, as dificuldades e ajustes necessários a fazer nos dados serão descritas.

Serão feitas consultas e seleções dos dados a fim de remover inconsistências. Um exemplo de inconsistência para os dados de aprendizagem é uma característica da sonda de temperatura cuja medição está presente nesta base de dados: Este modelo de sonda envia temperatura 0°C para o banco de dados caso uma falha de equipamento impossibilite a aferição da temperatura. Uma medição como a exemplificada deverá ser descartada do conjunto de dados para não afetar negativamente a aprendizagem.

Para organizar as séries temporais, os dados de cada galpão serão agrupados, e então organizadas as leituras de ambiência por lotes, baseado na data de coleta de cada registro em relação às datas de início e fim de lote. Desta forma, cada lote de cada galpão será considerado uma série de dados.

3.5 APRENDIZAGEM DE MÁQUINA, BUSCA E OTIMIZAÇÃO

O conjunto de séries temporais obtido na etapa de preparação irá alimentar uma *pipeline* responsável pela apredizagem, que então será capaz de prever a taxa de conversão para qualquer série.

Redes Neurais Recorrentes com células LSTM serão utilizadas para a aprendizagem. A escolha foi baseada nos bons resultados obtidos por (KLOTZ et al., 2020) e mencionados na Seção 2.6, e na flexibilidade quanto a sequências de entrada e saída (GéRON, 2019, p. 401).

A solução escolhida para aprendizagem indica ser mais promissora do que os algoritmos testados por Ribeiro et al. (2019, p. 6), sendo eles *Levenberg-Marquardt*, *Scaled Conjugate Gradient* e *Bayesian Regularization*. Destes *Bayesian Regularization* obteve o melhor desempenho em reduzir a taxa de conversão, sendo 2,6% menor do que a sugerida pelo especialista.

Um sistema de busca será empregado para determinar quais características destas séries causam maior ganho na produtividade. A busca utilizará Algoritmos Genéticos, como proposto por Klotz et al. (2020, p. 5).

Conforme discutido na Seção 2.5, considerando o potencial crescimento da base de dados, e o objetivo de que o aprendizado esteja sendo constantemente melhorado, caso o método de busca mostre-se muito demorado ou consumindo muitos recursos, este poderá ser substituído.

3.6 VALIDAÇÃO DOS RESULTADOS

A receita de ambiência gerada pelo sistema proposto será comparada às recomendações dos manuais de manejo Cobb-Vantress (2018) e Ross-Aviagen (2018), de forma a validar se são condizentes com as práticas de manejo recomendadas.

A receita gerada será então submetida ao sistema de regressão para que a taxa de conversão possa ser estimada.

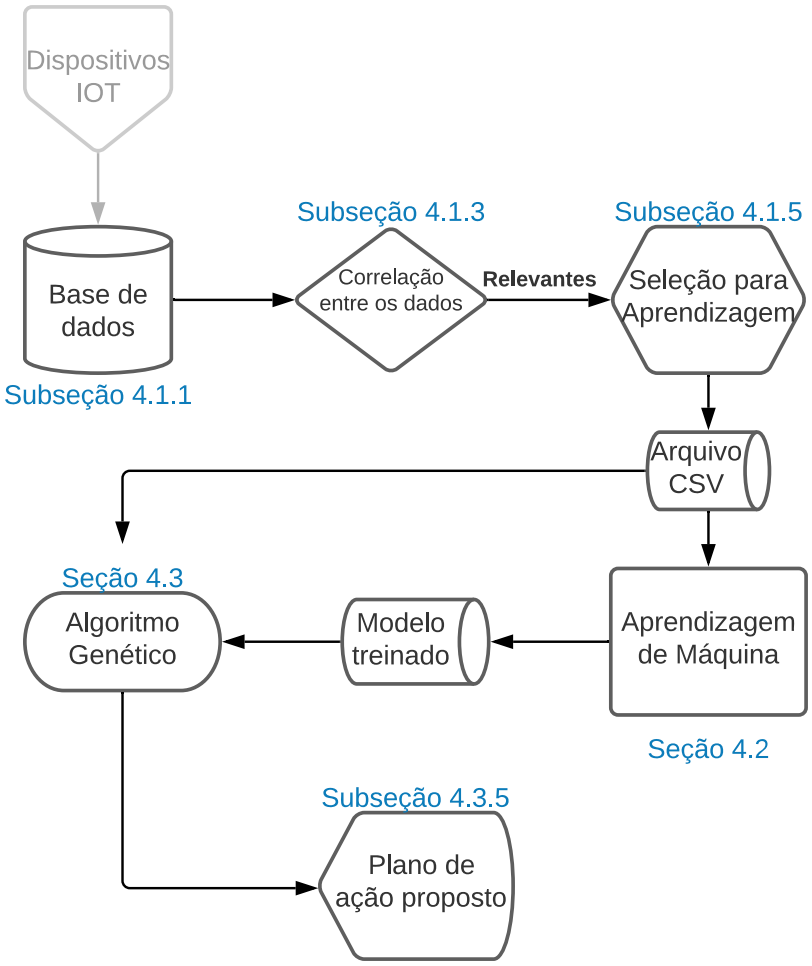
Por fim, a taxa de conversão estimada será comparada com a melhor taxa de conversão presente na base de dados. Será exibida na sua forma percentual, facilitando sua comparação com os resultados obtidos por Ribeiro et al. (2019) e Klotz et al. (2020)

4 PRINCIPAIS RESULTADOS

A obtenção dos planos de ação de temperatura, umidade e pressão envolveram etapas de seleção de características e engenharia de dados, detalhados na Seção 4.1, o processo de aprendizagem está descrito na Seção 4.2, resultando em um modelo de aprendizagem empregado a um algoritmo genético, abordado na Seção 4.3 para encontrar candidados a planos de ação cuja conversão alimentar fosse a menor possível. Os dados utilizados neste trabalho estavam originalmente armazenados em banco de dados relacional. Durante a etapa de engenharia de dados, estes dados foram acessados no Python, e transformados em um DataFrame Pandas, cujos recursos são ideais para manipulação, organização, seleção e distribuição dos dados. Selecionar as características mais apropriadas para o objetivo deste trabalho se fez essencial, pois apenas uma pequena parcela das características disponíveis no banco original, que já possuía mais de 10GB, eram relevantes para determinar a conversão alimentar a partir da temperatura, umidade e pressão.

A figura Figure 2 aponta em preto a ordem de realização das etapas realizadas () e indica em quais seções e subseções estas etapas serão abordadas.

Figura 2 – Fluxograma das etapas realizadas



Fonte: Autoria própria

4.1 ENGENHARIA DE DADOS

Nesta seção será detalhado o processo de escolha das características da base de dados, e maneiras eficientes de acessá-las para fins de treinar um modelo de Aprendizagem de Máquina.

A implementação completa das etapas descritas a seguir encontram-se disponíveis nos Appendix A e Appendix B

4.1.1 CARREGAMENTO DOS DADOS DE BANCO DE DADOS RELACIONAL

O banco em questão, PostgreSQL, possuía os dados brutos, a fim de realizar uma análise exploratória e encontrar as informações relevantes, uma primeira abordagem foi, em. uma única consulta SQL, reunir temperatura, umidade, pressão, idade do lote, peso médio das aves, quantidade de aves vivas, consumo de ração, transformar todos estes dados em uma *Materialized View*[1], para que, uma vez agregados, os dados pudessem ser facilmente acessados de um único local, tornando as consultas subsequentes menores e mais simples.

Todavia esta abordagem se mostrou pouco eficiente, pois a consulta para criar a *Materialized View* envolvia muitas tabelas, ficando demasiada complexa e devido a isto, outra abordagem foi utilizada, criando múltiplos DataFrames, cada DataFrame focado em preparar um tipo de informação, como por exemplo o peso das aves. Esta abordagem será explicada mais a frente.

4.1.2 DISTRIBUIÇÃO DOS DADOS

Para um melhor entendimento do conteúdo dos dados a serem trabalhados, foi gerado um histograma, exibido na Figure 3. Este tipo de visualização também possibilita identificar alguns *outliers*, dados que estejam muito distantes das regiões de maior incidência. O histograma foi obtido utilizando a função hist() do Pandas DataFrame.

A partir do histograma, algumas características puderam ser percebidas:

- As temperaturas variam entre 16 e 35 graus, sendo que a maior incidência ocorre entre 24 e 27 graus. Condiz com o indicado na literatura para a fase adulta dos frangos.
- A umidade está concentrada acima de 40% até 90%, devido fortemente ao emprego de placas evaporativas para resfriar os galpões.
- A maior porcentagem de pesagens encontra-se na categoria leve, ou seja um pouco abaixo da média
- Existe uma quantidade muito maior de pesagens quando as aves estão mais jovens (mais leves), pois quanto maiores as aves ficam, mais restrito fica o espaço disponível para que elas se movam e subam nas balanças. Frangos mais velhos também tendem a ser menos ativos.

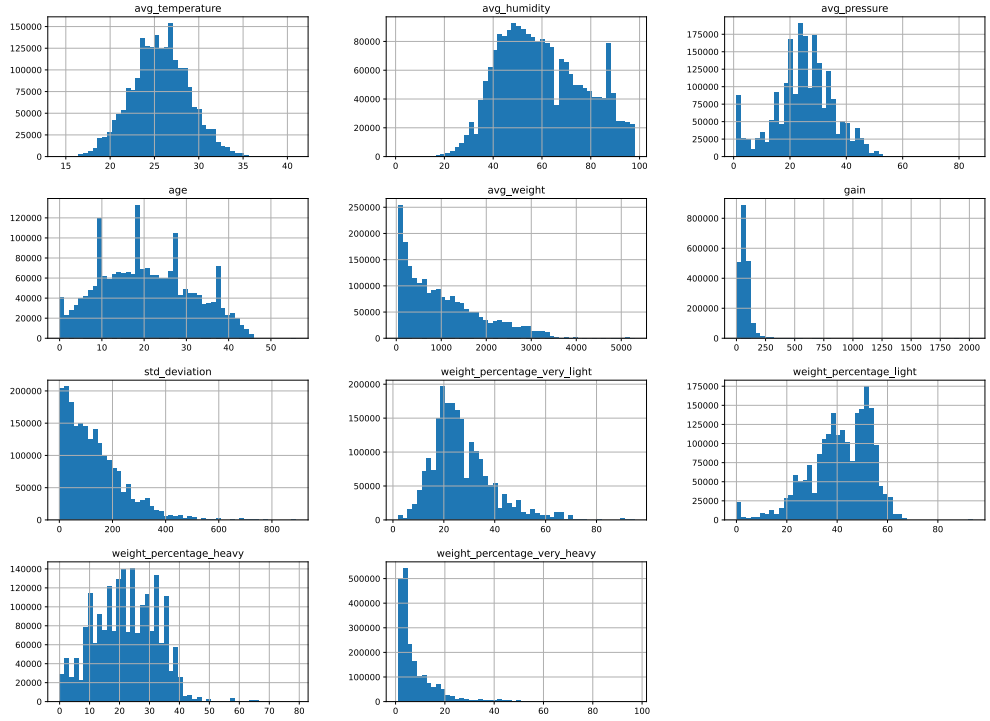
4.1.3 CORRELAÇÃO ENTRE OS DADOS

Entender como os dados estão correlacionados contribui na seleção das características a serem empregadas na aprendizagem de máquina, pois através das correlações é possível definir quais características podem impactar na previsão de uma variável de interesse, que no caso deste trabalho é a taxa de conversão alimentar.

A figura Figure 4 apresenta a matriz de dispersão construída utilizando a função scatterMatrix da biblioteca Pandas Plotting:

[1] Objeto de uma base de dados que representa o resultado de uma consulta, se assemelha a uma tabela

Figura 3 – Histograma dos dados



Fonte: Autoria própria

Na matriz de dispersão, linha 2, coluna 1, pode-se perceber que com o avanço da idade a temperatura tende a ser menor, o que condiz com a literatura. Quando as aves são mais jovens elas precisam de temperaturas mais altas para terem um bom desenvolvimento quando forem adultas.

Matrizes de dispersão também foram geradas para outras características, como peso médio, ganho de peso, desvio padrão, porém não foram observados resultados relevantes.

O cálculo do Coeficiente de Correlação Padrão, através do emprego da função corr() do Pandas Dataframe, para computar a correlação entre os atributos, resultou na Table 2:

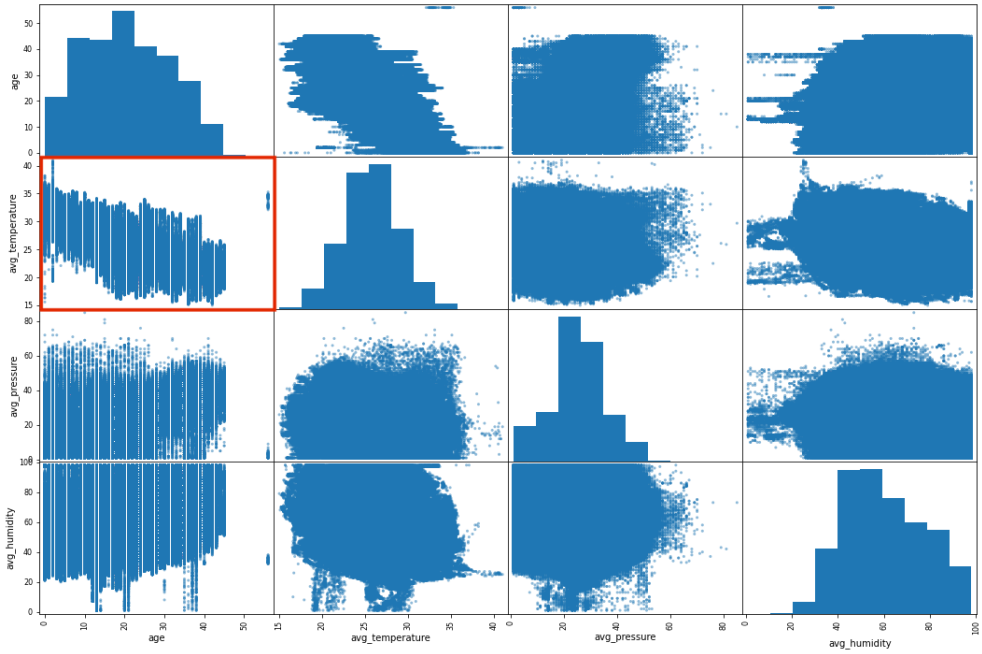
Tabela 2 – Correlação.

Parâmetro	Muito Leve	Leve	Pesado	Muito Pesado
Pressão	0.141938	0.169357	-0.188959	-0.186872
Umidade	0.061792	-0.016987	-0.054838	-0.031204
Temperatura	-0.153559	-0.335245	0.251622	0.348383

Fonte: Autoria própria

Evidenciaram-se algumas características:

Figura 4 – Matriz de Dispersão



Fonte: Autoria própria

- Correlação negativa da temperatura média, e positiva da pressão em relação a porcentagem de aves "muito leves";
- Correlação positiva da temperatura média e negativa da pressão em relação a porcentagem de aves "pesadas"e "muito pesadas".

Fazendo um paralelo destas correlações com as orientações presentes em (COBB-VANTRESS, 2018), (ROSS-AVIAGEN, 2018), que enfatizam que baixas temperaturas, principalmente no início do lote, prejuditam o desenvolvimento da ave adulta, pode-se assumir que a diminuição da temperatura faz com que a porcentagem de aves "muito leves"aumente, indicando que temperaturas muito baixas prejudiquem a uniformidade de pesos, aumentando a porcentagem de aves cujo peso está muito abaixo da mediana de peso daquele dia.

Não foi encontrada na literatura consultada neste trabalho, informações que indicassem efeitos prejudiciais do aumento de pressão para a uniformidade de peso das aves. Segundo (COBB-VANTRESS, 2018), um galpão mal selado faria com que a pressão não fosse suficiente para realizar a ventilação mínima, ocasionando aumento de dejetos, resfriamento das aves, maior estresse, mortalidade, conversão alimentar e gastos com aquecimento.

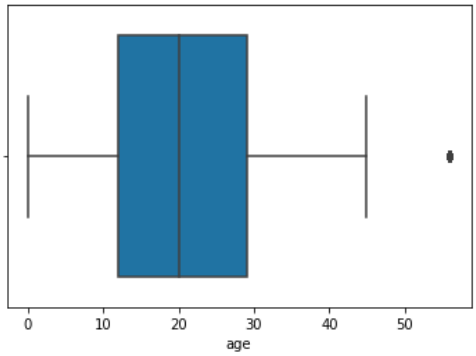
4.1.4 DETECÇÃO DE *OUTLIERS*

Para encontrar valores que estivessem muito distantes dos valores mais frequêntes da base de dados, e portanto, candidatos a serem *outliers*, utilizou-se da biblioteca Seaborn, gráfico do tipo *boxplot*, para todas as características. Destas, destacam-se duas, idade e temperatura, apresentadas na Figura 5 abaixo:

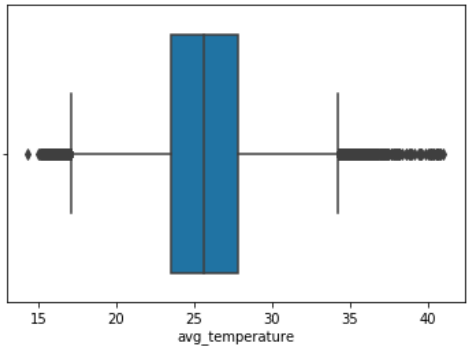
Observaram-se que a maioria das idades vai até 45 dias, com alguns outliers. Em

Figura 5 – Boxplot

(a) Boxplot da Idade



(b) Boxplot da Temperatura



Fonte: Autoria própria

conversa com a equipe de campo da Inobram Automações, constatou-se que estas idades são registradas quando o produtor mantém o controlador ligado mesmo após o galpão ter sido desalojado, assim, idades que ultrapassem os 50 dias seguramente podem ser consideradas outliers.

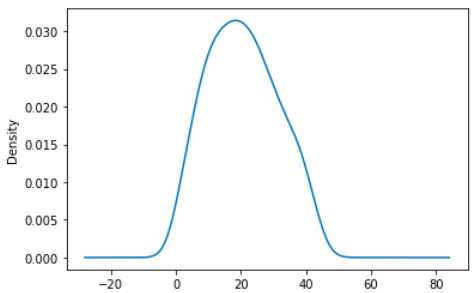
Percebe-se algo similar no gráfico *boxplot* para a temperatura, com valores abaixo de 25 e próximos de 40 graus. Também estes são registrados quando o controlador permanece ligado durante o período em que o galpão está desalojado, porém com o painel elétrico que alimenta os atuadores desligado, fazendo que não haja o controle, apenas o registro das leituras.

O método *Z-Score* Foi aplicado a estes dados para tratamento dos outliers, porém esta abordagem não foi levada a diante, optou-se por selecionar apenas lotes que se enquadrassem dentro de alguns critérios, que serão explicados com mais detalhes na próxima subseção, onde trataremos da aplicação da aprendizagem de máquina.

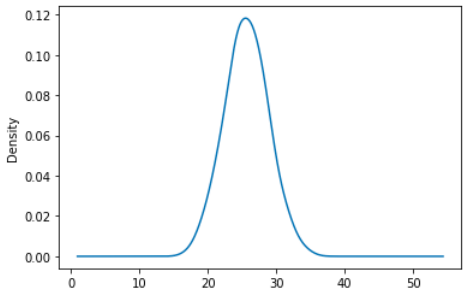
Foram geradas também as distribuições gaussianas de densidadade das variáveis através dos gráficos KDE (*Density Estimators*) do Pandas Dataframe, porém nenhuma informação adicional foi constatada. A critério de comparação com os boxplots da Figura 5, estão exibidas na Figura 6 as distribuições Gaussianas:

Figura 6 – KDE Plot

(a) Gaussiana da Idade



(b) Gaussiana da Temperatura



Fonte: Autoria própria

Vê-se que proporcionalmente a quantidade de outliers não é significativa a ponto de ser visível nas distribuições gaussianas.

4.1.5 SELEÇÃO DOS DADOS PARA APRENDIZAGEM DE MÁQUINA

A fim de fornecer ao modelo de aprendizagem de máquina dados capazes de gerar a previsão da conversão alimentar, alguns critérios foram utilizados:

- Apenas lotes que já foram finalizados foram considerados: Para que existam dados de todas as idades do lote.
- A quantidade de aves alojadas tenha sido fornecida: Esta é uma informação que o usuário precisa fornecer manualmente ao sistema, em alguns lotes esta informação não foi inserida.
- A mortalidade diária tenha sido fornecida: Assim como a quantidade de aves alojadas, a mortalidade precisa ser inserida pelo usuário no sistema, e alguns lotes não possuem esta informação. Ela é necessária para contabilizar a quantidade de aves vivas em cada idade de lote, com isto calcular o consumo de ração por ave e consequentemente a conversão alimentar.
- O peso dos silos de ração tenha sido registrado: O cálculo de consumo de ração é feito através da variação do peso dos silos. Nem todos os galpões possuem sistema de pesagem de silos.
- O peso das aves tenha sido registrado: Assim como a pesagem de silos, nem todos os galpões possuem sistema de pesagem de aves automatizado.

Após filtrar os 266 lotes presentes na base de dados através dos critérios acima, restaram 59 lotes que foram utilizados na aprendizagem de máquina, ou 22% dos lotes da base de dados.

As subseções a seguir, de preparação de dados, foram realizadas considerando apenas os lotes selecionados acima.

Para cada característica foi realizada uma consulta SQL no banco de dados relacional, e este resultado foi armazenado em um Pandas Dataframe. Ao final, um dataframe geral foi criado unindo os dados de cada dataframe específico. Os índices utilizados para associar os dados foram o identificador do lote e a idade do lote.

Os códigos-fonte implementados no Jupyter Notebook estão presentes no apêndice Appendix B, incluindo informações mais detalhadas

4.1.5.1 Peso médio das aves

O peso médio das aves está presente na base de dados já calculado, em gramas, por dia.

Ele é calculado pelo controlador fazendo a média simples de todos os pesos coletados durante o dia pelas balanças automáticas para aquele lote.

Ao carregar os dados para um dataframe, o peso foi convertido de gramas para kilogramas, para ficar na mesma unidade do peso do silo, medido em kilogramas.

Uma curva representando o peso médio das aves durante um lote pode ser visto na Figure 7

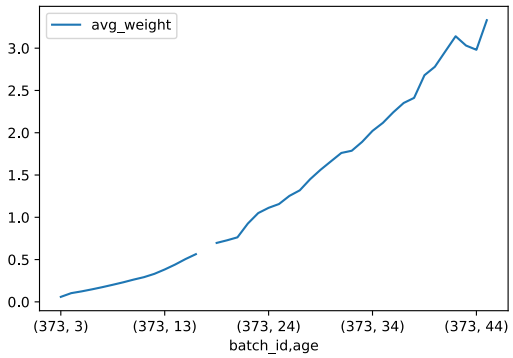
Períodos sem pesagens podem ocorrer por motivos como falhas nas balanças, manutenção dos equipamentos ou configuração inadequada de calibração.

Foi selecionado o id do lote, a idade, o peso médio e a data a que esta idade se refere, sendo que esta última apenas para conferência no momento de unir os *dataframes*.

4.1.5.2 Quantidade de aves vivas

Estão disponíveis na base de dados a quantidade de aves alojadas no início do lote e a mortalidade diária.

Figura 7 – Peso médio das aves durante um lote



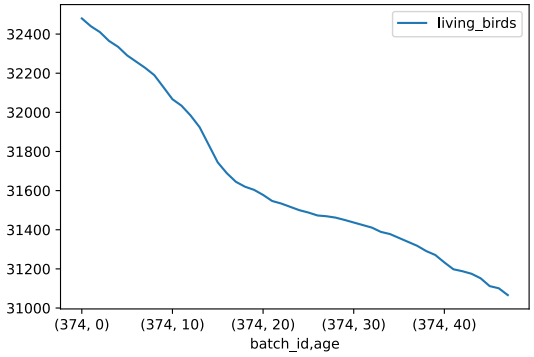
Fonte: Autoria própria

Para ser possível o cáculo da conversão alimentar em qualquer idade do lote, é necessário saber quantas aves estão vivas consumindo ração em dada idade.

A ração consumida pelas aves que morreram no decorrer do lote acaba impactando negativamente a conversão alimentar, portanto é desejável que este impacto negativo seja contabilizado, desta forma, quanto maior for a mortalidade, mais alta será a conversão alimentar e consequentemente pior será a avaliação deste lote

A curva da Figure 8 mostra a quantidade de aves vivas durante um lote qualquer.

Figura 8 – Quantidade de aves vivas durante um lote



Fonte: Autoria própria

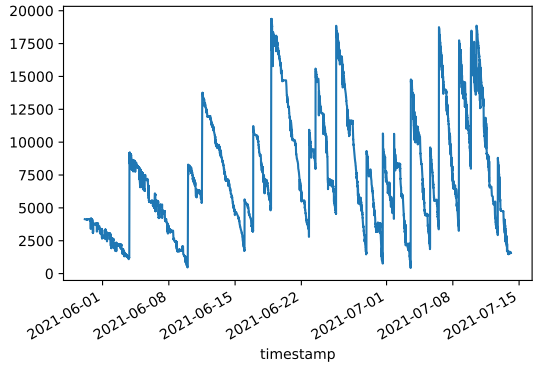
4.1.5.3 Consumo de Ração

A base de dados possui leituras de peso do silo minuto a minuto, conforme demostrado na Figure 9

O cálculo do consumo de ração baseou-se em somar as diferenças de peso entre uma leitura e outra. Caso a leitura recente fosse menor que a leitura anterior, considerou-se consumo de ração, caso fosse maior, considerou-se abastecimento do silo.

Os consumos de ração podem ser observados na Figure 9 nas curvas descendentes, já os abastecimentos podem ser observados nas retas verticais.

Figura 9 – Peso do silo durante um lote



Fonte: Autoria própria

A soma dos decrescimentos do peso do silo foi comparada com planilhas obtidas com produtores que mantém registro do consumo de ração baseado nos abastecimentos dos silos, e o peso do silo antes do abastecimento. Nesta comparação, a soma dos decrescimentos de peso se mostrou muito maior do que o consumo de ração.

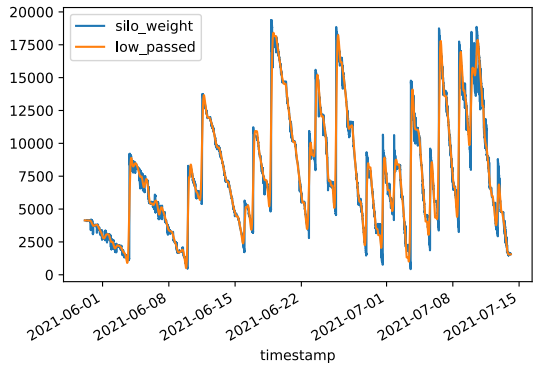
Com o objetivo de filtrar os dados para se aproximar do valor real consumido, algumas abordagens foram experimentadas:

A primeira delas foi aplicar um filtro passa-baixa do tipo Butterworth, para eliminar o efeito causado pela ação do vento no silo, o que pode fazer o peso oscilar para mais e para menos.

Mais detalhes sobre a aplicação do filtro podem ser encontrados no Appendix A na seção Redução de Ruídos

A Figure 10 demonstra a sobreposição da curva de peso do silo, em azul, pela curva após a aplicação do filtro Butterworth, em laranja.

Figura 10 – Sobreposição das curvas de peso com e sem filtro

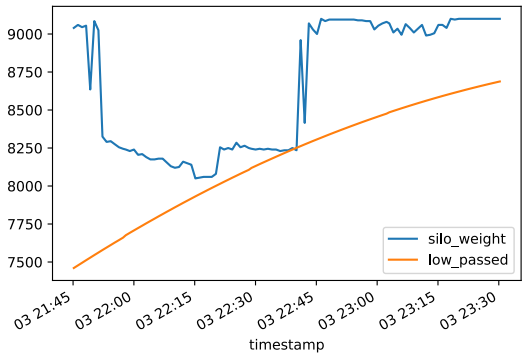


Fonte: Autoria própria

Diferentes ordens e frequências do filtro foram experimentadas, porém persistindo a discrepância no consumo.

Aproximando o gráfico de uma região menor, anomalias foram percebidas que não puderam ser resolvidas pelo filtro passa-baixas, como o caso exibido na Figure 11.

Figura 11 – Detalhe da anomalia no peso



Fonte: Autoria própria

Desconhecida a causa de anomalias como a da Figure 11, e considerando que uma variação tão significativa, na ordem de 100Kg/min não ocorre na prática.

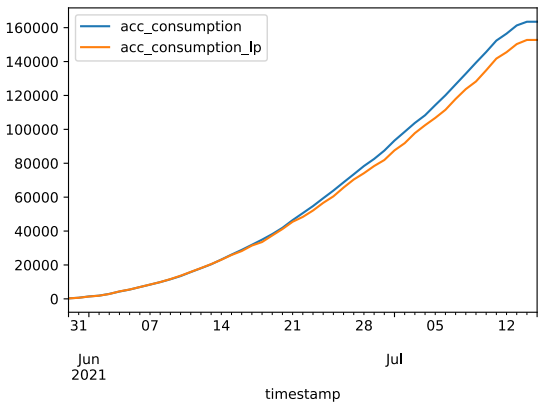
Partiu-se para uma alternativa mais simples ao filtro, considerar apenas os consumos inferiores a 100Kg/min.

O ajuste do ponto de corte de 100Kg/min foi feito utilizando como critério de comparação planilhas obtidas com produtores que mantém registro do consumo de ração baseado nos abastecimentos dos silos, e o peso do silo antes do abastecimento.

Esta regra simples trouxe resultados bastante satisfatórios, com um consumo de ração condizente com os consumos registrados nas planilhas dos produtores. O erro médio entre o consumo calculado e o considerado pelos produtores ficou em 3,8% sem o filtro Butterworth, a aplicação deste filtro na melhor configuração testada apenas aumentou o erro, motivo pelo ele não foi empregado nos dados que alimentaram o sistema de aprendizagem de máquina.

A Figure 12 mostra em azul a curva de consumo de ração acumulativa, considerando apenas variações menores que 100Kg/min, e a curva em laranja mostra o consumo acumulado após aplicado o filtro Butterworth.

Figura 12 – Consumo acumulado de ração para um lote



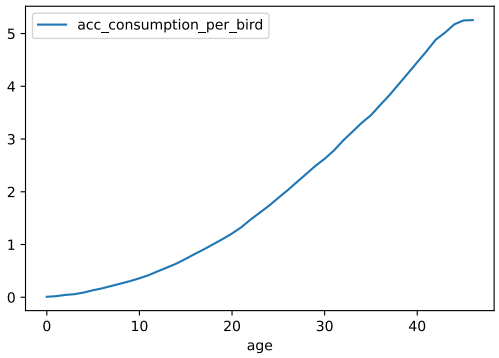
Fonte: Autoria própria

O consumo acumulado de ração será será crucial para calcular a conversão alimentar,

dividindo o consumo acumulado de ração pelo peso das aves vivas.

Dividindo o consumo acumulado de ração por dia, pela quantidade de aves vivas em cada dia calculado na Subsubseção 4.1.5.2, dividindo um pelo outro obtém-se o consumo acumulado de ração por ave. A Figure 13 demonstra este resultado.

Figura 13 – Consumo acumulado de ração por ave, em um lote



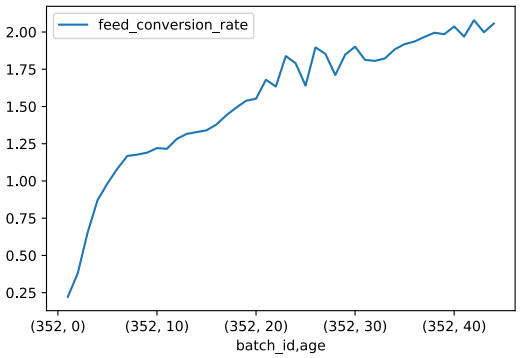
Fonte: Autoria própria

4.1.5.4 Conversão Alimentar

Após a obtenção da quantidade de aves vivas por dia, na Subsubseção 4.1.5.2 e o consumo acumulado de ração por ave na Subsubseção 4.1.5.3, podemos calcular a conversão alimentar ao longo do lote, dividindo um valor pelo outro. No Pandas Dataframe operações deste tipo, que consiste em dividir uma coluna pela outra, pode ser feito de maneira prática e eficiente, mais detalhes podem ser encontrados nos Appendix A e Appendix B, na seção Conversão Alimentar.

Um exemplo do comportamento da conversão alimentar pode ser observado na Figure 14.

Figura 14 – Conversão alimentar durante um lote



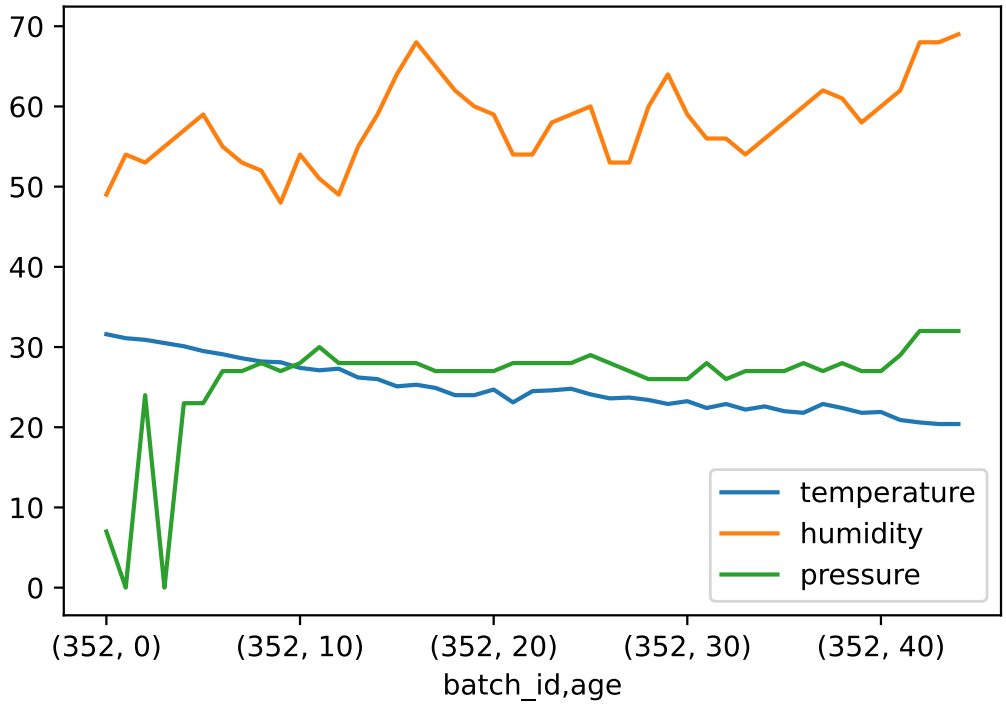
Fonte: Autoria própria

4.1.5.5 Ambiência

Os parâmetros de ambiência que foram empregados na aprendizagem de máquina são temperatura, umidade e pressão, considerados características de entrada da aprendizagem de máquina cuja saída é a conversão alimentar. Na base de dados eles são amostrados minuto a minuto, sendo que é comum um galpão possuir mais de uma sonda de temperatura ou umidade ao longo de sua extensão, de modo que o valor presente na base de dados é a média das sondas de cada característica.

A granularidade minuto a minuto destes parâmetros não é útil ao objetivo deste trabalho, pois não seria viável ou factível configurar planos de ação minuto a minuto nos controladores de ambiência. A granularidade foi transformada em um valor por dia, através da subamostragem por média dos valores minuto a minuto. Mais detalhes deste procedimento podem ser encontrados no Appendix B na seção Ambiência Média do Dia. As três características podem ser vistas na Figure 15, com temperatura em graus celcius, umidade em porcentagem e pressão em Pascais.

Figura 15 – Ambiência típica de um lote



Fonte: Autoria própria

4.1.5.6 Unificação dos dados e modularização

Todos dos dados preparados da Subsubseção 4.1.5.1 até a Subsubseção 4.1.5.5 foram unificados em um único Pandas Dataframe tendo como primeiro índice o identificador do lote e como segundo índice a idade do lote, ficando no formato representado na Table 3

Tabela 3 – Dataframe final.

Lote	Idade	...	Temp.	Umid.	Pressão	Qtd Aves	Cons. Acum Ave	Conv. Alim.
246	0	...	33.3	54.0	26.0	33980	NaN	NaN
246	1	...	31.8	65.0	0.0	33780	0.009473	0.135329
246	2	...	31.0	76.0	0.0	33629	0.023343	0.259366
246	3	...	30.6	76.0	18.0	33507	0.042678	0.402619
...	...	...	...	...	...	...	...	...

Fonte: Autoria própria

Para trazer mais modularidade ao projeto, optou-se por exportar o *dataframe* para o formato CSV[2]. Desta forma o arquivo de saída é relativamente pequeno, conterá apenas os dados a serem empregados na aprendizagem de máquina e otimização. Exportar para CSV e carregar de um arquivo CSV é bem suportado pelo Pandas. No caso deste trabalho, a base de dados original que possui mais de 10GB, após selecionados os dados de interesse e exportado para arquivo CSV resultou em um aquivo de saída de 200KB. Devido ao tamanho reduzido este arquivo pode ser facilmente importado nas etapas seguintes e, desde que mantida a estrutura dos dados, as etapas de engenharia de dados podem ser refeitas, e um novo arquivo CSV ser gerado, sem a necessidade de reescrever as etapas seguintes de aprendizagem e otimizaçao.

As etapas tanto de exportar quanto importar o CSV podem ser vistas no final do Appendix B.

4.2 APRENDIZAGEM DE MÁQUINA

Para a aprendizagem de máquina foi empregada a API Keras da biblioteca Tensorflow devido a vasto material sobre estas soluções, sendo referência na área. Também foi empregada a biblioteca Scikit Learn para funções de normalizar os dados e separar os conjuntos de treino e teste.

Utilizando o conceito de modularização esta etapa inicia importando o *dataframe* do arquivo CSV exportado na etapa de Engenharia de Dados descrita na Subsubseção 4.1.5.6.

Uma vez carregado o dataframe são definidos os índices primário como sendo o identificador do lote e secundário como sendo a idade do lote.

4.2.1 VISÃO GERAL DOS DADOS PARA APRENDIZAGEM

O arquivo CSV gerado possui algumas colunas que não são de interesse imediato da aprendizagem ou otimização, como data, porém foi decidido por incluí-la no arquivo para dar contexto aos dados e possibilitar que implementações futuras (não cobertas por este trabalho) pudessem explorar elementos como sazonalidade das estações do ano, por exemplo. No contexto desta etapa de aprendizagem foi desconsiderada a coluna data.

Consumo acumulado de ração e quantidade de aves vivas, também foram desconsideradas, pois já possuímos na tabela a conversão alimentar como saída do modelo de aprendizagem, que foi calculada a partir do consumo de ração e quantidade de aves.

Logo nas primeiras experimentações com a aprendizagem de máquina percebeu-se que a característica de idade do lote tem um impacto positivo bastante satisfatório na aprendizagem,

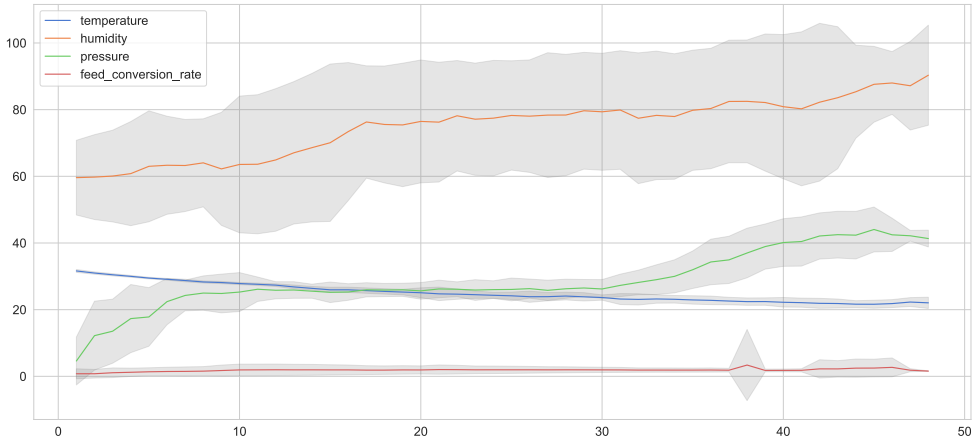
[2] *comma-separated values*, formato no qual os valores são separados por virgula, amplamente utilizado para armazenar dados estruturados no formato de tabela, como é o caso de um *dataframe*

e como a idade do lote foi utilizada como índice, para que pudesse também ser utilizada como caracterísica, foi gerada uma nova coluna, cópia da idade de lote, para ser utilizada como característica de entrada do modelo de aprendizagem.

Outro filtro que foi aplicado foi remover os lotes cuja taxa de conversão alimentar ultrapassasse 2.4, pois verificou-se que estes lotes "ruins"prejudicavam a aprendizagem, gerando anomalias nas previsões, como curvas atipicas em pontos específicos.

Aplicados os filtros acima, restaram 34 lotes para treino e 12 para teste.

Figura 16 – Invervalos de variação dos dados que foram utilizados na aprendizagem



Fonte: Autoria própria

Para auxiliar no entendimento da distribuição dos dados que serão utilizados na aprendizagem, a Figure 16 mostra o intervalo de variação do conjunto de dados, em cinza, e cada linha representa a média daquela variável. Podemos perceber que temos pouca variação de temperatura, por exemplo, indicando que nosso conjunto de dados pode não ser muito representativo, o que indica também que com o crescimento da base, novas amostras podem trazer contribuições significativas para estes resultados.

4.2.2 SEPARANDO OS DADOS EM CONJUNTOS DE TREINO E TESTE

Separar os dados em dois conjuntos, de treino e teste, possibilita que após o modelo de aprendizagem ter sido treinado, seja possível testar sua capacidade de prever instantes futuros da série temporal que nunca foi conhecida pelo modelo, indicativo de que ocorreu o aprendizado e quão boas estão as previsões em relação aos valores verdadeiros conhecidos.

De modo a manter a integridade da série temporal, decidiu-se separar os conjuntos de treino e teste por lotes de frango. Assim o conjunto de treino é composto por lotes completos diferentes dos lotes completos que compoem o conjunto de teste.

Para a divisão dos conjuntos foi utilizada a função *train_test_split* da biblioteca Scikit Learn. Esta função gerou uma lista de idendificadores de lotes de treino e outra de lotes de teste, sendo que o conjunto de treino era composto por 80% dos lotes, e o conjunto de teste era composto pelos 20% restantes.

4.2.3 NORMALIZAÇÃO

A normalização visa transformar todas as variáveis para que fiquem dentro de uma mesma escala de variação, para que as que variem mais expressivamente, como é o caso da Pressão, não tornem as variações de temperatura por exemplo que é mais sutil, insignificantes.

Foi empregado o método RobustScaler da biblioteca Scikit Learn, tanto para o conjunto de treino quanto o de teste, gerando um *array* Numpy tanto para as características (ou *features*), quanto para a variável de saída, Conversão Alimentar.

Uma cópia dos *dataframes* de treino e teste tiveram suas colunas de características e saída substituídas pelo *array* Numpy de saída da normalização, tendo como resultado final desta etapa *dataframes* contendo os valores de suas variáveis normalizadas. Estes por sua vez alientarão o sistema de aprendizagem.

4.2.4 DIVIDINDO AS SÉRIES TEMPORAIS EM SEGMENTOS

Segmentos neste contexto são janelas de tempo na série temporal, no caso deste trabalho, com largura de dois dias. A técnica, também chamada de *sliding window* consiste em gerar pequenas séries temporais que alimentam a função de aprendizagem, para que ela aprenda a prever o próximo elemento da série, transformando-o em um problema de aprendizado supervisionado com regressão.

Foi criada uma função que divide o dataframe em segmentos, e toma o cuidado de nunca criar um segmento que utilize parte dos dados de um lote e parte dos dados de outro, por exemplo, o último dia de um lote e o primeiro dia do lote seguinte, garantindo assim que os segmentos representem situações reais. Esta função recebe como parâmetros dois *dataframes*, um contendo as características de entrada, chamado de X (temperatura, umidade, pressão e idade) e outro *dataframe* contendo a saída y, também chamada de *label* (conversão alimentar), e oferece como retorno dois *arrays* Numpy: Um com as entradas e outro com as saídas, o *array* de entradas contendo uma lista de segmentos, cada segmento contém dois dias de dados (o tamanho da janela temporal) e cada dia contém as 4 características de entrada citadas anteriormente; o outro dataframe retornado possui a saída (conversão alimentar) de cada respectivo segmento do conjunto de entradas.

As dimensões dos *arrays* retornados são:

- X Treino: (1198, 2, 4)
- X Teste: (431, 2, 4)
- y Treino: (1198, 1)
- y Teste: (431, 1)

Mais detalhes da implementação podem ser vistos no Appendix B, na seção Criando Pedaços das Séries Temporais.

4.2.5 MODELO DE APRENDIZAGEM

Para implementar a aprendizagem utilizando células LSTM foi escolhida a biblioteca API Keras do Tensorflow.

O modelo contém três camadas, sendo elas *LSTM*, *Dropout* e *Dense*.

Na camada LSTM foram experimentados como parâmetros a quantidade de neurônios e a utilização de bias. Empiricamente, melhor relação entre tempo de treinamento e qualidade das predições foi obtida com 170 neurônios e a utilização de bias ativada.

Com relação às funções de ativação, resultados mais satisfatórios foram obtidos com as funções padrão da API Keras para a camada LSTM, sendo elas 'tanh' para ativação e

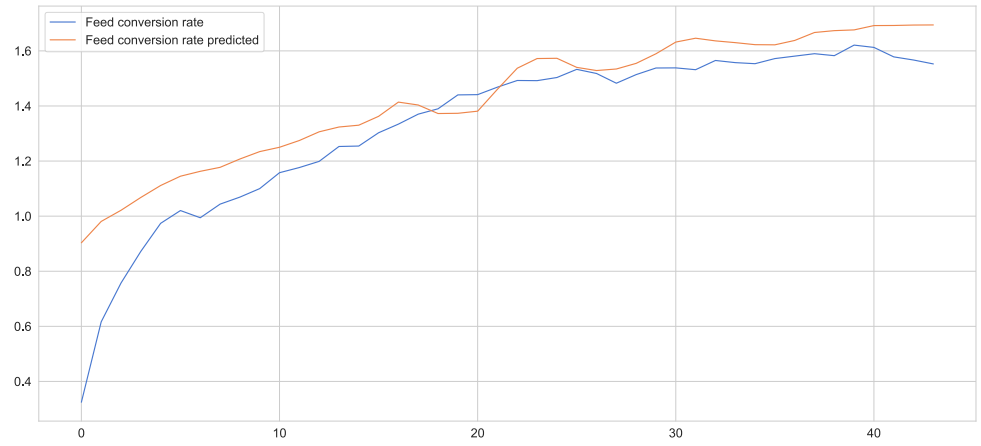
'sigmoid' para ativação recorrente. A aprendizagem apresentou melhores resultados sem função de ativação na camada densa.

Executando por 50 épocas, com todas as amostras sendo utilizadas em cada época, o erro quadrático médio com otimizador 'adam' foi de 0.0847

A implementação completa da aprendizagem está disponível no Appendix C na seção Modelo LSTM no Keras.

Neste mesmo apêndice, logo abaixo, na seção Modelo LSTM no Keras com funções de ativação, foram testadas outras funções de ativação, inclusive as mesmas combinações de parâmetros utilizadas no trabalho de (KLOTZ et al., 2020), porém para o conjunto de dados testado não apresentou melhores resultados.

Figura 17 – Conversão alimentar calculada VS prevista



Fonte: Autoria própria

A figura Figure 17 demostra a previsão da conversão alimentar para um lote qualquer do conjunto de teste.

Percebe-se um erro maior nas primeiras idades porém a conversão alimentar é mais relevante nas idades finais do lote, onde o erro é menor.

Ao final da etapa de aprendizagem, tanto o modelo treinado quanto os transformadores de normalização foram exportados para um arquivo e salvos.

Isto possibilita que na próxima etapa, de busca e otimização, o modelo possa ser carregado já treinado, e a normalização já ajustada para o conjunto de dados.

4.3 BUSCA E OTIMIZAÇÃO UTILIZANDO ALGORITMO GENÉTICO

Uma vez obtido o modelo capaz de prever a conversão alimentar para uma determinada idade a partir de um plano de ação, surge a necessidade de encontrar o plano de ação capaz de prover a menor conversão alimentar. Para isto foi implementado um algoritmo genético capaz de encontrar um bom plano de ação.

Esta etapa é iniciada importando o modelo de aprendizagem treinado e os transformadores de normalização, além do arquivo CSV que contém o dataframe com os dados de ambiência e conversão alimentar abordado na Subsubseção 4.1.5.6.

A implementação completa da etapa de busca e otimização utilizando algoritmo genético está disponível no Appendix D

4.3.1 População inicial

A população inicial é composta por todos os lotes do *dataframe*, portanto foi gerado um novo *dataframe* composto pelas colunas de características (temperatura, umidade, pressão e idade do lote), indexado pelo identificador do lote e pela idade. Sendo que cada plano de ação de um lote é considerado uma solução possível.

4.3.2 Avaliação ou *fitness*

Para avaliar cada solução, foi implementada a função que recebe como parâmetro um plano de ação e o submete a predição da conversão alimentar através do modelo de aprendizagem já treinado.

Da conversão alimentar prevista, é considerada apenas a conversão alimentar da idade de 40 dias. A pontuação deste plano de ação foi definida como o inverso da conversão alientar da idade 40. Com isto, quanto maior fosse a conversão alimentar, menor seria a pontuação deste plano de ação.

Nesta mesma função também foi implementado um sistema de penalização da solução, utilizando critérios inspirados nos manuais de manejo (COBB-VANTRESS, 2018), sendo penalizadas as soluções em que:

- Temperatura máxima seja maior que 34°C
- Temperatura mínima seja menor que 14°C
- Umidade máxima seja maior que 70%
- Umidade mínima seja menor que 30%

Caso a solução infrinja algum destes critérios, sua pontuação é penalizada em 10%

4.3.3 Recombinação ou *crossover*

A função desenvolvida para recombinação recebe como parâmetro um conjunto de planos de ação, e, para cada plano de ação escolhe aleatoriamente outro plano de ação do conjunto.

Para cada par de soluções, é escolhida uma característica, como por exemplo a temperatura, sendo gerada uma nova solução onde a característica escolhida da primeira solução é substituída pela da segunda solução.

A função retorna um conjunto de soluções de mesma quantidade do conjunto recebido como parâmetro, porém para cada solução retornada, uma das características sofreu recombinação.

4.3.4 Mutações

Foram implementadas quatro funções de mutação diferentes, porém apenas duas acabaram sendo utilizadas, por produzirem boas soluções:

Ambas recebem como parâmetros um conjunto de planos de ação e respondem com o conjunto com a mutação aplicada.

A primeira delas, chamada de *crossover mutation* é muito similar a função de recombinação, citada anteriormente, porém ao invés de trocar uma característica com a de outra solução, ela faz a média desta característica entre as duas soluções. Por exemplo: A solução 1

possui umidade de 70% na idade 15, já a solução 2 possui umidade 30% para a mesma idade. A nova solução gerada possuirá uma umidade de $(70 + 30)/2$, ou seja, 50%.

A segunda função de mutação, chamada de *random column mutation* é um pouco mais elaborada: Ela foi criada com o objetivo de causar uma mutação suave e gradual, para uma característica aleatória da solução original ao longo de todo o lote. Viu-se esta necessidade porque mudanças de ambiência não ocorrem abruptamente e não haveria sentido físico causar mutações em idades aleatórias do lote. A maneira encontrada de atingir este objetivo foi gerar uma curva de 50 pontos, baseada em um segmento aleatório da função seno. Ao realizar a mutação, uma característica aleatória da solução original é multiplicada por um ponto da curva gerada, causando o efeito desejado de mudança suave e gradual ao longo do lote.

4.3.5 Busca e classificação de soluções

O classificador de soluções seleciona as 15 soluções que obtiveram melhor pontuação na função de avaliação.

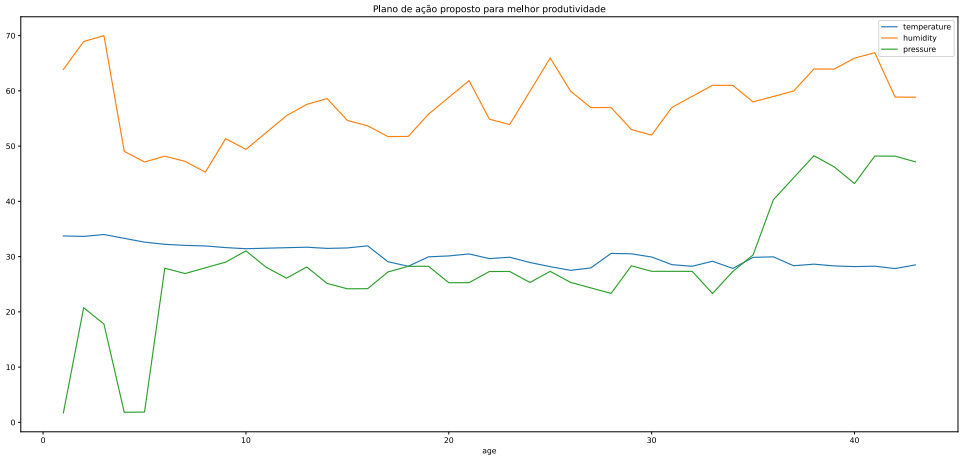
Estas 15 soluções passarão para a próxima geração sem sofrer alterações.

As funções de mutação *crossover mutation* e *random column mutation* são aplicadas às 15 melhores soluções citadas acima, gerando mais 15 soluções cada.

Desta forma, estas 45 soluções, passarão para a próxima geração, onde o procedimento será repetido.

As iterações de busca possuem dois critérios de parada, o primeiro é um limite na quantidade de gerações, que para os resultados aqui apresentados foi limitada a 200 gerações. O segundo ocorre caso a melhor pontuação não mude por mais de 10 gerações, este critério de parada nunca foi atingido antes de 200 gerações

Figura 18 – Plano de ação sugerido para menor taxa de conversão alimentar na idade 40

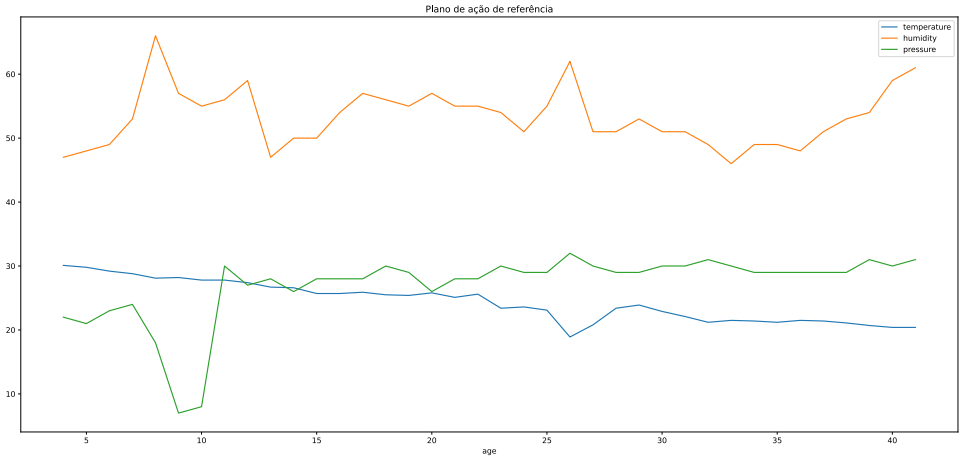


Fonte: Autoria própria

Após executar por 200 gerações, o melhor plano de ação encontrado está representado na Figure 18 obteve para a idade 40 dias uma taxa de conversão alimentar de 1,5007.

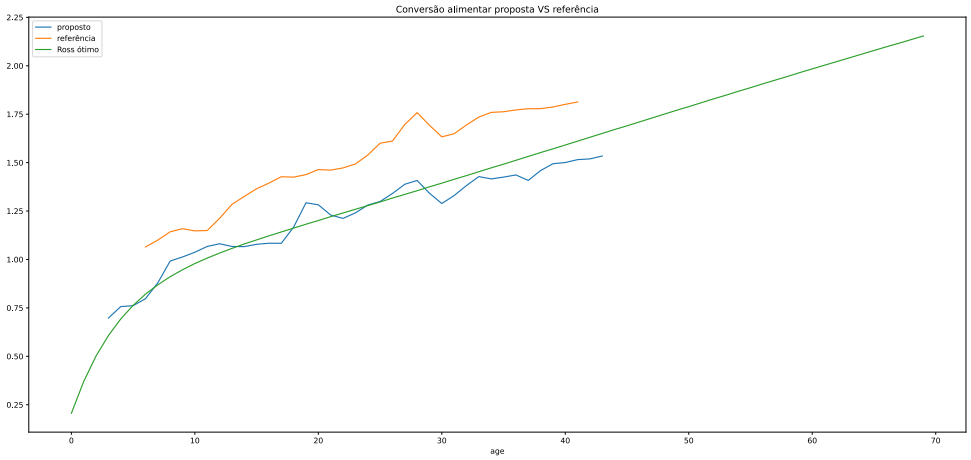
A taxa de conversão alimentar ótima para a idade 40 dias, segundo (AVIAGEN, 2019), é de 1,558. Onde o plano de ação aqui proposto sugere uma melhora de 3,68%.

Figura 19 – Plano de ação para referência



Fonte: Autoria própria

Figura 20 – Conversão alimentar do plano proposto VS referência VS Ótimo Ross



Fonte: Autoria própria

Comparando também com o melhor plano de ação obtido por (KLOTZ et al., 2020), cuja conversão alimentar prevista para a idade 40 é 1,5602, com a solução proposta por este trabalho indicando uma melhora de 3,81%

Na Figure 19 está exibido um plano de ação qualquer do conjunto de treinamento, para referência, e na Figure 20 está a comparação da conversão alimentar do plano proposto e do plano de referência.

5 CONCLUSÃO

Chegar ao objetivo de um plano de ação de ambiência cuja conversão alimentar fosse a menor possível, indicando melhor produtividade, se mostrou bastante desafiador principalmente ao utilizar como fonte de informação dados coletados de maneira automática. Muitas etapas de preparação dos dados foram necessárias para que fosse possível aplicar a Aprendizagem de Máquina e a busca e otimização utilizando Algoritmos Genéticos. Todavia, o esforço é necessário, se almejarmos que tal sistema seja aplicado em produção, onde dados muitas vezes ruins ou inconsistentes podem ser recebidos.

O plano de ação aqui sugerido, caso aplicado a um lote real, precisaria ser acompanhado de perto pela equipe de manejo para realizar ajustes de ambiência caso o comportamento das aves demonstre sinais de clima inadequado para as mesmas. Os elementos aleatórios da mutação do algoritmo genético fazem com que o plano sugerido seja diferente a cada execução. As penalidades aplicadas as soluções podem ser aprimoradas de acordo com o comportamento das aves constatado na execução do plano.

6 TRABALHOS FUTUROS

A previsão da conversão alimentar poderia ser incluída em uma plataforma de gestão, assim a produtividade de um lote em andamento poderia ser estimada com os dados reais do lote até o instante atual, e para os dias futuros ser utilizada o plano de ação que está configurado no controlador para o restante do lote, de modo que a previsão fosse ficando mais precisa com o avanço da idade do lote.

Apesar de terem sido consideradas temperatura, umidade e pressão como características de entrada que mais afetam a produtividade, outras características relevantes não foram consideradas por falta de informações. O sistema apresentado neste trabalho poderia ser extentido para incluir mais características, que venham a melhorar ainda mais suas previsões, dentre elas:

1. Sexo das aves alojadas;
2. Características das rações fornecidas às aves;
3. Temperatura e umidade externas do galpão;
4. Características de sazonalidade, como estação do ano;
5. Altitude em relação ao nível do mar;
6. Localização Geográfica;

Entre outras.

Prever a data e a hora em que as aves atingirão um peso alvo também seria possível, alterando a amostragem de diária para horária. Esta informação é importante tanto para a logística de desalojamento e coleta das aves, quanto para outra previsão importante: Quanto de ração ainda será necessária até que as aves atinjam o peso alvo.

A etapa de preparação de dados gera como saída um arquivo CSV contendo o conjunto de dados para aprendizagem; A etapa de aprendizagem importa o arquivo CSV, treina um modelo e exporta tanto o modelo quanto os transformadores de normalização em formato de arquivo; A etapa de busca e otimização importa o conjunto de dados do arquivo CSV, o modelo de aprendizagem treinado e os transformadores de normalização. Um próximo passo seria integrar as etapas a um fluxo que seja executado automaticamente baseado em algum critério, como por exemplo a cada 50 novos lotes adicionados a base de dados, fazendo com que a aprendizagem possa se tornar cada vez melhor, produzindo melhores previsões.

Integrar os resultados gerados pelo sistema proposto neste trabalho, com a entrada de configurações de controladores avícolas, ou mesmo com ferramentas de gestão, são, em concordância com o que sugere Ribeiro et al. (2019, p. 8), um esforço no sentido de tornar prática a utilização da aprendizagem de máquina como solução de melhora de produtividade das granjas.

Ribeiro et al. (2019, p. 8) também pontua o fato de que o especialista em manejo de aviários tende a manter a temperatura abaixo da indicada pelos manuais de manejo, com o objetivo de utilizar menos recursos. A depender do modelo de aquecedor, tais recursos podem ser energia elétrica ou lenha. Esta questão poderá ser explorada de forma a identificar até que ponto a diminuição da taxa de conversão gera lucros maiores do que a economia de luz ou lenha obtida ao deixar a temperatura abaixo da recomendada. Tais resultados poderiam indicar se a abordagem do especialista é economicamente viável.

Referências

ABPA. **Relatório Anual 2020**. 2020. 160 p. Disponível em: <https://abpa-br.org/mercados/>. Acesso em: 21 de Abril de 2020. Citado na página 1.

ABREU, V. M. N.; ABREU, P. G. d. Os desafios da ambiência sobre os sistemas de aves no brasil. **Revista Brasileira de Zootecnia**, v. 40, p. 1–14, 2011. Disponível em: <http://www.sbz.org.br>. Acesso em: 1 de julho de 2020. Citado na página 5.

AHMAD, A. A.-S.; ANDRAS, P. Scalability analysis comparisons of cloud-based software services. **Journal of Cloud Computing**, v. 10, n. 8, p. 17, 2019. Citado na página 6.

AMAZON. **Amazon Web Services (AWS) Machine Learning Solutions**. 2020. Disponível em: <https://aws.amazon.com/pt/machine-learning/>. Acesso em: 13 de Setembro de 2020. Citado na página 6.

AVIAGEN. **Ross Performance Guide**. [S.l.], 2019. 16 p. Disponível em: <http://eu.aviagen.com/tech-center/download/1339/Ross308-308FF-BroilerPO2019-EN.pdf>. Acesso em: 18 de Setembro de 2021. Citado na página 29.

BLANES-VIDAL, V.; FITAS, V.; TORRES, A. Differential pressure as a control parameter for ventilation in poultry houses: effect on air velocity in the zone occupied by animals. **Spanish Journal of Agricultural Research**, v. 1, n. 5, p. 31–37, 2007. Disponível em: <http://www.inia.es/sjar>. Acesso em: 1 de julho de 2020. Citado na página 5.

CETIC.BR. **Pesquisa sobre o Uso das Tecnologias de Informação e Comunicação nos Domicílios Brasileiros**. 2018. Disponível em: <https://cetic.br/media/docs/publicacoes/2/12225320191028-tic_dom_2018_livro_eletronico.pdf>. Acesso em: 7 de Fevereiro de 2020. Citado na página 6.

CHIANDUSSI, G. et al. Comparison of multi-objective optimization methodologies for engineering applications. **Computers & Mathematics with Applications**, v. 63, n. 5, p. 912 – 942, 2012. ISSN 0898-1221. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0898122111010406>. Citado na página 9.

COBB-VANTRESS. **Broiler Management Guide**. [S.l.], 2018. 112 p. Disponível em: <https://www.cobb-vantress.com/assets/5c7576a214/Broiler-guide-R1.pdf>. Acesso em: 20 de abril de 2020. Citado 7 vezes nas páginas 1, 2, 4, 5, 12, 16 e 28.

ESPíNDOLA, C. J. **Trajetórias do progresso técnico na cadeia produtiva de carne de frango do Brasil**. Florianópolis: [s.n.], 2012. 89–113 p. Disponível em: <https://doi.org/10.5007/2177-5230.2012v27n53p89>. Acesso em: 1 de Novembro de 2020. Citado na página 1.

FACELLI, K. et al. **Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina**. Rio de Janeiro: LTC, 2011. Citado na página 6.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 8.

GOOGLE. **Google Cloud Platform Machine Learning Solutions**. 2020. Disponível em: <https://cloud.google.com/ai-platform>. Acesso em: 13 de Setembro de 2020. Citado na página 6.

GéRON, A. **Mãos à Obra Aprendizado de Máquina com Scikit-Learn & TensorFlow: Conceitos, Ferramentas e Técnicas Para a Construção de Sistemas Inteligentes**. Rio de Janeiro: Alta Books, 2019. Citado 4 vezes nas páginas 6, 7, 8 e 11.

INOBRAM, A. e. S. e. A. **Manual de Instalação e Operação**: Smaai5. [S.l.], 2020. 40 p. Disponível em: <https://www.inobram.com.br/downloads>. Acesso em: 24 de Agosto de 2020. Citado 2 vezes nas páginas 5 e 6.

KLOTZ, D. et al. **Estimating action plans for smart poultry houses**. 2020. 18 p. Disponível em: <https://www.researchgate.net/publication/343710863>. Acesso em: 5 de novembro de 2020. Citado 8 vezes nas páginas 2, 6, 8, 9, 11, 12, 27 e 30.

LENZ, M. L. e. a. **Fundamentos de Aprendizagem de Máquina**. Porto Alegre: SAGAH, 2020. Citado 2 vezes nas páginas 6 e 7.

LIMA, J. F.; SIQUEIRA, S. H. G.; ARAúJO, D. V. **Avicultura**: Relato setorial. [S.l.], 1995. 43 p. Disponível em: <https://www.bndes.gov.br>. Acesso em: 13 de fevereiro de 2020. Citado na página 4.

LORENCENA, M. C. et al. A framework for modelling, control and supervision of poultry farming. **International Journal of Production Research**, Taylor & Francis, v. 58, n. 10, p. 3164–3179, 2020. Disponível em: <https://doi.org/10.1080/00207543.2019.1630768>. Citado na página 9.

MICROSOFT. **Azure Machine Learning Solutions**. 2020. Disponível em: <https://azure.microsoft.com/en-us/free/machine-learning/>. Acesso em: 13 de Setembro de 2020. Citado na página 6.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, Massachusetts: A Bradford Book The MIT Press, 1996. Citado na página 8.

MITCHELL, T. **Machine Learning**. McGraw-Hill, 1997. (McGraw-Hill International Editions). ISBN 9780071154673. Disponível em: <https://books.google.com.br/books?id=EoYBngEACAAJ>. Citado na página 6.

POP, D. **Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions**. Romênia: [s.n.], 2016. 12 p. Disponível em: <https://arxiv.org/abs/1603.08767>. Acesso em: 19 de Abril de 2020. Citado na página 6.

QUINTINO, L. F. **Indústria 4.0**. 2. ed. Porto Alegre: SAGAH, 2019. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595028531>. Acesso em: 26 de Junho de 2020. Citado na página 6.

RIBEIRO, R. et al. Generating action plans for poultry management using artificial neural networks. **Computers and Electronics in Agriculture**, v. 161, p. 131 – 140, 2019. ISSN 0168-1699. BigData and DSS in Agriculture. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0168169917304258>. Citado 5 vezes nas páginas 2, 9, 11, 12 e 32.

ROSS-AVIAGEN. **Ross Broiler Management Handbook**. [S.l.], 2018. 148 p. Disponível em: <http://www.aviagen.com/>. Acesso em: 05 de abril de 2020. Citado 6 vezes nas páginas 1, 2, 4, 5, 12 e 16.

SANTOS, B. P. et al. Industry 4.0: Challenges and opportunities. **Revista Produção e Desenvolvimento**, v. 4, n. 1, p. 111–124, Mar. 2018. Disponível em: <https://revistas.cefet-rj.br/index.php/producaoedesenvolvimento/article/view/e316>. Citado na página 6.

Siami-Namini, S.; Tavakoli, N.; Siami Namin, A. **A Comparison of ARIMA and LSTM in Forecasting Time Series**. 2018. 1394-1401 p. Citado na página 9.

SKINNER-NOBLE, D. O.; TEETER, R. G. **Components of Feed Efficiency in Broiler Breeding Stock: The Use of Fasted Body Temperature as an Indicator Trait for Feed Conversion in Broiler Chickens**. Oklahoma: [s.n.], 2004. 515–520 p. Citado na página 1.

TAVáREZ, M. A.; SANTOS, F. S. d. I. **Impact of genetics and breeding on broiler production performance: a look into the past, present, and future of the industry**. Dominican Republic: [s.n.], 2016. 37–41 p. Disponível em: <https://academic.oup.com/af/article/6/4/37/4638813>. Acesso em: 7 de Novembro de 2020. Citado na página 3.

VIANA, M. S.; JUNIOR, O. M.; CONTRERAS, R. C. Transgenic genetic algorithm to minimize the makespan in the job shop scheduling problem. In: **International Conference on Agents and Artificial Intelligence - ICAART**. [S.l.]: SciTePress, 2020. Citado na página 9.

Apêndices

APÊNDICE A – Engenharia de Dados

inobram-platform-data-engineering

November 26, 2021

1 Análise dos dados de leituras

1.1 Recriando a base

1.1.1 Observações

- Apenas as versões 12 e anteriores do PostgreSQL são compatíveis com MacOS High Sierra e anteriores
- Ao instalar o PostgreSQL a partir do site oficial, ainda é necessário exportar o caminho dos binários:

```
export PATH=/Library/PostgreSQL/12/bin:$PATH
```

Para tornar o export permanente, adicionar ao ~/.bash_profile:

```
echo "export PATH=/Library/PostgreSQL/12/bin:$PATH" >> ~/.bash_profile
```

Após a instalação via EnterpriseDB, o servidor PostgreSQL passa a iniciar automaticamente com o sistema. Para manipular o serviço, utilizar o comando: `./pg_ctl stop -D /Library/PostgreSQL/12/data`

Antes de prosseguir, caso não esteja trabalhando diretamente com o banco de produção, comente os últimos 4 comandos do dump SQL que revogam as permissões públicas e atribuiem todos os acessos ao cloudsqladmin (usuário do Google Cloud SQL, de onde a base foi exportada).

Para poder importar o dump, a base de dados precisa ser criada previamente. A criação de uma base vazia pode ser feita da seguinte forma:

```
psql -U postgres
postgres=# CREATE DATABASE "inobram-platform-local";
\q
psql -U postgres inobram-platform-local < ~/Downloads/inobram-platform-20200609.sql
```

1.2 Importar o dump do banco de dados utilizando o psql

O banco de dados original possui algumas permissões concedidas que gerarão um erro ao importar caso os usuários cloudsqladmin e cloudsqlsuperuser não existam.

Como o arquivo dump é muito grande (atualmente cerca de 11GB), fica impraticável abrir o arquivo em um editor de texto para remover estas permissões. Desde modo, é mais fácil adicionar estes usuários ao Postgres antes de proceder com a importação:

```
/Library/PostgreSQL/12/bin/psql -U postgres
```

```
CREATE USER cloudsqladmin WITH ENCRYPTED PASSWORD 'postgres';
CREATE USER cloudsqlsuperuser WITH ENCRYPTED PASSWORD 'postgres';
```

1.3 Criar uma *materialized view* com os dados de interesse

Uma Materialized View é uma consulta transformada em tabela, aqui ela está sendo utilizada para otimizar as análises de dados, uma vez que ela reúne em uma única tabela os dados de interese cruzados de duas tabelas distintas.

```
psql -U postgres inobram-platform-local -c 'CREATE MATERIALIZED VIEW reading_weight_analysis AS
```

O comando acima cria a Materialized View reading_weight_analysis, e popula com os dados, porém sempre que as tabelas de origem forem alteradas e queira-se que a Materialized View atualize seus valores, é necessário executar o seguinte comando:

```
psql -U postgres inobram-platform-local -c 'REFRESH MATERIALIZED VIEW reading_weight_analysis;
```

1.4 Carregando dados do banco PostgreSQL através do Pandas

A função a seguir irá efetuar uma consulta SQL no banco de dados, e os dados serão transformados em um DataFrame a ser manipulado pelo Pandas

[1]:
```
import pandas as pd
import sqlalchemy
import psycopg2

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "inobram-platform-local"
DB_USER = "postgres"
DB_PASSWORD = "postgres"
CON = f"postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/
 ↪{DB_NAME}"

SQL = "SELECT at AS avg_temperature, ah AS avg_humidity, p1 AS avg_pressure,␣
 ↪age, aw AS avg_weight, gain, sd AS std_deviation, wpvl AS␣
 ↪weight_percentage_very_light, wpl AS weight_percentage_light, wph AS␣
 ↪weight_percentage_heavy, wpvh AS weight_percentage_very_heavy FROM␣
 ↪reading_weight_analysis;"

def load_db_data(con=CON, sql=SQL):
    return pd.read_sql_query(sql, con)
```

1.5 Observando os dados carregados

O comando a seguir mostra as 5 primeiras linhas dos dados do DataFrame:

[2]:
```
readings = load_db_data()
```

[3]:
```
readings.head()
```

[3]:

	avg_temperature	avg_humidity	avg_pressure	age	avg_weight	gain	\
0	27.9	53	41	6	52.0	52.0	
1	32.0	48	38	6	52.0	52.0	
2	32.6	47	39	6	52.0	52.0	
3	32.1	50	39	6	52.0	52.0	
4	32.5	44	37	6	52.0	52.0	

	std_deviation	weight_percentage_very_light	weight_percentage_light	\
0	9.8	39.0	25.0	
1	9.8	39.0	25.0	
2	9.8	39.0	25.0	
3	9.8	39.0	25.0	
4	9.8	39.0	25.0	

	weight_percentage_heavy	weight_percentage_very_heavy
0	17.0	17.0
1	17.0	17.0
2	17.0	17.0
3	17.0	17.0
4	17.0	17.0

[4]:
```
readings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2085214 entries, 0 to 2085213
Data columns (total 11 columns):
 #   Column                       Dtype
---  ------                       -----
 0   avg_temperature              float64
 1   avg_humidity                 int64
 2   avg_pressure                 int64
 3   age                          int64
 4   avg_weight                   float64
 5   gain                         float64
 6   std_deviation                float64
 7   weight_percentage_very_light float64
 8   weight_percentage_light      float64
 9   weight_percentage_heavy      float64
 10  weight_percentage_very_heavy float64
dtypes: float64(8), int64(3)
memory usage: 175.0 MB
```

[5]:
```
readings["age"].value_counts()
```

[5]:
```
21    69979
20    69196
18    67310
```

```
26    66725
16    66182
19    65364
14    65322
15    65195
17    64137
13    64099
22    62609
23    62329
10    62112
11    61990
25    60244
24    60225
12    58533
9     57996
27    53200
8     52042
28    51112
30    49315
7     47900
32    44687
31    44677
29    43195
33    42590
6     42076
5     39821
38    37111
36    35554
35    34995
37    34955
34    34260
4     32819
39    29485
3     28374
41    24788
1     24417
2     22868
40    22507
42    19496
0     16489
43    13178
44     9365
45     4286
56      105
Name: age, dtype: int64
```

[6]:
```
readings.describe()
```

[6]:

	avg_temperature	avg_humidity	avg_pressure	age
count	2.085214e+06	2.085214e+06	2.085214e+06	2.085214e+06
mean	2.564394e+01	6.020913e+01	2.497377e+01	2.059856e+01
std	3.219686e+00	1.748853e+01	1.042872e+01	1.078780e+01
min	1.430000e+01	1.000000e+00	1.000000e+00	0.000000e+00
25%	2.350000e+01	4.700000e+01	1.900000e+01	1.200000e+01
50%	2.560000e+01	5.800000e+01	2.500000e+01	2.000000e+01
75%	2.780000e+01	7.300000e+01	3.200000e+01	2.900000e+01
max	4.100000e+01	9.800000e+01	8.500000e+01	5.600000e+01

	avg_weight	gain	std_deviation
count	2.085214e+06	2.085214e+06	2.085214e+06
mean	1.043937e+03	7.449286e+01	1.330489e+02
std	8.715959e+02	8.456479e+01	1.102167e+02
min	3.900000e+01	0.000000e+00	1.000000e+00
25%	3.120000e+02	4.200000e+01	4.710000e+01
50%	8.300000e+02	6.800000e+01	1.092000e+02
75%	1.544000e+03	8.900000e+01	1.887000e+02
max	5.298000e+03	2.032000e+03	8.916000e+02

	weight_percentage_very_light	weight_percentage_light
count	2.085214e+06	2.085214e+06
mean	2.708574e+01	4.123058e+01
std	1.215884e+01	1.217812e+01
min	2.000000e+00	0.000000e+00
25%	1.900000e+01	3.400000e+01
50%	2.500000e+01	4.300000e+01
75%	3.300000e+01	5.100000e+01
max	9.500000e+01	9.400000e+01

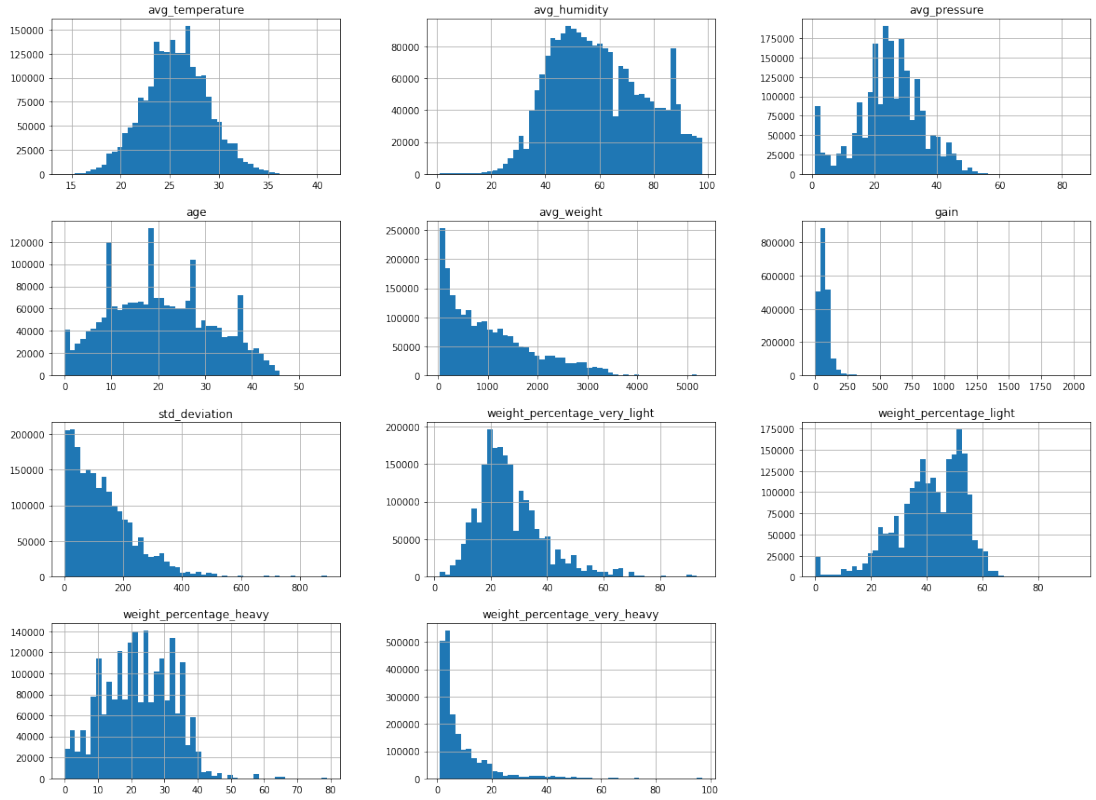
	weight_percentage_heavy	weight_percentage_very_heavy
count	2.085214e+06	2.085214e+06
mean	2.224933e+01	8.326525e+00
std	1.043142e+01	9.637100e+00
min	0.000000e+00	1.000000e+00
25%	1.500000e+01	3.000000e+00
50%	2.200000e+01	4.000000e+00
75%	3.100000e+01	1.100000e+01
max	7.900000e+01	9.700000e+01

1.6 Histograma

Permite analisar o numero de instâncias (no eixo vertical) que possuem um determinado intervalo de dados (no eixo horizontal)

[7]:
```
%matplotlib inline
# a linha acima e necessaria apenas no notebook jupiter
```

```
import matplotlib.pyplot as plt
readings.hist(bins=50, figsize=(20,15))
plt.savefig("histograma.svg")
#plt.show()
```



1.7 Criando um conjunto de testes estratificado

De modo a criar um conjunto de testes que seja representativo do conjunto total de dados, vamos nos certificar que a proporção de leituras por idade de lote seja mantida, para que idades de lote que possuem maior quantidade de leitura não tendenciem os resultados.

Está sendo usada uma semente randômica específica para garantir que os mesmos dados aleatórios sejam selecionados, mesmo que executemos a seleção inúmeras vezes.

[8]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(readings, readings["age"]):
    strat_train_set = readings.loc[train_index]
    strat_test_set = readings.loc[test_index]
```

Você pode usar o comando a seguir caso queira verificar que as proporções foram mantidas:

```
strat_test_set["age"].value_counts()/len(strat_test_set)
strat_train_set["age"].value_counts()/len(strat_train_set)
readings["age"].value_counts()/len(readings)
```

Para garantir que não iremos alterar o conjunto de treinamento, iremos criar uma cópia dele para manipularmos:

[9]:
```
readings = strat_train_set.copy()
```

1.8 Correlação dos dados

Os comandos a seguir analisam a correlação dos diferentes parâmetros com o as porcentagens de peso, onde:

[10]:
```
controlled_attributes = ["avg_pressure", "avg_humidity", "avg_temperature"]
```

1.8.1 Porcentagem de aves muito leves (quanto menor melhor)

[11]:
```
target = ["weight_percentage_very_light"]
corr_matrix = readings[controlled_attributes + target].corr()
corr_matrix[target.pop()].sort_values(ascending=False)
```

[11]:
```
weight_percentage_very_light    1.000000
avg_pressure                    0.141938
avg_humidity                    0.061792
avg_temperature                -0.153559
Name: weight_percentage_very_light, dtype: float64
```

Os dados acima indicam que o aumento da pressão parece aumentar o percentual de aves abaixo do peso, deve-se evitar pressões muito altas. Também é possível perceber que quanto mais a temperatura cresce, mais os pesos muito leves diminuem, temperaturas baixas devem ser evitadas.

Conclusão: Evitar pressões altas e temperaturas baixas

1.8.2 Porcentagem de aves leves (quanto menor melhor)

[12]:
```
target = ["weight_percentage_light"]
corr_matrix = readings[controlled_attributes + target].corr()
corr_matrix[target.pop()].sort_values(ascending=False)
```

[12]:
```
weight_percentage_light    1.000000
avg_pressure               0.169357
avg_humidity              -0.016987
avg_temperature           -0.335245
Name: weight_percentage_light, dtype: float64
```

Os dados acima indicam que o aumento da umidade parece aumentar o percentual de aves com peso leve, logo, devem ser evitadas. Novamente o aumento da temperatura colabora para a diminuição dos pesos leves.

Conclusão: Evitar umidades altas e temperaturas baixas

1.8.3 Porcentagem de aves pesadas (quanto maior melhor)

[13]:
```
target = ["weight_percentage_heavy"]
corr_matrix = readings[controlled_attributes + target].corr()
corr_matrix[target.pop()].sort_values(ascending=False)
```

[13]:
```
weight_percentage_heavy    1.000000
avg_temperature            0.251622
avg_humidity              -0.054838
avg_pressure              -0.188959
Name: weight_percentage_heavy, dtype: float64
```

Os dados acima indicam que quanto maior a temperatura, maior o percentual de aves consideradas pesadas. A pressão contribui negativamente, de modo que quanto menor a pressão, maior o percentual de aves pesadas.

Conclusão: Evitar temperaturas baixas e pressões altas

1.8.4 Porcentagem de aves muito pesadas (quanto maior melhor)

[14]:
```
target = ["weight_percentage_very_heavy"]
corr_matrix = readings[controlled_attributes + target].corr()
corr_matrix[target.pop()].sort_values(ascending=False)
```

[14]:
```
weight_percentage_very_heavy    1.000000
avg_temperature                 0.348383
avg_humidity                   -0.031204
avg_pressure                   -0.186872
Name: weight_percentage_very_heavy, dtype: float64
```

Aqui novamente é possível observar que temperaturas altas contribuem, enquanto umidades altas prejudicam.

Conclusão: Evitar temperaturas baixas e umidades altas

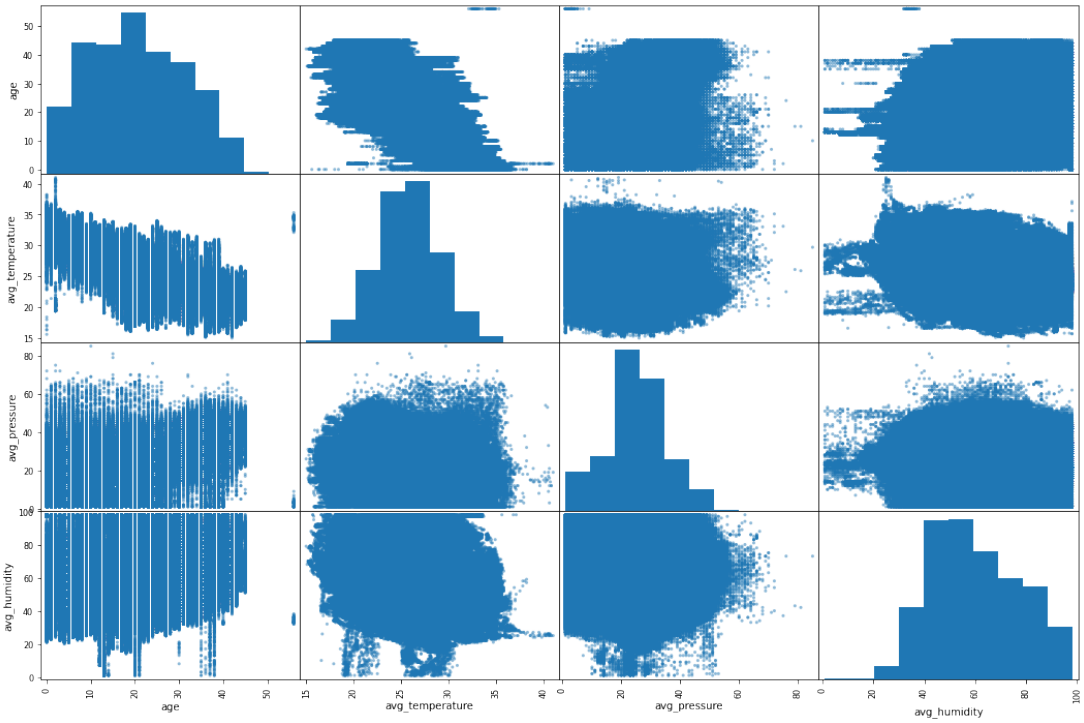
1.8.5 Conclusão das correlações

Para que tenhamos a maior parte das aves acima do peso, devemos buscar temperaturas mais altas, pressões e umidades mais baixas

1.9 Visualizando correlações através da Matriz de Dispersão

O médodo anterior, utilizando coeficientes de correlação é eficiente apenas em encontrar correlações lineares, quando este não for o caso. A Matriz de Dispersão auxilia a visualizar outros tipos de correlações:

[15]:
```
from pandas.plotting import scatter_matrix

attributes = ["age", "avg_temperature", "avg_pressure", "avg_humidity"]
matriz_correlacao = scatter_matrix(readings[attributes], figsize=(18, 12))
#plt.show()
plt.savefig("correlacao.png")
```

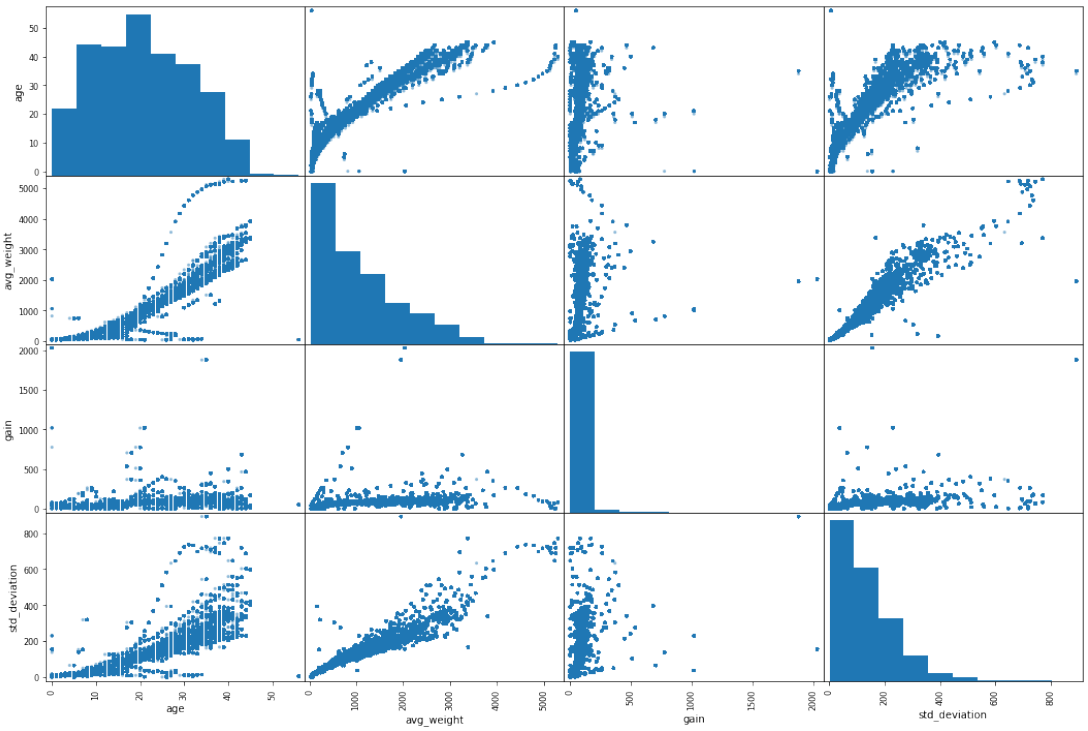


Exceto o fato de que a temperatura média pode ser menor quando a idade aumenta, nenhuma informação relevante pode ser observada na matriz acima.

[16]:
```
from pandas.plotting import scatter_matrix

attributes = ["age", "avg_weight", "gain", "std_deviation"]
scatter_matrix(readings[attributes], figsize=(18, 12))
```

[16]:
```
array([[<AxesSubplot:xlabel='age', ylabel='age'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='age'>,
        <AxesSubplot:xlabel='gain', ylabel='age'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='age'>],
       [<AxesSubplot:xlabel='age', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='gain', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='avg_weight'>],
       [<AxesSubplot:xlabel='age', ylabel='gain'>,
```

```
  <AxesSubplot:xlabel='avg_weight', ylabel='gain'>,
  <AxesSubplot:xlabel='gain', ylabel='gain'>,
  <AxesSubplot:xlabel='std_deviation', ylabel='gain'>],
 [<AxesSubplot:xlabel='age', ylabel='std_deviation'>,
  <AxesSubplot:xlabel='avg_weight', ylabel='std_deviation'>,
  <AxesSubplot:xlabel='gain', ylabel='std_deviation'>,
  <AxesSubplot:xlabel='std_deviation', ylabel='std_deviation'>]],
dtype=object)
```

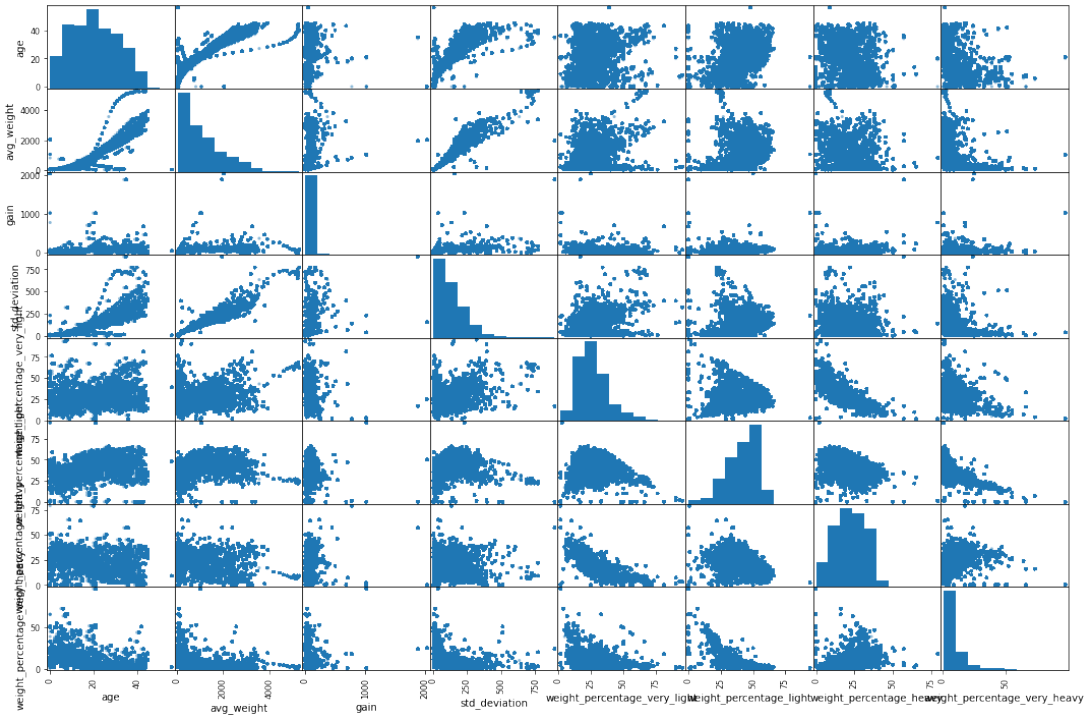


[17]:
```
from pandas.plotting import scatter_matrix

attributes = ["age", "avg_weight", "gain", "std_deviation",␣
 ↪"weight_percentage_very_light", "weight_percentage_light",␣
 ↪"weight_percentage_heavy", "weight_percentage_very_heavy"]
scatter_matrix(readings[attributes], figsize=(18, 12))
```

[17]:
```
array([[<AxesSubplot:xlabel='age', ylabel='age'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='age'>,
        <AxesSubplot:xlabel='gain', ylabel='age'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='age'>,
        <AxesSubplot:xlabel='weight_percentage_very_light', ylabel='age'>,
        <AxesSubplot:xlabel='weight_percentage_light', ylabel='age'>,
        <AxesSubplot:xlabel='weight_percentage_heavy', ylabel='age'>,
```

```
        <AxesSubplot:xlabel='weight_percentage_very_heavy', ylabel='age'>],
       [<AxesSubplot:xlabel='age', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='gain', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='avg_weight'>,
        <AxesSubplot:xlabel='weight_percentage_light', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='weight_percentage_heavy', ylabel='avg_weight'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='avg_weight'>],
       [<AxesSubplot:xlabel='age', ylabel='gain'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='gain'>,
        <AxesSubplot:xlabel='gain', ylabel='gain'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='gain'>,
        <AxesSubplot:xlabel='weight_percentage_very_light', ylabel='gain'>,
        <AxesSubplot:xlabel='weight_percentage_light', ylabel='gain'>,
        <AxesSubplot:xlabel='weight_percentage_heavy', ylabel='gain'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy', ylabel='gain'>],
       [<AxesSubplot:xlabel='age', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='gain', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='std_deviation'>,
        <AxesSubplot:xlabel='weight_percentage_light', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='weight_percentage_heavy', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='std_deviation'>],
       [<AxesSubplot:xlabel='age', ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='avg_weight',
ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='gain', ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='std_deviation',
ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='weight_percentage_light',
ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='weight_percentage_heavy',
ylabel='weight_percentage_very_light'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='weight_percentage_very_light'>],
       [<AxesSubplot:xlabel='age', ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='gain', ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='weight_percentage_light'>,
```

```
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='weight_percentage_light',
ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='weight_percentage_heavy',
ylabel='weight_percentage_light'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='weight_percentage_light'>],
       [<AxesSubplot:xlabel='age', ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='avg_weight', ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='gain', ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_light',
ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_heavy',
ylabel='weight_percentage_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='weight_percentage_heavy'>],
       [<AxesSubplot:xlabel='age', ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='avg_weight',
ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='gain', ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='std_deviation',
ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_very_light',
ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_light',
ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_heavy',
ylabel='weight_percentage_very_heavy'>,
        <AxesSubplot:xlabel='weight_percentage_very_heavy',
ylabel='weight_percentage_very_heavy'>]],
      dtype=object)
```



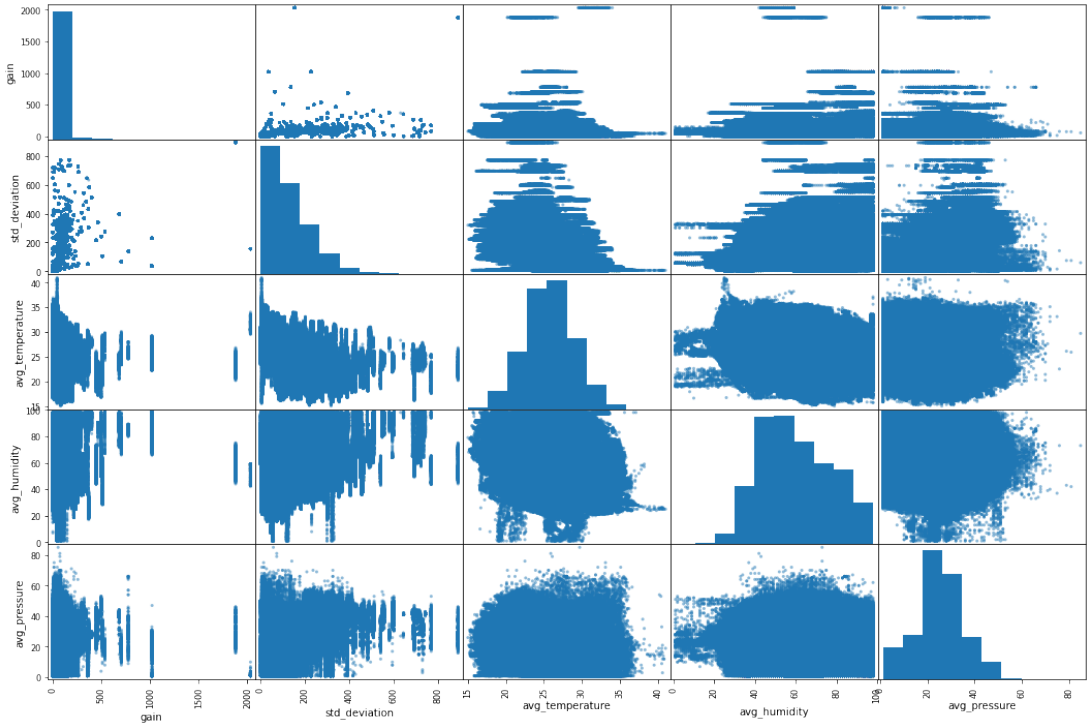
Desta última matriz, cruzando todos os parâmetros, nenhuma informação útil foi observada.

Vamos agora analisar alguns dados com mais detalhes, por exemplo: Como os parâmetros de Temperatura, Umidade e Pressão afetam o ganho e o desvio padrão:

[18]:
```
from pandas.plotting import scatter_matrix

attributes = ["gain", "std_deviation", "avg_temperature", "avg_humidity",␣
 ↪"avg_pressure"]
scatter_matrix(readings[attributes], figsize=(18, 12))
```

[18]:
```
array([[<AxesSubplot:xlabel='gain', ylabel='gain'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='gain'>,
        <AxesSubplot:xlabel='avg_temperature', ylabel='gain'>,
        <AxesSubplot:xlabel='avg_humidity', ylabel='gain'>,
        <AxesSubplot:xlabel='avg_pressure', ylabel='gain'>],
       [<AxesSubplot:xlabel='gain', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='avg_temperature', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='avg_humidity', ylabel='std_deviation'>,
        <AxesSubplot:xlabel='avg_pressure', ylabel='std_deviation'>],
       [<AxesSubplot:xlabel='gain', ylabel='avg_temperature'>,
        <AxesSubplot:xlabel='std_deviation', ylabel='avg_temperature'>,
        <AxesSubplot:xlabel='avg_temperature', ylabel='avg_temperature'>,
```

```
  <AxesSubplot:xlabel='avg_humidity', ylabel='avg_temperature'>,
  <AxesSubplot:xlabel='avg_pressure', ylabel='avg_temperature'>],
 [<AxesSubplot:xlabel='gain', ylabel='avg_humidity'>,
  <AxesSubplot:xlabel='std_deviation', ylabel='avg_humidity'>,
  <AxesSubplot:xlabel='avg_temperature', ylabel='avg_humidity'>,
  <AxesSubplot:xlabel='avg_humidity', ylabel='avg_humidity'>,
  <AxesSubplot:xlabel='avg_pressure', ylabel='avg_humidity'>],
 [<AxesSubplot:xlabel='gain', ylabel='avg_pressure'>,
  <AxesSubplot:xlabel='std_deviation', ylabel='avg_pressure'>,
  <AxesSubplot:xlabel='avg_temperature', ylabel='avg_pressure'>,
  <AxesSubplot:xlabel='avg_humidity', ylabel='avg_pressure'>,
  <AxesSubplot:xlabel='avg_pressure', ylabel='avg_pressure'>]],
dtype=object)
```



1.10 Preparando os dados para algoritmos de aprendizagem de máquina

Inicialmente, vamos separar os parâmetros sob os quais o controlador do aviário consegue atuar (previsores) dos resultados que estes valores medidos geraram (rótulos), sendo os previsores age, avg_temperature, avg_humidity, avg_pressure e os rótulos avg_weight, gain, std_deviation, weight_percentage_very_light, weight_percentage_light, weight_percentage_heavy, weight_percentage_very_heavy.

[19]:
```
# o parâmetro axis=1 indica que os rótulos são as colunas citadas, e não índices
readings = strat_train_set.drop(columns=["avg_weight", "gain", "std_deviation",␣
 ↪"weight_percentage_very_light", "weight_percentage_light",␣
 ↪"weight_percentage_heavy", "weight_percentage_very_heavy"])
readings_labels = strat_train_set.drop(columns=["age", "avg_temperature",␣
 ↪"avg_humidity", "avg_pressure"])
```

[20]:
```
readings.head()
```

[20]:

	avg_temperature	avg_humidity	avg_pressure	age
1551937	23.7	62	41	37
1708646	19.6	80	28	32
456398	27.6	94	21	12
580907	22.3	60	47	38
1139604	29.6	36	25	11

[21]:
```
readings_labels.head()
```

[21]:

	avg_weight	gain	std_deviation	weight_percentage_very_light	\
1551937	5144.0	22.0	717.1	65.0	
1708646	1855.0	113.0	172.7	8.0	
456398	380.0	51.0	62.0	23.0	
580907	2549.0	82.0	312.5	40.0	
1139604	288.0	36.0	44.6	25.0	

	weight_percentage_light	weight_percentage_heavy	\
1551937	23.0	8.0	
1708646	31.0	45.0	
456398	34.0	31.0	
580907	49.0	7.0	
1139604	36.0	30.0	

	weight_percentage_very_heavy
1551937	2.0
1708646	14.0
456398	11.0
580907	2.0
1139604	9.0

1.11 Exemplo de inner join

```
SELECT b.device, b.quantity_housed, r.rec_at, m.quantity AS mortality
FROM batch_batch AS b
INNER JOIN reading_reading AS r
    ON r.mac = b.device
INNER JOIN mortality_mortality AS m
    ON m.batch_id = b.id
```

```
WHERE b.quantity_housed > 0
    AND b.state = 'closed'
  AND b.id IN (
    SELECT DISTINCT batch_id FROM mortality_mortality
  )
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') > b.initial_date
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') < b.final_date
LIMIT 10;
```

1.12 Cálculo do consumo de ração

Na base de dados utilizada, não temos o consumo de ração pronto para uso. Para obtê-lo, precisaremos calcular através da redução de peso dos silos de ração com o passar do tempo.

O peso do silo é registrado minuto a minuto, através de balanças instadas nos pés dos silos. Para uma análise dos dados, será considerado um lote em que: * Esteja encerrado; * A quantidade de aves alojadas tenha sido fornecida; * A mortalidade tenha sido fornecida; * O consumo de água tenha sido registrado; * O peso dos silos de ração tenha sido registrado;

A critério de validação da preparação dos dados, serão utilizados dois lotes de galpões diferentes, para os quais se possui informações validadas de Conversão Alimentar, abastecimentos de ração, entre outros detalhes.

1.12.1 Obtendo os lotes que atendem as características

Uma lista de todos os lotes que atendem a estas características (base do dia 16/6/21), foi obtida com o seguinte código SQL:

```
SELECT DISTINCT b.id
FROM batch_batch AS b
INNER JOIN reading_reading AS r
    ON r.mac = b.device
INNER JOIN mortality_mortality AS m
    ON m.batch_id = b.id
WHERE b.quantity_housed > 0
    AND b.state = 'closed'
  AND b.id IN (
    SELECT DISTINCT batch_id FROM mortality_mortality
  )
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') > b.initial_date
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') < b.final_date
  AND r.h2o_acc > 0
  AND (r.sps1 > 0 OR r.sps2 > 0)
ORDER BY b.id DESC
```

Desta lista resultante, foi escolhido o lote mais recente, id 352, para calcular analisar o consumo de ração, foi criada uma materialized view apenas com os dados do lote a ser observado:

```
CREATE MATERIALIZED VIEW silo_weight_batch_373 AS
SELECT b.id AS batch_id, r.id AS reading_id, b.device, r.sps1, r.sps2, r.rec_at AS t_rec_at, T
FROM reading_reading AS r
```

```
INNER JOIN batch_batch AS b
    ON b.device = r.mac
INNER JOIN mortality_mortality AS m
    ON m.batch_id = b.id
  AND m.age = r.age
WHERE b.quantity_housed > 0
    AND b.state = 'closed'
  AND b.device = r.mac
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') > b.initial_date
  AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') < b.final_date
  AND r.h2o_acc > 0
  AND (r.sps1 > 0 OR r.sps2 > 0)
  AND b.id = 373
ORDER BY reading_id
```

1.12.2 Visualizando o peso do silo ao longo do lote:

[25]:
```
import pandas as pd
import sqlalchemy
import psycopg2

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "inobram-platform-local-21-07-30"
DB_USER = "postgres"
DB_PASSWORD = "postgres"
CON = f"postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/
 ↪{DB_NAME}"

SQL = "SELECT sps1 AS silo_weight, rec_at AS timestamp FROM␣
 ↪silo_weight_batch_373;"

def load_db_data(con=CON, sql=SQL):
    return pd.read_sql_query(sql, con)
```

[26]:
```
silo_weight = load_db_data()
silo_weight.head()
```

[26]:

	silo_weight	timestamp
0	4140	2021-05-30 03:00:01+00:00
1	4125	2021-05-30 06:45:03+00:00
2	4125	2021-05-30 06:46:04+00:00
3	4125	2021-05-30 06:47:04+00:00
4	4125	2021-05-30 06:48:05+00:00

[27]:
```
silo_weight.dtypes
```

[27]:
```
silo_weight                  int64
timestamp      datetime64[ns, UTC]
dtype: object
```

[28]:
```
silo_weight = silo_weight.set_index('timestamp')
```

[29]:
```
silo_weight.head(3)
```

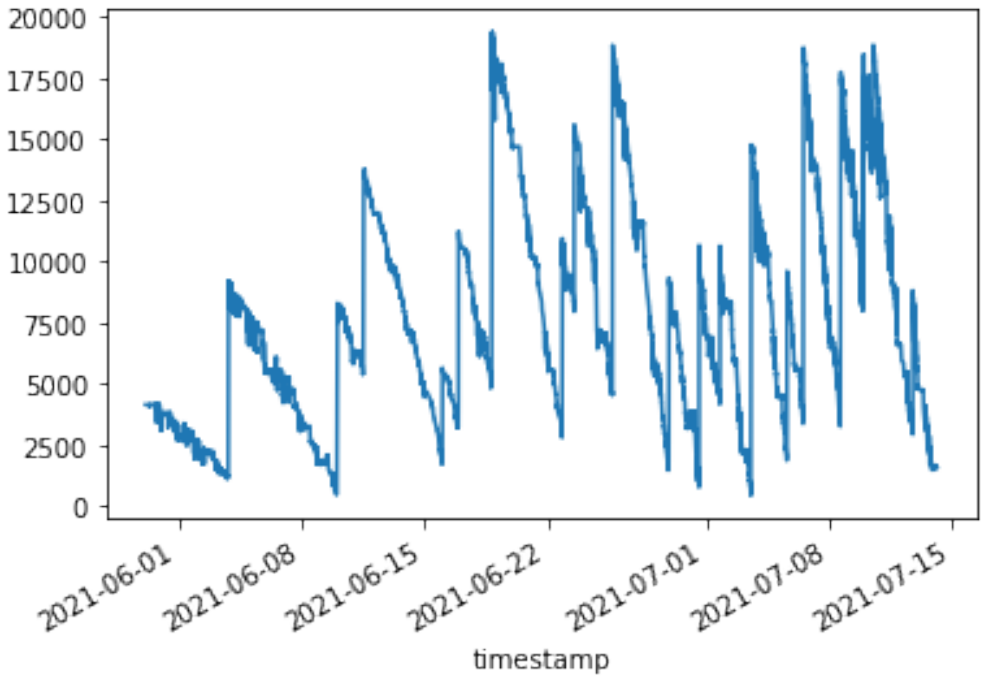
[29]:

timestamp	silo_weight
2021-05-30 03:00:01+00:00	4140
2021-05-30 06:45:03+00:00	4125
2021-05-30 06:46:04+00:00	4125

[30]:
```
silo_weight.index
```

[30]:
```
DatetimeIndex(['2021-05-30 03:00:01+00:00', '2021-05-30 06:45:03+00:00',
               '2021-05-30 06:46:04+00:00', '2021-05-30 06:47:04+00:00',
               '2021-05-30 06:48:05+00:00', '2021-05-30 06:49:06+00:00',
               '2021-05-30 06:50:06+00:00', '2021-05-30 06:51:06+00:00',
               '2021-05-30 06:52:07+00:00', '2021-05-30 06:53:07+00:00',
               …
               '2021-07-14 02:50:06+00:00', '2021-07-14 02:51:07+00:00',
               '2021-07-14 02:52:07+00:00', '2021-07-14 02:53:08+00:00',
               '2021-07-14 02:54:08+00:00', '2021-07-14 02:55:09+00:00',
               '2021-07-14 02:56:10+00:00', '2021-07-14 02:57:10+00:00',
               '2021-07-14 02:58:11+00:00', '2021-07-14 02:59:12+00:00'],
              dtype='datetime64[ns, UTC]', name='timestamp', length=57070,
freq=None)
```

[33]:
```
import matplotlib.pyplot as plt

silo_weight_plot = silo_weight['silo_weight'].plot()
silo_weight_plot.figure.savefig('silo-weight.svg')
```



1.12.3 Redução de ruídos

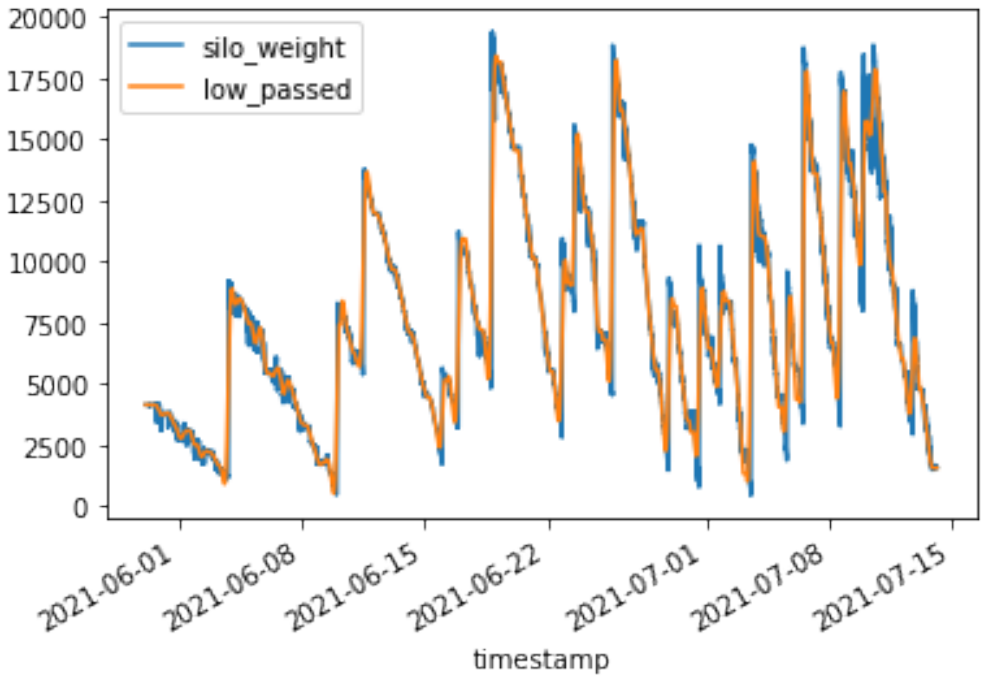
No gráfico acima é possível perceber bastante ruído no peso do silo, pois as retas verticais de aumento de peso, com mais de 2500Kg de variação, significam que o silo está sendo abastecido. Qualquer pequeno aumento de peso é ruído e deveria ser desconsiderado.

Estes ruídos em geral possuem uma frequência bem maior que os intervalos de abastecimento e consumo. Com isto em mente será aplicado um filtro passa baixa.

Lote 373

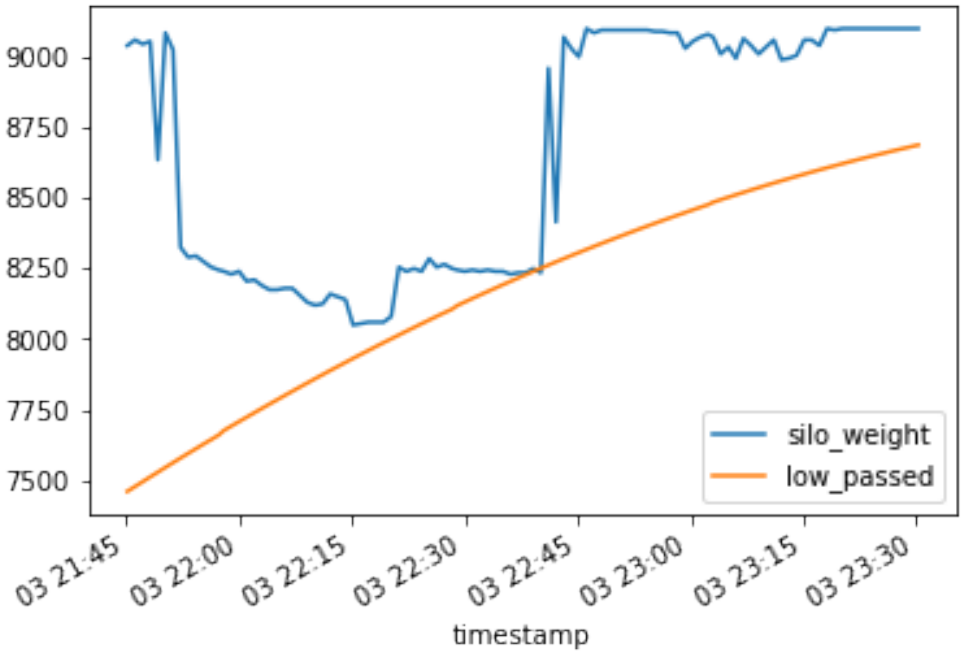
[44]:
```
from scipy import signal

#b, a = signal.butter(3, 0.15, btype='lowpass', analog=False)
b, a = signal.butter(3, 0.003, btype='lowpass', analog=False)
low_passed = signal.filtfilt(b, a, silo_weight['silo_weight'])

if 'low_passed' in silo_weight.keys():
    discard = silo_weight.pop('low_passed')
silo_weight.insert(1, 'low_passed', low_passed)
```

[52]:
```
silo_weight['silo_weight'].plot(legend=True)
silo_weight['low_passed'].plot(legend=True).figure.savefig('silo-weight-butter.
↪svg')
```



[53]:
```
silo_weight.loc['2021-06-03 21:45':'2021-06-03 23:30', 'silo_weight'].
 ↪plot(legend=True)
silo_weight.loc['2021-06-03 21:45':'2021-06-03 23:30', 'low_passed'].
 ↪plot(legend=True).figure.savefig('silo-weight-butter-anomaly.svg')
```

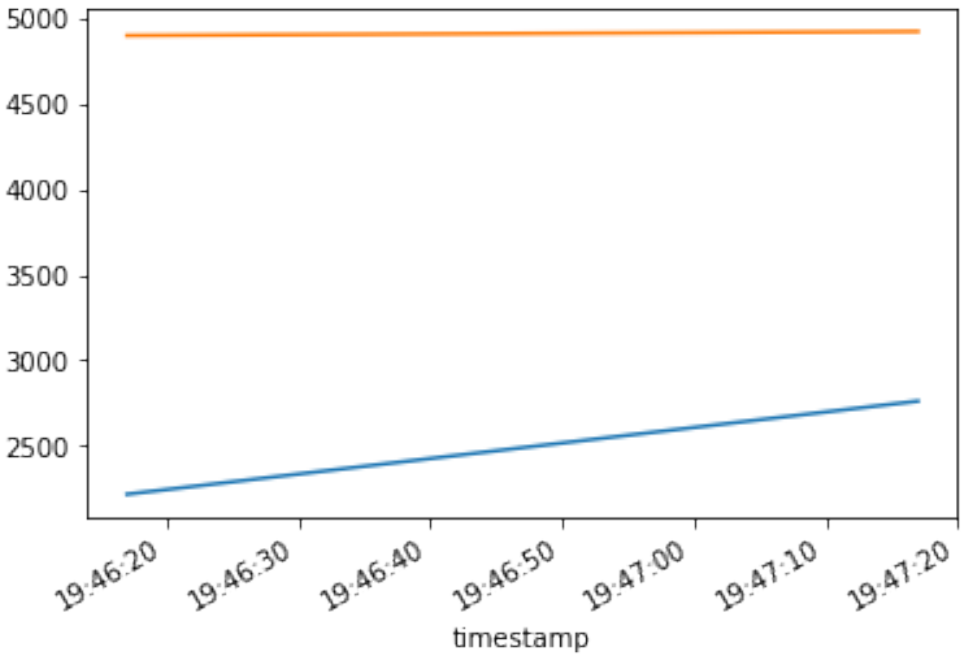
```
/Users/gean/projects/tcc/.venv/lib/python3.8/site-
packages/pandas/core/indexing.py:862: FutureWarning: Value based partial slicing
on non-monotonic DatetimeIndexes with non-existing keys is deprecated and will
raise a KeyError in a future Version.
  return getattr(section, self.name)[new_key]
/Users/gean/projects/tcc/.venv/lib/python3.8/site-
packages/pandas/core/indexing.py:862: FutureWarning: Value based partial slicing
on non-monotonic DatetimeIndexes with non-existing keys is deprecated and will
raise a KeyError in a future Version.
  return getattr(section, self.name)[new_key]
```



[48]:
```
silo_weight.loc['2021-06-03 19:46':'2021-06-03 19:47', 'silo_weight'].plot()
silo_weight.loc['2021-06-03 19:46':'2021-06-03 19:47', 'low_passed'].plot()
```

```
/Users/gean/projects/tcc/.venv/lib/python3.8/site-
packages/pandas/core/indexing.py:862: FutureWarning: Value based partial slicing
on non-monotonic DatetimeIndexes with non-existing keys is deprecated and will
raise a KeyError in a future Version.
  return getattr(section, self.name)[new_key]
/Users/gean/projects/tcc/.venv/lib/python3.8/site-
packages/pandas/core/indexing.py:862: FutureWarning: Value based partial slicing
on non-monotonic DatetimeIndexes with non-existing keys is deprecated and will
raise a KeyError in a future Version.
  return getattr(section, self.name)[new_key]
```

[48]: `<AxesSubplot:xlabel='timestamp'>`



[42]:
```
import pandas_bokeh

silo_weight.plot_bokeh(kind='line')
```

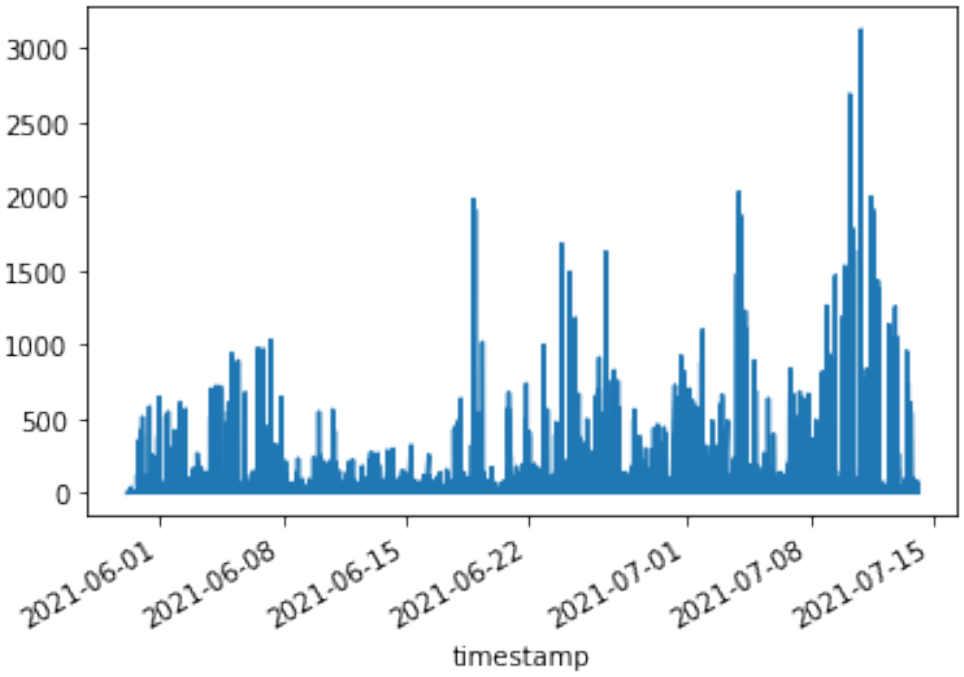
[42]:
```
Figure(id='1002', …)
```

[49]:
```
consumption = [0]
total_silo_weights = silo_weight['silo_weight'].size
valid_silo_weights = 0
invalid_silo_weights = 0
for index in range(1, silo_weight['silo_weight'].size):
    if silo_weight['silo_weight'][index - 1] >=␣
 ↪(silo_weight['silo_weight'][index]):
        instant_consumption = silo_weight['silo_weight'][index - 1] -␣
 ↪silo_weight['silo_weight'][index]
        #instant_consumption = None if instant_consumption > 40 else␣
 ↪instant_consumption
        consumption.append(instant_consumption)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        consumption.append(0)
        invalid_silo_weights += 1
if 'consumption' in silo_weight.keys():
```

```
    discard = silo_weight.pop('consumption')
silo_weight.insert(1, 'consumption', consumption)
```

[50]:
```
silo_weight['consumption'].plot()
#silo_weight['silo_weight'].plot()
```

[50]: <AxesSubplot:xlabel='timestamp'>



[51]:
```
print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
print('invalid silo weights: ', invalid_silo_weights)
print('total silo weights: ', total_silo_weights)
print('Batch feed consumption: ', silo_weight['consumption'].sum())
```

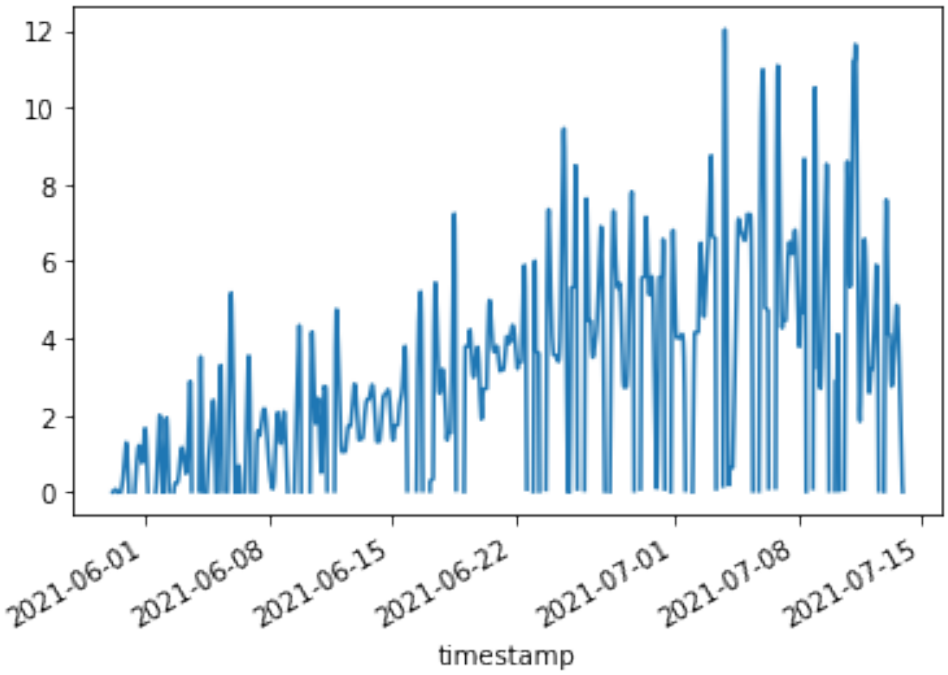
```
total silo weights:  57070
valid silo weights:  45803
invalid silo weights:  11266
total silo weights:  57070
Batch feed consumption:  720745
```

[54]:
```
consumption = [0]
total_silo_weights = silo_weight['low_passed'].size
valid_silo_weights = 0
invalid_silo_weights = 0
```

```
for index in range(1, silo_weight['low_passed'].size):
    if silo_weight['low_passed'][index - 1] >=␣
 ↪(silo_weight['low_passed'][index]):
        instant_consumption = silo_weight['low_passed'][index - 1] -␣
 ↪silo_weight['low_passed'][index]
        #instant_consumption = None if instant_consumption > 40 else␣
 ↪instant_consumption
        consumption.append(instant_consumption)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        consumption.append(None)
        invalid_silo_weights += 1
if 'consumption' in silo_weight.keys():
    discard = silo_weight.pop('consumption')
silo_weight.insert(1, 'consumption', consumption)
```

[55]:
```
silo_weight['consumption'].plot()
#silo_weight['low_passed'].plot()
```

[55]: <AxesSubplot:xlabel='timestamp'>



[56]:
```
print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
```

```
print('invalid silo weights: ', invalid_silo_weights)
print('total silo weights: ', total_silo_weights)
print('Batch feed consumption: ', silo_weight['consumption'].sum())
```

```
total silo weights:  57070
valid silo weights:  42379
invalid silo weights:  14690
total silo weights:  57070
Batch feed consumption:  152467.90070447262
```

1.12.4 Consumo baseado nos abastecimentos do silo

Calcular o consumo baseado nas diminuições de peso se mostrou muito ruidoso e delicado de se aplicar filtro, pois as variacões são relativamente pequenas.

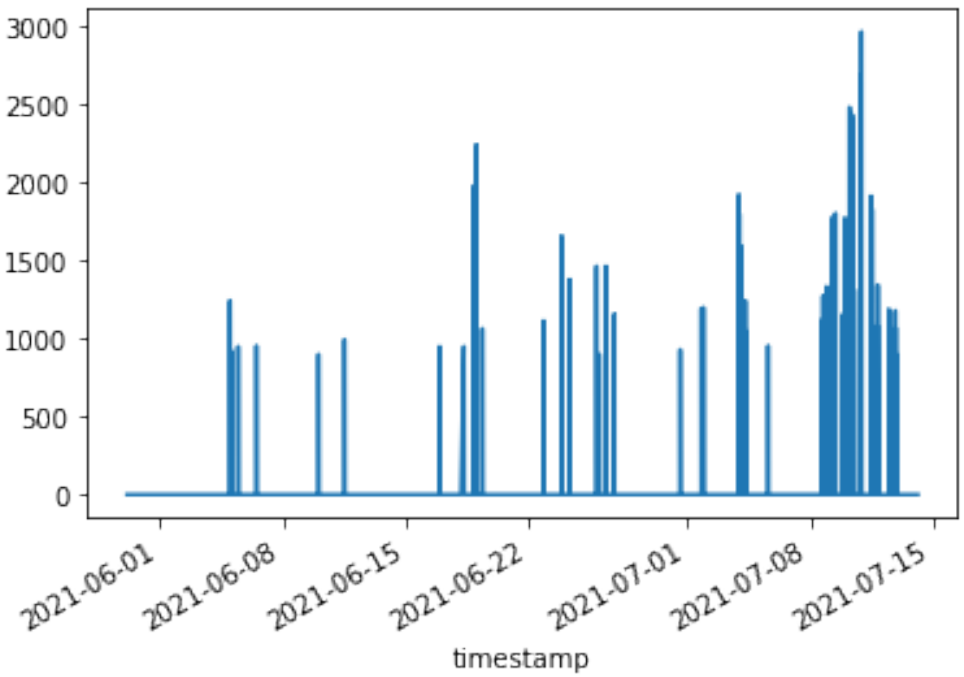
Baseado no método utilizado pelo produtor, pode ser calculado o consumo diário baseando-se nos abastecimentos do silo. Considera-se o peso inicial do silo e 1000 a 1500Kg de ração nas linhas de comedouros. A cada final de dia, calcula-se a diminuição de peso. Se naquele dia o silo foi abastecido, faz-se dia anterior + abastecimento - dia atual.

Sem filtro passa-baixa

[57]:
```
refills = [0]
total_silo_weights = silo_weight['silo_weight'].size
valid_silo_weights = 0
invalid_silo_weights = 0
for index in range(1, silo_weight['silo_weight'].size):
    if (silo_weight['silo_weight'][index - 1] + 900) <=␣
 ↪(silo_weight['silo_weight'][index]):
        instant_refill = silo_weight['silo_weight'][index] -␣
 ↪silo_weight['silo_weight'][index - 1]
        #instant_consumption = None if instant_consumption > 40 else␣
 ↪instant_consumption
        refills.append(instant_refill)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        refills.append(0)
        invalid_silo_weights += 1
if 'refills' in silo_weight.keys():
    discard = silo_weight.pop('refills')
silo_weight.insert(1, 'refills', refills)

innitial_load = silo_weight['silo_weight'][0]
final_load = silo_weight['silo_weight'][-1]
batch_consumption = innitial_load + silo_weight['refills'].sum() - final_load
```

[58]:
```
silo_weight['refills'].plot()
#silo_weight['low_passed'].plot()
```

[58]: <AxesSubplot:xlabel='timestamp'>



[59]:
```
print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
print('invalid silo weights: ', invalid_silo_weights)
print('refills: ', int(silo_weight['refills'].sum()))
print('innitial load: ', int(innitial_load))
print('final_load: ', int(final_load))
print('Batch feed consumption: ', int(batch_consumption))
```

```
total silo weights:  57070
valid silo weights:  81
invalid silo weights:  56988
refills:  111000
innitial load:  4140
final_load:  1565
Batch feed consumption:  113575
```

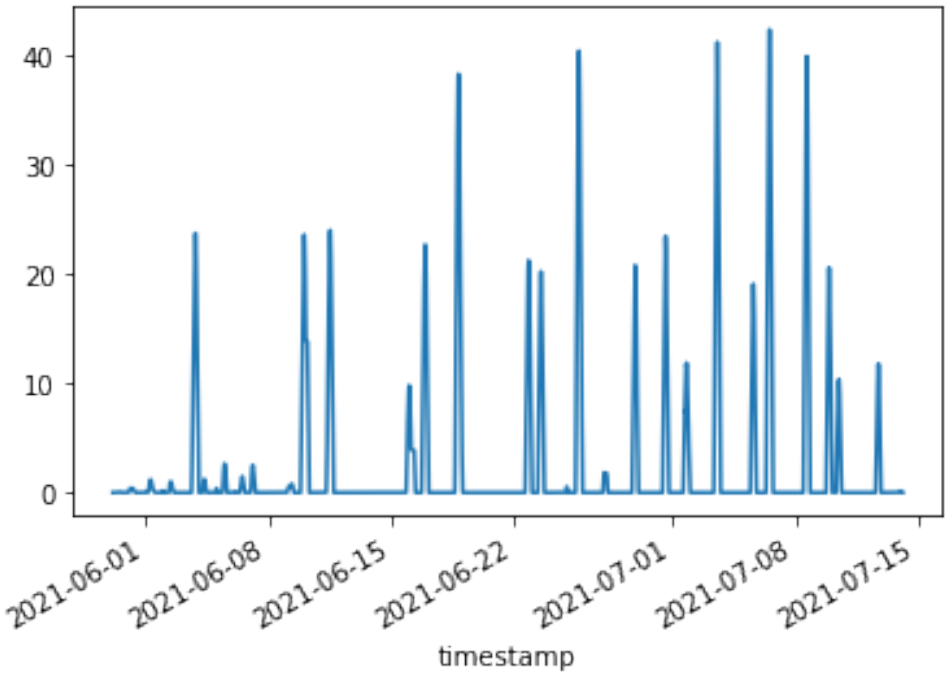
Com filtro passa-baixa

[60]:
```
refills = [0]
total_silo_weights = silo_weight['low_passed'].size
```

```
valid_silo_weights = 0
invalid_silo_weights = 0
for index in range(1, silo_weight['low_passed'].size):
    if (silo_weight['low_passed'][index - 1]) <=␣
 ↪(silo_weight['low_passed'][index]):
        instant_refill = silo_weight['low_passed'][index] -␣
 ↪silo_weight['low_passed'][index - 1]
        #instant_consumption = None if instant_consumption > 40 else␣
 ↪instant_consumption
        refills.append(instant_refill)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        refills.append(0)
        invalid_silo_weights += 1
if 'refills' in silo_weight.keys():
    discard = silo_weight.pop('refills')
silo_weight.insert(1, 'refills', refills)

innitial_load = silo_weight['low_passed'][0]
final_load = silo_weight['low_passed'][-1]
batch_consumption = innitial_load + silo_weight['refills'].sum() - final_load
```

[61]:
```
silo_weight['refills'].plot()
#silo_weight['low_passed'].plot()
```

[61]: <AxesSubplot:xlabel='timestamp'>



[62]:
```
print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
print('invalid silo weights: ', invalid_silo_weights)
print('refills: ', int(silo_weight['refills'].sum()))
print('innitial load: ', int(innitial_load))
print('final_load: ', int(final_load))
print('Batch feed consumption: ', int(batch_consumption))
```

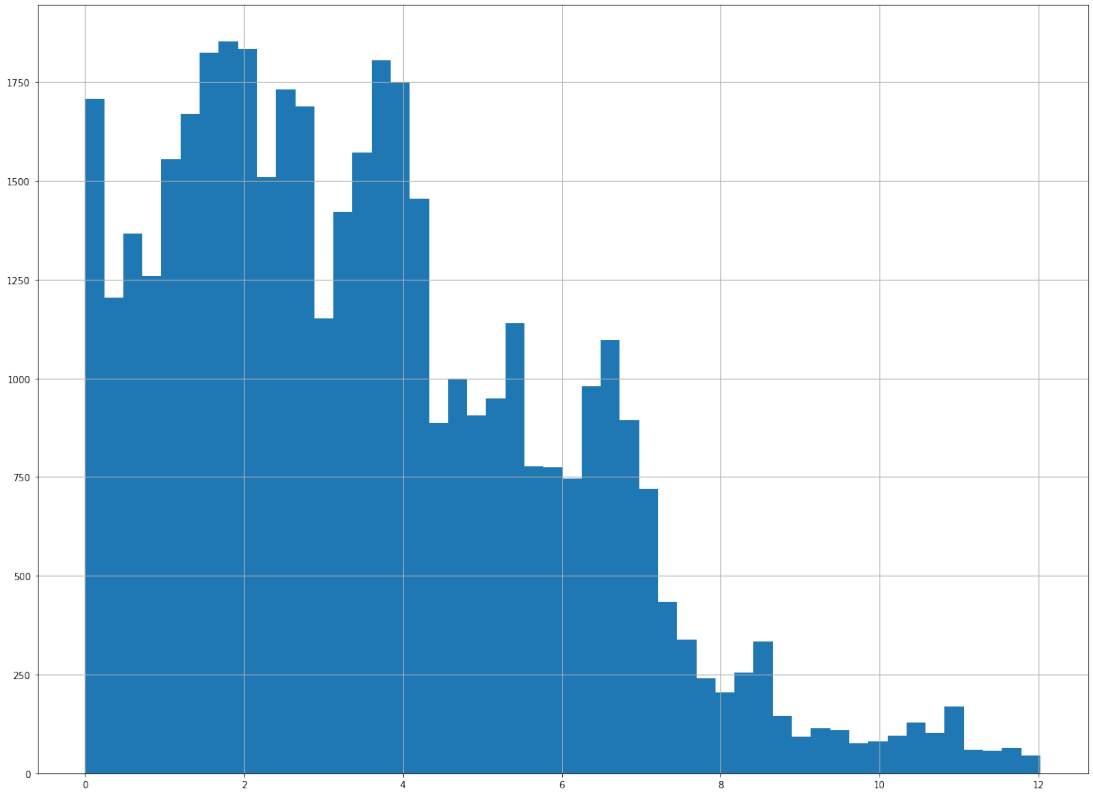
```
total silo weights:  57070
valid silo weights:  14690
invalid silo weights:  42379
refills:  149889
innitial load:  4137
final_load:  1559
Batch feed consumption:  152467
```

1.12.5 Histograma do consumo de ração

Permite analisar os valores de consumo de ração mais comuns no intervalo de 1 minuto, com isto fica mais fácil fazer um filtro para desconsiderar outliers. Os outliers no nosso caso são o vento, alguém subindo no silo ou os momentos em que o silo é abastecido.

[63]:
```
%matplotlib inline
# a linha acima e necessaria apenas no notebook jupiter
import matplotlib.pyplot as plt
```

```
silo_weight['consumption'].hist(bins=50, figsize=(20,15))
#silo_weight['silo_weight'].hist(bins=50, figsize=(20,15))
plt.show()
```



No histograma acima é possível constatar que a maioria das medidas de consumo minuto a minuto está entre 0Kg e 40Kg. Com base nisto, iremos desconsiderar variações de peso de silo maiores do que estes valores, pois quando ocorrerem possívelmente sejam ruídos.

1.13 Consumo Diário de Ração (Resampling)

Inicialmente faremos o cálculo da taxa de conversão alimentar baseado em dados diários, então iremos somar todo consumo de ração ocorrido no galpão durante o dia:

[64]:
```
columns = ['consumption']
silo_consumption_daily_sum = silo_weight[columns].resample('D').sum()
```

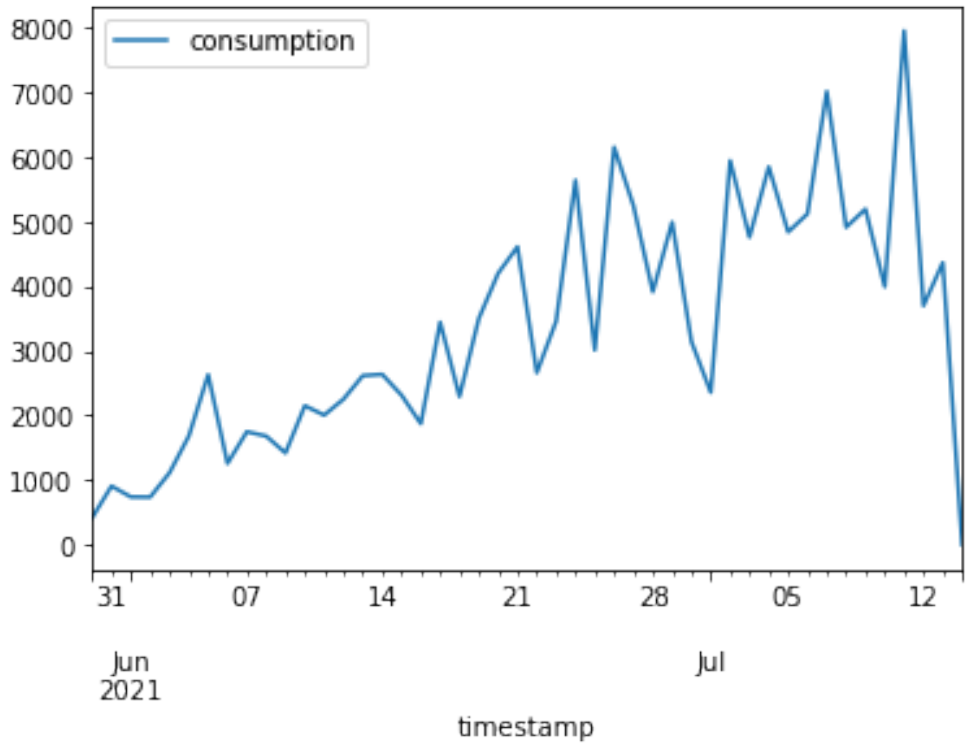
Verificando que temos a quantidade de dias de um lote:

[65]:
```
print('Quantidade de dias: ', silo_consumption_daily_sum.shape[0])
```

```
Quantidade de dias:  46
```

[66]:
```
silo_consumption_daily_sum.plot()
```

[66]: <AxesSubplot:xlabel='timestamp'>



[67]:
```
silo_consumption_daily_sum.plot_bokeh(kind='line')
```

[67]: Figure(id='1336', …)

Devido à grande diminuição de peso nos silos nos primeiros dias do lote, supomos que as linhas de comedouros que estavam vazias antes do alojamento são preenchidas com ração, fazendo com que o peso do silo diminua bastante, outra possibilidade é que nos primeiros dias as aves ingiram uma quantidade muito maior de ração. Outros lotes serão analisados a critério de comparação.

2 Batch 374

[68]:
```
import pandas as pd
import sqlalchemy
import psycopg2

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "inobram-platform-local-21-07-30"
```

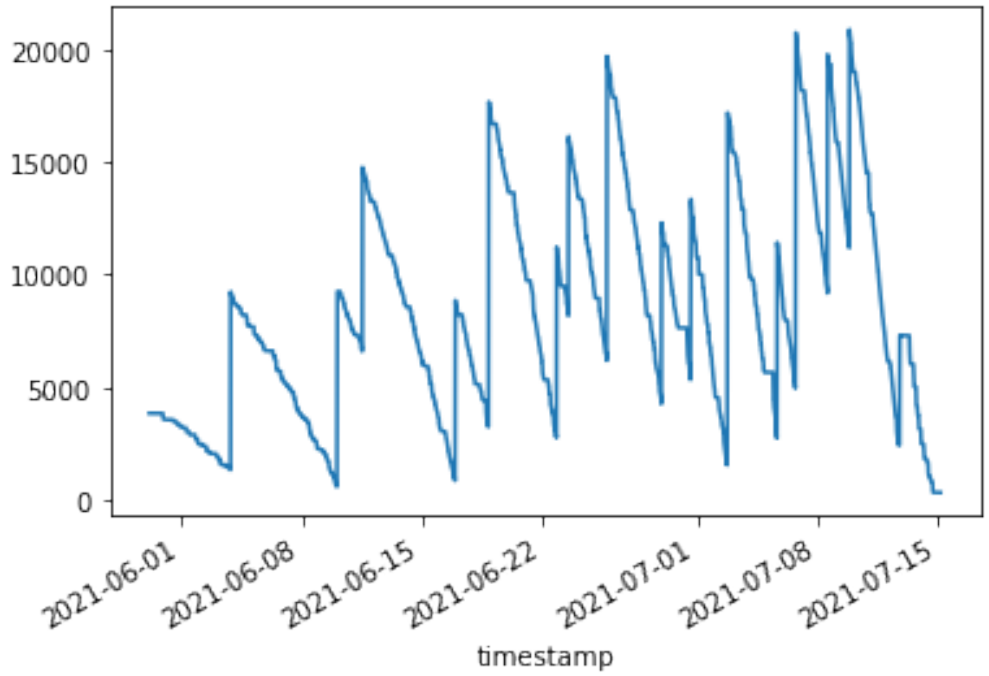
```
DB_USER = "postgres"
DB_PASSWORD = "postgres"
CON = f"postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/
 ↪{DB_NAME}"

SQL = "SELECT sps1 AS silo_weight, rec_at AS timestamp, age FROM␣
 ↪silo_weight_batch_374;"

def load_db_data(con=CON, sql=SQL):
    return pd.read_sql_query(sql, con)
```

[69]:
```
silo_weight_374 = load_db_data()
```

[70]:
```
silo_weight_374 = silo_weight_374.set_index('timestamp')
```

[71]:
```
silo_weight_374['silo_weight'].plot()
```

[71]: <AxesSubplot:xlabel='timestamp'>

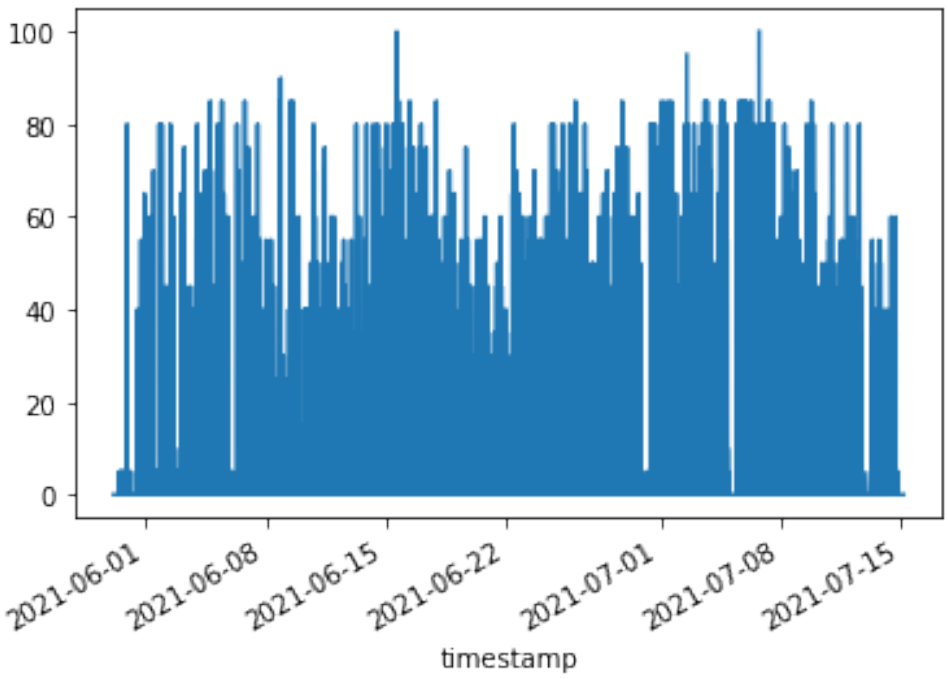


[72]:
```
consumption = [0]
total_silo_weights = silo_weight_374['silo_weight'].size
valid_silo_weights = 0
invalid_silo_weights = 0
for index in range(1, silo_weight_374['silo_weight'].size):
```

```
    if silo_weight_374['silo_weight'][index - 1] >=␣
↪(silo_weight_374['silo_weight'][index]):
        instant_consumption = silo_weight_374['silo_weight'][index - 1] -␣
↪silo_weight_374['silo_weight'][index]
        instant_consumption = 0 if instant_consumption > 100 else␣
↪instant_consumption
        consumption.append(instant_consumption)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        consumption.append(None)
        invalid_silo_weights += 1
if 'consumption' in silo_weight_374.keys():
    discard = silo_weight_374.pop('consumption')
silo_weight_374.insert(1, 'consumption', consumption)
```

[73]:
```
silo_weight_374['consumption'].plot()
#silo_weight['silo_weight'].plot()
```

[73]: <AxesSubplot:xlabel='timestamp'>



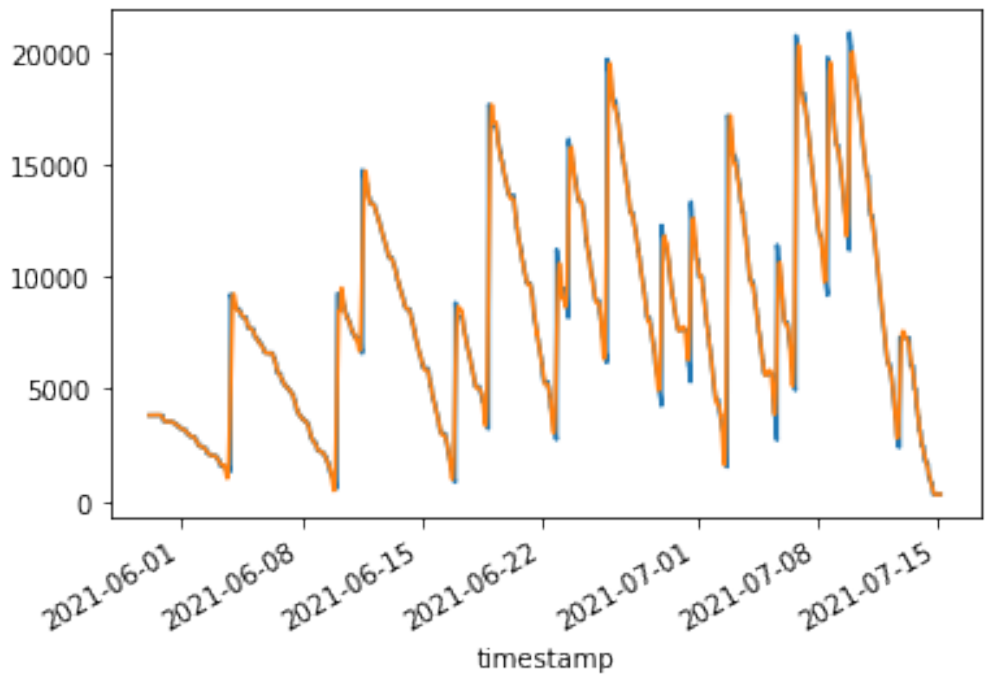
[74]:
```
print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
print('invalid silo weights: ', invalid_silo_weights)
```

```
print('total silo weights: ', total_silo_weights)
print('Batch feed consumption: ', silo_weight_374['consumption'].sum())
```

```
total silo weights:  61306
valid silo weights:  60786
invalid silo weights:  519
total silo weights:  61306
Batch feed consumption:  163495.0
```

[75]:
```
from scipy import signal

#b, a = signal.butter(3, 0.15, btype='lowpass', analog=False)
b, a = signal.butter(3, 0.0036, btype='lowpass', analog=False)
low_passed = signal.filtfilt(b, a, silo_weight_374['silo_weight'])

if 'low_passed' in silo_weight_374.keys():
    discard = silo_weight_374.pop('low_passed')
silo_weight_374.insert(1, 'low_passed', low_passed)

#silo_weight.loc['2021-06-03 21:45':'2021-06-03 23:30', 'silo_weight'].plot()
#silo_weight.loc['2021-06-03 21:45':'2021-06-03 23:30', 'low_passed'].plot()
silo_weight_374['silo_weight'].plot()
silo_weight_374['low_passed'].plot()
#silo_weight.loc['2021-06-03 19:46':'2021-06-03 19:47', 'low_passed'].plot()
#silo_weight.loc['2021-06-03 19:46':'2021-06-03 19:47', 'low_passed'].plot()
```

[75]: <AxesSubplot:xlabel='timestamp'>



[76]:
```
consumption = [0]
total_silo_weights = silo_weight_374['low_passed'].size
valid_silo_weights = 0
invalid_silo_weights = 0
for index in range(1, silo_weight_374['low_passed'].size):
    if silo_weight_374['low_passed'][index - 1] >=␣
 ↪(silo_weight_374['low_passed'][index]):
        instant_consumption = silo_weight_374['low_passed'][index - 1] -␣
 ↪silo_weight_374['low_passed'][index]
        instant_consumption = 0 if instant_consumption > 100 else␣
 ↪instant_consumption
        consumption.append(instant_consumption)
        valid_silo_weights += 1
    else:
        #consumption.append(consumption[index - 1])
        consumption.append(None)
        invalid_silo_weights += 1
if 'consumption_lp' in silo_weight_374.keys():
    discard = silo_weight_374.pop('consumption_lp')
silo_weight_374.insert(1, 'consumption_lp', consumption)

print('total silo weights: ', total_silo_weights)
print('valid silo weights: ', valid_silo_weights)
print('invalid silo weights: ', invalid_silo_weights)
print('total silo weights: ', total_silo_weights)
print('Batch feed consumption: ', silo_weight_374['consumption'].sum())
print('Batch feed consumption - Low Pass Filter: ',␣
 ↪silo_weight_374['consumption_lp'].sum())
```

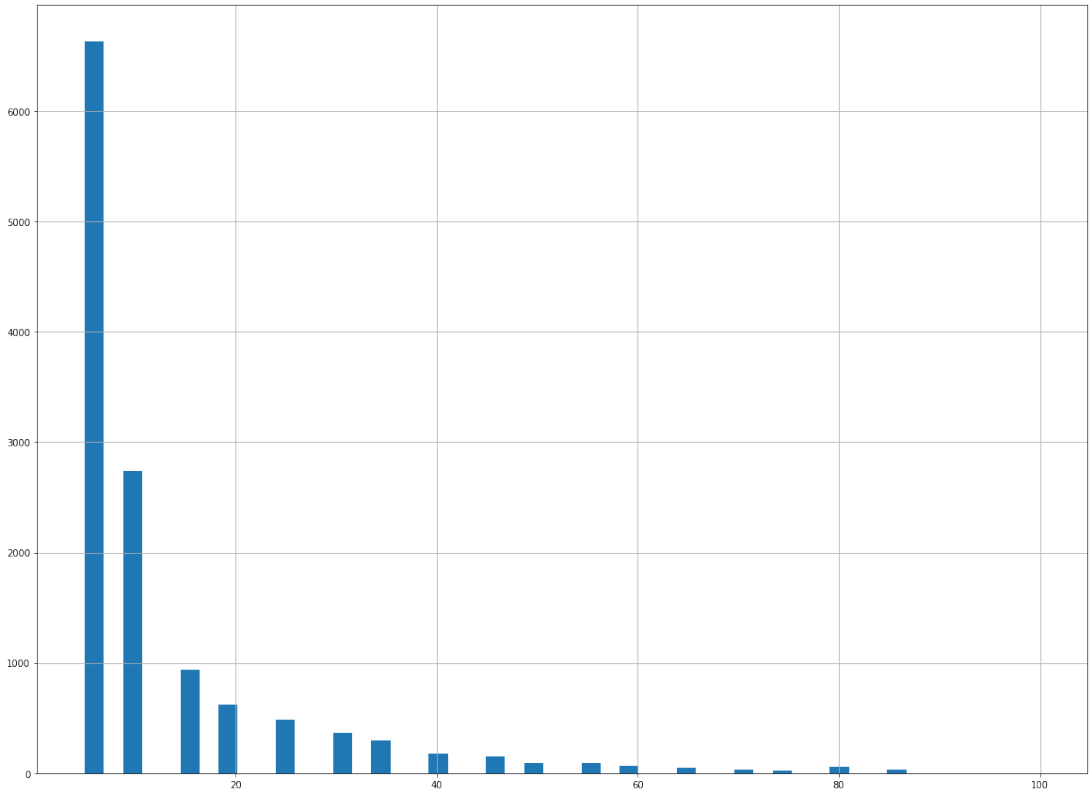
```
total silo weights:  61306
valid silo weights:  51626
invalid silo weights:  9679
total silo weights:  61306
Batch feed consumption:  163495.0
Batch feed consumption - Low Pass Filter:  152750.14496221126
```

2.0.1 Histograma do consumo de ração

Permite analisar os valores de consumo de ração mais comuns no intervalo de 1 minuto, com isto fica mais fácil fazer um filtro para desconsiderar outliers. Os outliers no nosso caso são o vento, alguém subindo no silo ou os momentos em que o silo é abastecido.

[119]:
```
%matplotlib inline
# a linha acima e necessaria apenas no notebook jupiter
import matplotlib.pyplot as plt
```

```
silo_weight_374.query('consumption > 0')['consumption'].hist(bins=50,␣
↪figsize=(20,15))
plt.show()
```



No histograma acima é possível constatar que a maioria das medidas de consumo minuto a minuto está entre 0Kg e 40Kg. Com base nisto, iremos desconsiderar variações de peso de silo maiores do que estes valores, pois quando ocorrerem possívelmente sejam ruídos.

2.1 Consumo Diário de Ração (Resampling)

Inicialmente faremos o cálculo da taxa de conversão alimentar baseado em dados diários, então iremos somar todo consumo de ração ocorrido no galpão durante o dia:

[78]:
```
columns = ['consumption', 'consumption_lp']
silo_consumption_374_daily_sum = silo_weight_374[columns].resample('D').sum()
silo_consumption_374_age = silo_weight_374['age'].resample('D').min()

if 'age' in silo_consumption_374_daily_sum.keys():
    discard = silo_consumption_374_daily_sum.pop('age')
silo_consumption_374_daily_sum.insert(0, 'age', silo_consumption_374_age)
```

2.1.1 Consumo acumulado de ração

Aqui será calculado o consumo acumulado de ração até a idade em questão

[79]:
```
from copy import deepcopy

daily_stats_374 = deepcopy(silo_consumption_374_daily_sum)

acc_consumption = []
acc_consumption.append(daily_stats_374['consumption'][0])
for index in range(1, daily_stats_374['consumption'].size):
    acc_consumption.append(acc_consumption[index - 1] +␣
 ↪daily_stats_374['consumption'][index])

if 'acc_consumption' in daily_stats_374.keys():
    discard = daily_stats_374.pop('acc_consumption')
daily_stats_374.insert(1, 'acc_consumption', acc_consumption)

acc_consumption = []
acc_consumption.append(daily_stats_374['consumption_lp'][0])
for index in range(1, daily_stats_374['consumption_lp'].size):
    acc_consumption.append(acc_consumption[index - 1] +␣
 ↪daily_stats_374['consumption_lp'][index])

if 'acc_consumption_lp' in daily_stats_374.keys():
    discard = daily_stats_374.pop('acc_consumption_lp')
daily_stats_374.insert(1, 'acc_consumption_lp', acc_consumption)
```

[80]:
```
daily_stats_374
```

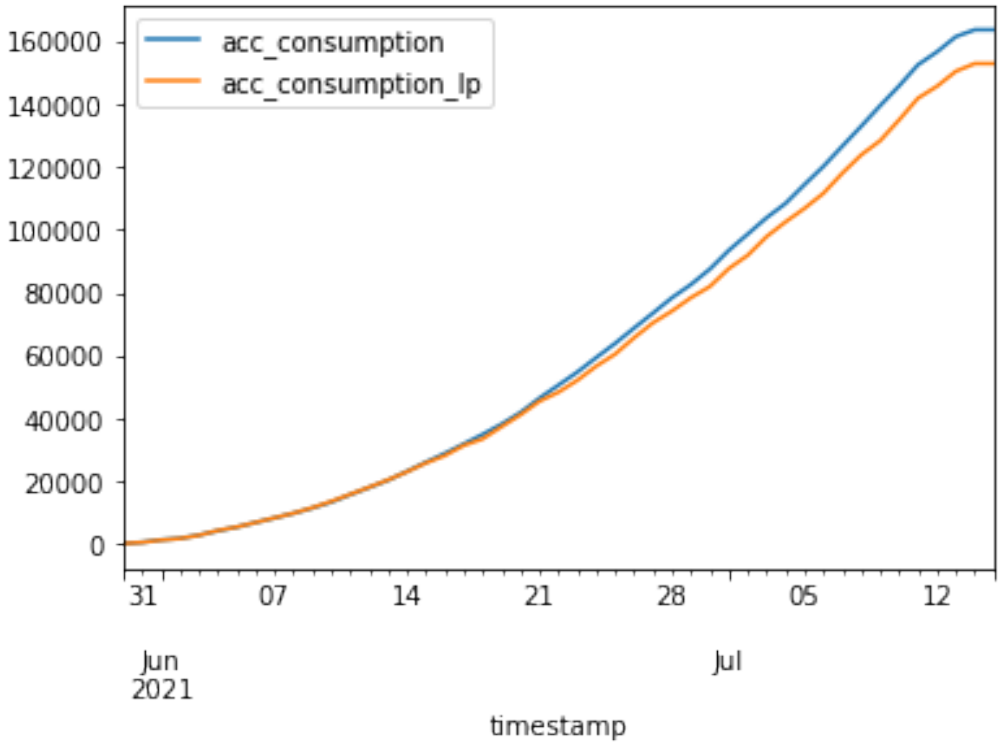
[80]:

timestamp	age	acc_consumption_lp	acc_consumption	\
2021-05-30 00:00:00+00:00	0	262.788043	295.0	
2021-05-31 00:00:00+00:00	1	638.981000	665.0	
2021-06-01 00:00:00+00:00	2	1365.192810	1435.0	
2021-06-02 00:00:00+00:00	3	1872.659801	1900.0	
2021-06-03 00:00:00+00:00	4	2866.976555	2925.0	
2021-06-04 00:00:00+00:00	5	4411.652100	4320.0	
2021-06-05 00:00:00+00:00	6	5509.710980	5475.0	
2021-06-06 00:00:00+00:00	7	6957.035424	6960.0	
2021-06-07 00:00:00+00:00	8	8430.622162	8390.0	
2021-06-08 00:00:00+00:00	9	9896.347883	9870.0	
2021-06-09 00:00:00+00:00	10	11651.004972	11550.0	
2021-06-10 00:00:00+00:00	11	13653.275648	13440.0	
2021-06-11 00:00:00+00:00	12	15904.095103	15805.0	
2021-06-12 00:00:00+00:00	13	18154.901384	18070.0	
2021-06-13 00:00:00+00:00	14	20496.043021	20380.0	
2021-06-14 00:00:00+00:00	15	23062.943250	23200.0	

```
2021-06-15 00:00:00+00:00   16        25870.906622          26105.0
2021-06-16 00:00:00+00:00   17        28202.250392          28890.0
2021-06-17 00:00:00+00:00   18        31440.727956          31895.0
2021-06-18 00:00:00+00:00   19        33523.917424          34935.0
2021-06-19 00:00:00+00:00   20        37395.101876          38210.0
2021-06-20 00:00:00+00:00   21        41186.978823          41910.0
2021-06-21 00:00:00+00:00   22        45484.638016          46455.0
2021-06-22 00:00:00+00:00   23        48346.089094          50585.0
2021-06-23 00:00:00+00:00   24        52117.976748          54740.0
2021-06-24 00:00:00+00:00   25        56625.234690          59360.0
2021-06-25 00:00:00+00:00   26        60461.526480          63870.0
2021-06-26 00:00:00+00:00   27        65651.169327          68730.0
2021-06-27 00:00:00+00:00   28        70406.163811          73500.0
2021-06-28 00:00:00+00:00   29        74133.361154          78360.0
2021-06-29 00:00:00+00:00   30        78366.394194          82550.0
2021-06-30 00:00:00+00:00   31        81906.914354          87450.0
2021-07-01 00:00:00+00:00   32        87607.627044          93395.0
2021-07-02 00:00:00+00:00   33        91836.444389          98565.0
2021-07-03 00:00:00+00:00   34        97774.816754         103785.0
2021-07-04 00:00:00+00:00   35       102443.408424         108305.0
2021-07-05 00:00:00+00:00   36       106740.462356         114295.0
2021-07-06 00:00:00+00:00   37       111523.607064         120100.0
2021-07-07 00:00:00+00:00   38       117933.198516         126495.0
2021-07-08 00:00:00+00:00   39       123745.086907         132855.0
2021-07-09 00:00:00+00:00   40       128257.564446         139365.0
2021-07-10 00:00:00+00:00   41       134845.337145         145655.0
2021-07-11 00:00:00+00:00   42       141755.848988         152375.0
2021-07-12 00:00:00+00:00   43       145526.899032         156470.0
2021-07-13 00:00:00+00:00   44       150302.994535         161310.0
2021-07-14 00:00:00+00:00   45       152750.144962         163495.0
2021-07-15 00:00:00+00:00   46       152750.144962         163495.0

                          consumption  consumption_lp
timestamp
2021-05-30 00:00:00+00:00        295.0      262.788043
2021-05-31 00:00:00+00:00        370.0      376.192957
2021-06-01 00:00:00+00:00        770.0      726.211810
2021-06-02 00:00:00+00:00        465.0      507.466991
2021-06-03 00:00:00+00:00       1025.0      994.316754
2021-06-04 00:00:00+00:00       1395.0     1544.675546
2021-06-05 00:00:00+00:00       1155.0     1098.058880
2021-06-06 00:00:00+00:00       1485.0     1447.324443
2021-06-07 00:00:00+00:00       1430.0     1473.586739
2021-06-08 00:00:00+00:00       1480.0     1465.725720
2021-06-09 00:00:00+00:00       1680.0     1754.657090
2021-06-10 00:00:00+00:00       1890.0     2002.270676
2021-06-11 00:00:00+00:00       2365.0     2250.819455
```

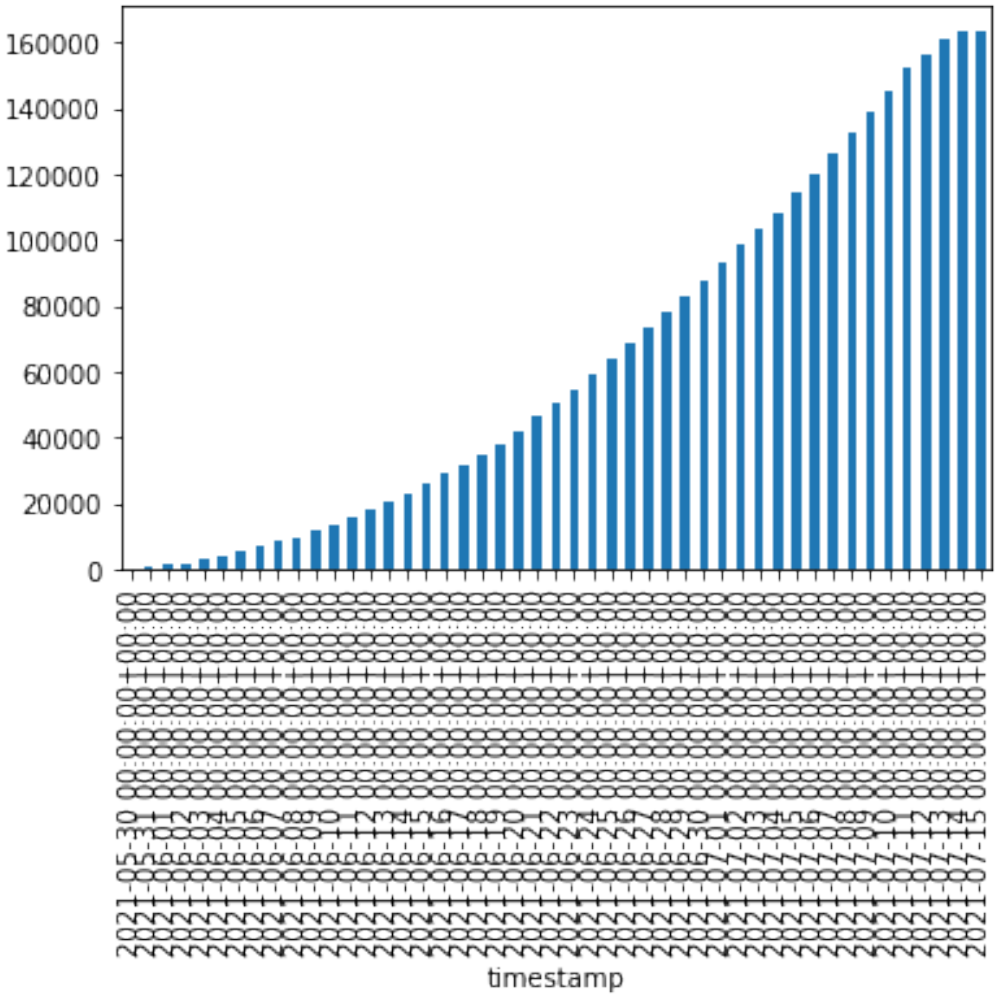
```
2021-06-12 00:00:00+00:00       2265.0     2250.806281
2021-06-13 00:00:00+00:00       2310.0     2341.141637
2021-06-14 00:00:00+00:00       2820.0     2566.900229
2021-06-15 00:00:00+00:00       2905.0     2807.963372
2021-06-16 00:00:00+00:00       2785.0     2331.343770
2021-06-17 00:00:00+00:00       3005.0     3238.477564
2021-06-18 00:00:00+00:00       3040.0     2083.189468
2021-06-19 00:00:00+00:00       3275.0     3871.184452
2021-06-20 00:00:00+00:00       3700.0     3791.876947
2021-06-21 00:00:00+00:00       4545.0     4297.659193
2021-06-22 00:00:00+00:00       4130.0     2861.451079
2021-06-23 00:00:00+00:00       4155.0     3771.887654
2021-06-24 00:00:00+00:00       4620.0     4507.257942
2021-06-25 00:00:00+00:00       4510.0     3836.291789
2021-06-26 00:00:00+00:00       4860.0     5189.642848
2021-06-27 00:00:00+00:00       4770.0     4754.994484
2021-06-28 00:00:00+00:00       4860.0     3727.197343
2021-06-29 00:00:00+00:00       4190.0     4233.033039
2021-06-30 00:00:00+00:00       4900.0     3540.520160
2021-07-01 00:00:00+00:00       5945.0     5700.712690
2021-07-02 00:00:00+00:00       5170.0     4228.817345
2021-07-03 00:00:00+00:00       5220.0     5938.372365
2021-07-04 00:00:00+00:00       4520.0     4668.591670
2021-07-05 00:00:00+00:00       5990.0     4297.053931
2021-07-06 00:00:00+00:00       5805.0     4783.144708
2021-07-07 00:00:00+00:00       6395.0     6409.591452
2021-07-08 00:00:00+00:00       6360.0     5811.888392
2021-07-09 00:00:00+00:00       6510.0     4512.477539
2021-07-10 00:00:00+00:00       6290.0     6587.772698
2021-07-11 00:00:00+00:00       6720.0     6910.511843
2021-07-12 00:00:00+00:00       4095.0     3771.050044
2021-07-13 00:00:00+00:00       4840.0     4776.095504
2021-07-14 00:00:00+00:00       2185.0     2447.150427
2021-07-15 00:00:00+00:00          0.0        0.000000
```

[89]:
```
daily_stats_374['acc_consumption'].plot(legend=True)
daily_stats_374['acc_consumption_lp'].plot(legend=True).figure.
 ↪savefig('acc-consumption-filtered.svg')
```



[88]:
```
daily_stats_374['acc_consumption'].plot.bar()
```

[88]:
```
<AxesSubplot:xlabel='timestamp'>
```



3 Aves vivas

A sequir será calculada a quantidade de aves vivas em cada dia do lote. Isto será feito reduzindo da quantidade de aves alojadas a mortalidade diária.

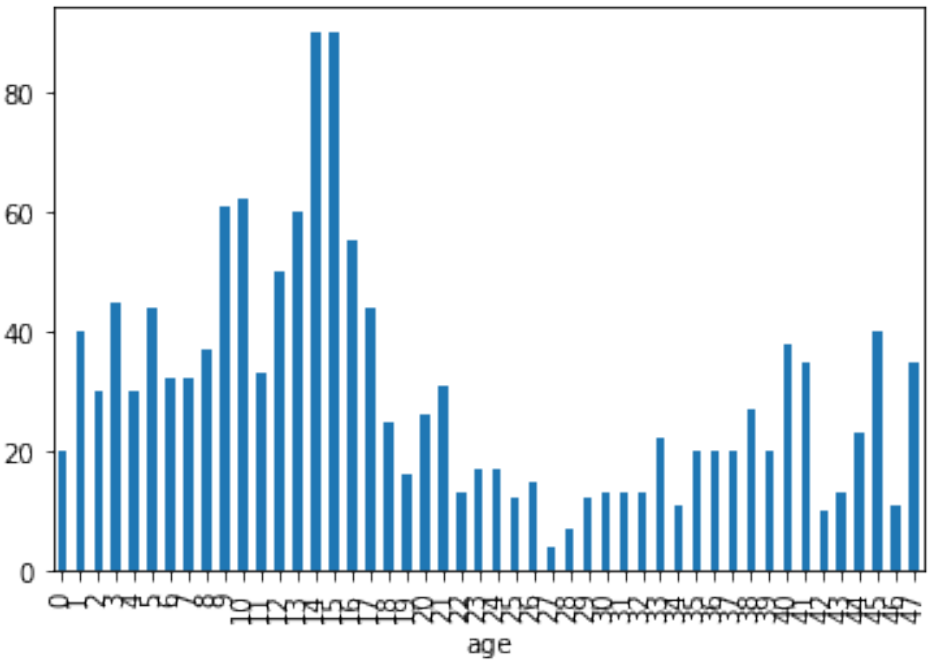
[90]:
```
sql='''SELECT quantity_housed, initial_date, final_date
       FROM batch_batch
       WHERE id = 374;'''
quantity_housed_374 = load_db_data(sql=sql)

quantity_housed_374 = quantity_housed_374['quantity_housed'][0]
```

[91]:
```
sql='''SELECT quantity, age
       FROM mortality_mortality
```

```
        WHERE batch_id = 374;'''
mortality_374 = load_db_data(sql=sql).set_index('age')
```

[92]:
```
mortality_374['quantity'].plot.bar()
```

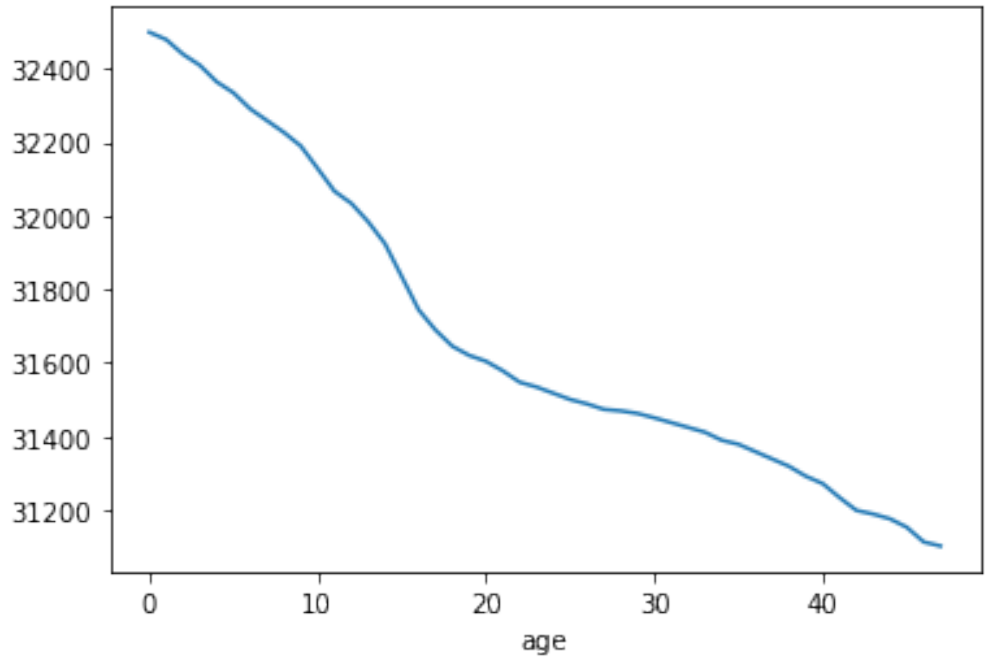
[92]: <AxesSubplot:xlabel='age'>



[93]:
```
living_birds_374 = [quantity_housed_374]
for index in range(1, mortality_374['quantity'].size):
    living_birds_374.append(living_birds_374[index - 1] -␣
 ↪mortality_374['quantity'][index - 1])

if 'living_birds' in mortality_374.keys():
    discard = mortality_374.pop('living_birds')
mortality_374.insert(1, 'living_birds', living_birds_374)
```

[94]:
```
mortality_374['living_birds'].plot()
```

[94]: <AxesSubplot:xlabel='age'>



[95]:
```
daily_stats_374['timestamp'] = daily_stats_374.index
```

[96]:
```
daily_stats_374 = daily_stats_374.set_index('age')
```

[97]:
```
if 'living_birds' in daily_stats_374.keys():
    discard = daily_stats_374.pop('living_birds')
```

[98]:
```
daily_stats_374.insert(0, 'living_birds', mortality_374['living_birds'])
```

[99]:
```
daily_stats_374
```

[99]:

age	living_birds	acc_consumption_lp	acc_consumption	consumption
0	32500	262.788043	295.0	295.0
1	32480	638.981000	665.0	370.0
2	32440	1365.192810	1435.0	770.0
3	32410	1872.659801	1900.0	465.0
4	32365	2866.976555	2925.0	1025.0
5	32335	4411.652100	4320.0	1395.0
6	32291	5509.710980	5475.0	1155.0
7	32259	6957.035424	6960.0	1485.0
8	32227	8430.622162	8390.0	1430.0
9	32190	9896.347883	9870.0	1480.0
10	32129	11651.004972	11550.0	1680.0

11	32067	13653.275648	13440.0	1890.0
12	32034	15904.095103	15805.0	2365.0
13	31984	18154.901384	18070.0	2265.0
14	31924	20496.043021	20380.0	2310.0
15	31834	23062.943250	23200.0	2820.0
16	31744	25870.906622	26105.0	2905.0
17	31689	28202.250392	28890.0	2785.0
18	31645	31440.727956	31895.0	3005.0
19	31620	33523.917424	34935.0	3040.0
20	31604	37395.101876	38210.0	3275.0
21	31578	41186.978823	41910.0	3700.0
22	31547	45484.638016	46455.0	4545.0
23	31534	48346.089094	50585.0	4130.0
24	31517	52117.976748	54740.0	4155.0
25	31500	56625.234690	59360.0	4620.0
26	31488	60461.526480	63870.0	4510.0
27	31473	65651.169327	68730.0	4860.0
28	31469	70406.163811	73500.0	4770.0
29	31462	74133.361154	78360.0	4860.0
30	31450	78366.394194	82550.0	4190.0
31	31437	81906.914354	87450.0	4900.0
32	31424	87607.627044	93395.0	5945.0
33	31411	91836.444389	98565.0	5170.0
34	31389	97774.816754	103785.0	5220.0
35	31378	102443.408424	108305.0	4520.0
36	31358	106740.462356	114295.0	5990.0
37	31338	111523.607064	120100.0	5805.0
38	31318	117933.198516	126495.0	6395.0
39	31291	123745.086907	132855.0	6360.0
40	31271	128257.564446	139365.0	6510.0
41	31233	134845.337145	145655.0	6290.0
42	31198	141755.848988	152375.0	6720.0
43	31188	145526.899032	156470.0	4095.0
44	31175	150302.994535	161310.0	4840.0
45	31152	152750.144962	163495.0	2185.0
46	31112	152750.144962	163495.0	0.0

age	consumption_lp	timestamp
0	262.788043	2021-05-30 00:00:00+00:00
1	376.192957	2021-05-31 00:00:00+00:00
2	726.211810	2021-06-01 00:00:00+00:00
3	507.466991	2021-06-02 00:00:00+00:00
4	994.316754	2021-06-03 00:00:00+00:00
5	1544.675546	2021-06-04 00:00:00+00:00
6	1098.058880	2021-06-05 00:00:00+00:00
7	1447.324443	2021-06-06 00:00:00+00:00

```
8       1473.586739 2021-06-07 00:00:00+00:00
9       1465.725720 2021-06-08 00:00:00+00:00
10      1754.657090 2021-06-09 00:00:00+00:00
11      2002.270676 2021-06-10 00:00:00+00:00
12      2250.819455 2021-06-11 00:00:00+00:00
13      2250.806281 2021-06-12 00:00:00+00:00
14      2341.141637 2021-06-13 00:00:00+00:00
15      2566.900229 2021-06-14 00:00:00+00:00
16      2807.963372 2021-06-15 00:00:00+00:00
17      2331.343770 2021-06-16 00:00:00+00:00
18      3238.477564 2021-06-17 00:00:00+00:00
19      2083.189468 2021-06-18 00:00:00+00:00
20      3871.184452 2021-06-19 00:00:00+00:00
21      3791.876947 2021-06-20 00:00:00+00:00
22      4297.659193 2021-06-21 00:00:00+00:00
23      2861.451079 2021-06-22 00:00:00+00:00
24      3771.887654 2021-06-23 00:00:00+00:00
25      4507.257942 2021-06-24 00:00:00+00:00
26      3836.291789 2021-06-25 00:00:00+00:00
27      5189.642848 2021-06-26 00:00:00+00:00
28      4754.994484 2021-06-27 00:00:00+00:00
29      3727.197343 2021-06-28 00:00:00+00:00
30      4233.033039 2021-06-29 00:00:00+00:00
31      3540.520160 2021-06-30 00:00:00+00:00
32      5700.712690 2021-07-01 00:00:00+00:00
33      4228.817345 2021-07-02 00:00:00+00:00
34      5938.372365 2021-07-03 00:00:00+00:00
35      4668.591670 2021-07-04 00:00:00+00:00
36      4297.053931 2021-07-05 00:00:00+00:00
37      4783.144708 2021-07-06 00:00:00+00:00
38      6409.591452 2021-07-07 00:00:00+00:00
39      5811.888392 2021-07-08 00:00:00+00:00
40      4512.477539 2021-07-09 00:00:00+00:00
41      6587.772698 2021-07-10 00:00:00+00:00
42      6910.511843 2021-07-11 00:00:00+00:00
43      3771.050044 2021-07-12 00:00:00+00:00
44      4776.095504 2021-07-13 00:00:00+00:00
45      2447.150427 2021-07-14 00:00:00+00:00
46         0.000000 2021-07-15 00:00:00+00:00
```

3.0.1 Consumo por ave

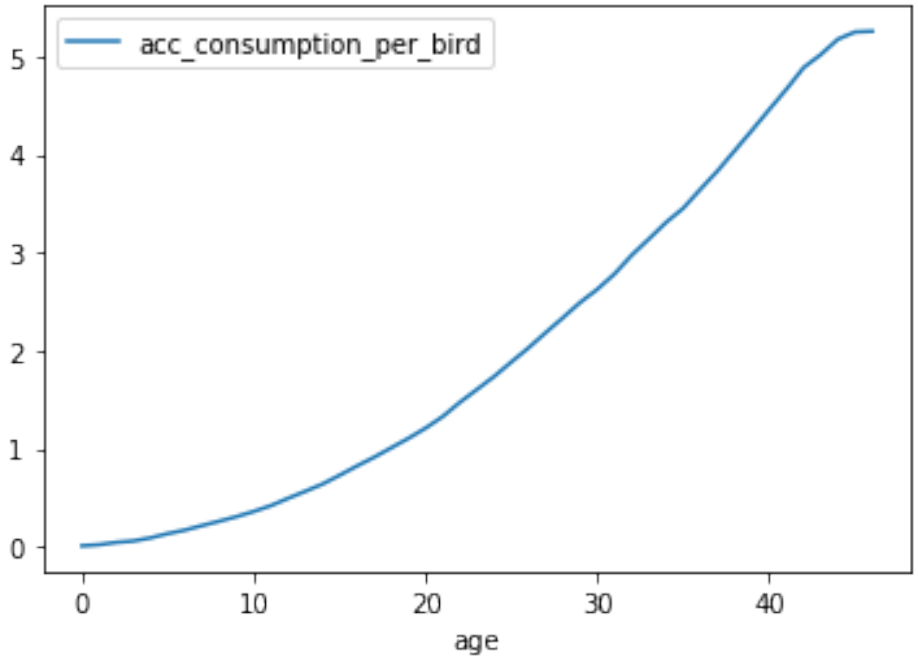
Como a quantidade de aves alojadas diminui durante o lote devido a mortalidade, o consumo por ave poderia ser obtido através da quantidade consumida no dia, dividido pela quantidade de aves vivas neste mesmo dia.

Contudo, é uma prática comum dividir o total consumido pela quantidade de aves vivas, de modo que quanto maior a mortalidade, maior será a quantidade consumida por ave obtida neste cálculo

grosseiro. Portanto, o aumento da mortalidade faz com que o consumo por ave viva fique maior, prejudicando a Conversão Alimentar, o que não prejudicará nossa análise.

[100]:
```
acc_consumption_per_bird_374 = []
for index in range(0, daily_stats_374['living_birds'].size):
    acc_consumption_per_bird_374.
 ↪append(daily_stats_374['acc_consumption'][index]/
 ↪daily_stats_374['living_birds'][index])

if 'acc_consumption_per_bird' in daily_stats_374.keys():
    discard = daily_stats_374.pop('acc_consumption_per_bird')
daily_stats_374.insert(1, 'acc_consumption_per_bird',␣
 ↪acc_consumption_per_bird_374)
```

[120]:
```
daily_stats_374['acc_consumption_per_bird'].plot(legend=True).figure.
 ↪savefig('acc-consumption-per-bird.svg')
```



4 Peso Médio

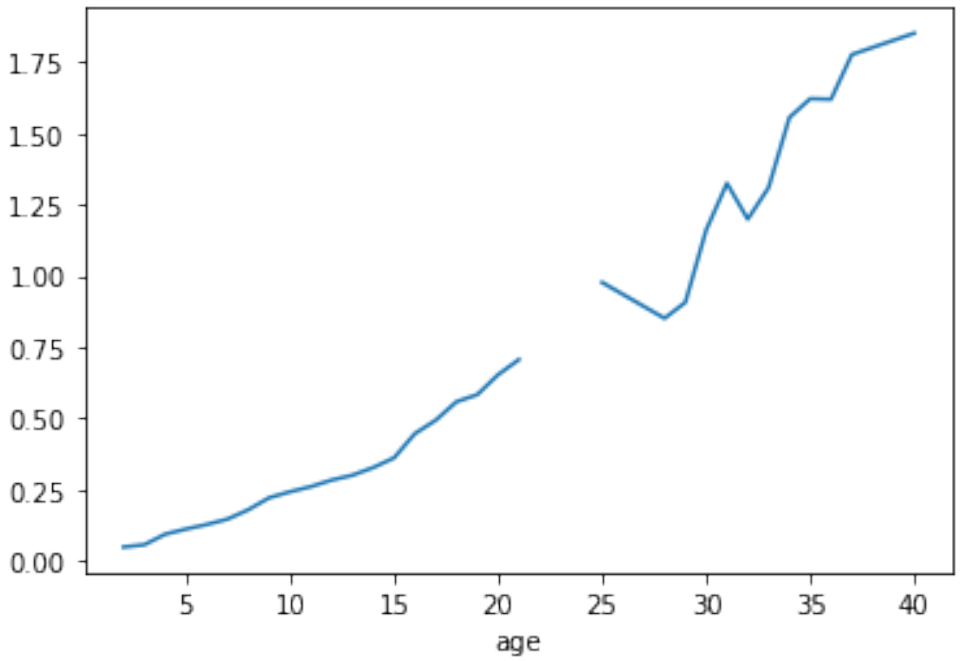
O peso médio diário das aves pode ser obtido da tabela de análise de peso.

[102]:
```
sql='''SELECT wa.age,
              wa.aw / 1000 AS average_weight,
              wa.mac AS device,
```

```
            wa.rec_at AS t_rec_at,
            TO_TIMESTAMP(wa.rec_at, 'YYYY-MM-DD HH24:MI:SS') AS rec_at
    FROM weight_analysis_weightanalysis AS wa
    INNER JOIN batch_batch AS b
    ON b.device = wa.mac
    WHERE b.device = wa.mac
      AND TO_DATE(LEFT(wa.rec_at, 10), 'YYYY-MM-DD') >= b.initial_date
      AND TO_DATE(LEFT(wa.rec_at, 10), 'YYYY-MM-DD') <= b.final_date
      AND b.id = 374
    ORDER BY age;'''
average_weight = load_db_data(sql=sql).set_index('age')
```

[103]:
```
average_weight['average_weight'].plot()
```

[103]: <AxesSubplot:xlabel='age'>



Agora vamos adicionar o peso médio das aves ao nosso conjunto daily_stats:

[104]:
```
if 'average_weight' in daily_stats_374.keys():
    discard = daily_stats_374.pop('average_weight')
daily_stats_374.insert(1, 'average_weight', average_weight['average_weight'])
```

5 Conversão Alimentar

[105]:
```
daily_stats_374['fcr'] = (daily_stats_374['acc_consumption_per_bird']/
↪daily_stats_374['average_weight'])
```

[106]:
```
daily_stats_374[['acc_consumption_per_bird', 'average_weight', 'fcr']]
```

[106]:

age	acc_consumption_per_bird	average_weight	fcr
0	0.009077	NaN	NaN
1	0.020474	NaN	NaN
2	0.044236	0.048	0.921573
3	0.058624	0.056	1.046855
4	0.090375	0.093	0.971779
5	0.133601	0.111	1.203616
6	0.169552	0.127	1.335054
7	0.215754	0.146	1.477765
8	0.260341	0.179	1.454417
9	0.306617	0.221	1.387407
10	0.359488	0.242	1.485489
11	0.419122	0.260	1.612009
12	0.493382	0.283	1.743399
13	0.564970	0.300	1.883233
14	0.638391	0.327	1.952267
15	0.728781	0.361	2.018783
16	0.822360	0.446	1.843857
17	0.911673	0.493	1.849235
18	1.007900	0.558	1.806273
19	1.104839	0.583	1.895092
20	1.209024	0.653	1.851492
21	1.327190	0.706	1.879872
22	1.472565	NaN	NaN
23	1.604142	NaN	NaN
24	1.736840	NaN	NaN
25	1.884444	0.977	1.928807
26	2.028392	NaN	NaN
27	2.183777	NaN	NaN
28	2.335632	0.851	2.744573
29	2.490624	0.907	2.746002
30	2.624801	1.161	2.260811
31	2.781754	1.325	2.099437
32	2.972091	1.199	2.478809
33	3.137913	1.310	2.395354
34	3.306413	1.556	2.124944
35	3.451622	1.622	2.128004
36	3.644843	1.620	2.249903
37	3.832408	1.777	2.156673

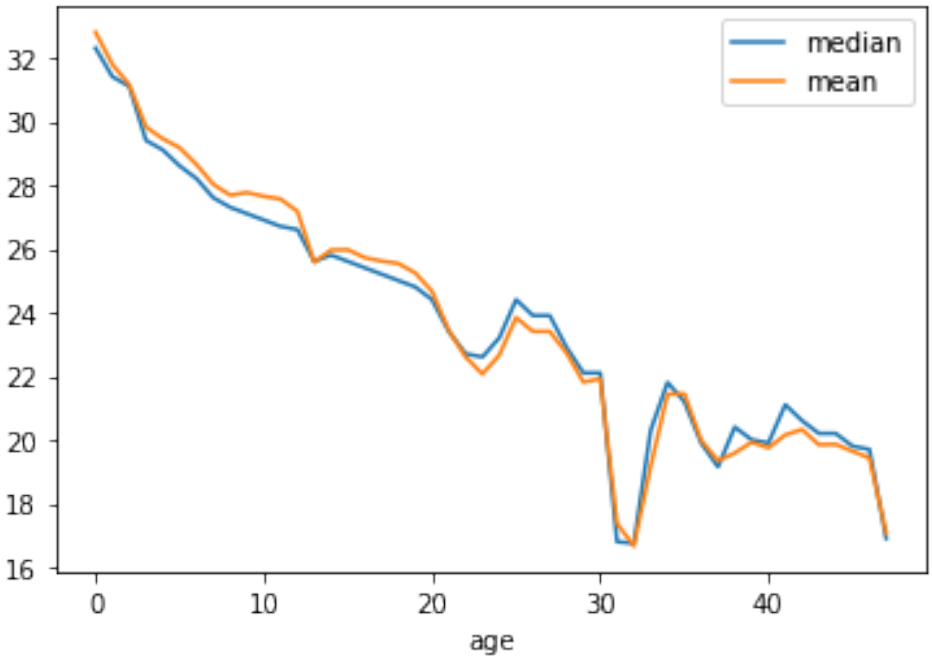
```
38                  4.039051            NaN       NaN
39                  4.245790            NaN       NaN
40                  4.456685          1.852  2.406417
41                  4.663497            NaN       NaN
42                  4.884127            NaN       NaN
43                  5.016994            NaN       NaN
44                  5.174338            NaN       NaN
45                  5.248299            NaN       NaN
46                  5.255046            NaN       NaN
```

5.1 Ambiência média do dia

[107]:
```
sql='''SELECT r.age,
            r.at AS average_temperature,
            r.ah AS average_humidity,
            r.p1 AS pressure,
            r.air_co21 AS co2,
            TO_TIMESTAMP(r.rec_at, 'YYYY-MM-DD HH24:MI:SS') AS rec_at
    FROM reading_reading AS r
    INNER JOIN batch_batch AS b
    ON b.device = r.mac
    WHERE b.device = r.mac
      AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') >= b.initial_date
      AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') <= b.final_date
      AND b.id = 374
    ORDER BY age;'''
average_ambience = load_db_data(sql=sql).set_index('age')
```

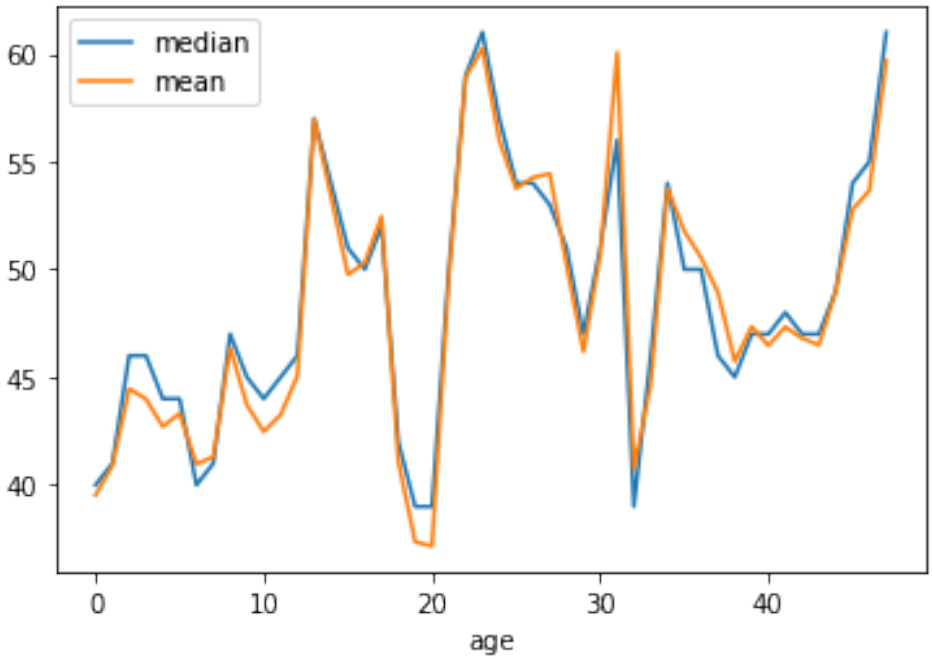
[108]:
```
average_ambience.groupby(['age'])['average_temperature'].median().
 ↪plot(label='median', legend=True)
average_ambience.groupby(['age'])['average_temperature'].mean().
 ↪plot(label='mean', legend=True)
```

[108]:
```
<AxesSubplot:xlabel='age'>
```



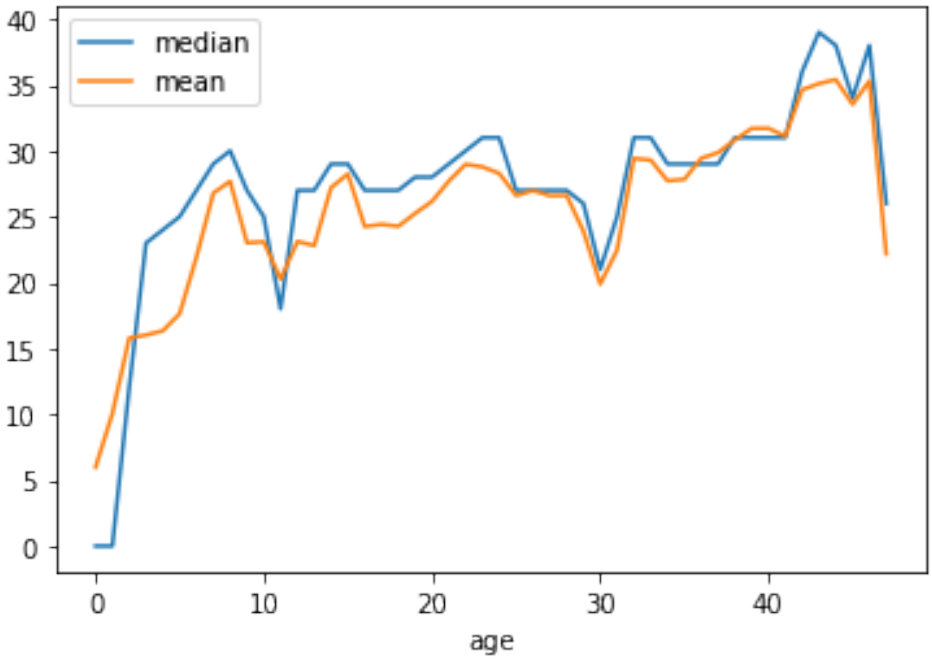
[109]:
```
average_ambience.groupby(['age'])['average_humidity'].median().
 ↪plot(label='median', legend=True)
average_ambience.groupby(['age'])['average_humidity'].mean().plot(label='mean',␣
 ↪legend=True)
```

[109]: <AxesSubplot:xlabel='age'>



[110]:
```
average_ambience.groupby(['age'])['pressure'].median().plot(label='median',␣
↪legend=True)
average_ambience.groupby(['age'])['pressure'].mean().plot(label='mean',␣
↪legend=True)
```

[110]: <AxesSubplot:xlabel='age'>



[111]:
```
average_temperature = average_ambience.groupby(['age'])['average_temperature'].
 ↪median()
```

[112]:
```
average_ambience_daily = average_ambience.
 ↪groupby(['age'])['average_temperature'].median()
```

[113]:
```
if 'average_temperature' in daily_stats_374.keys():
    discard = daily_stats_374.pop('average_temperature')
daily_stats_374.insert(1, 'average_temperature', average_ambience.
 ↪groupby(['age'])['average_temperature'].median())

if 'average_humidity' in daily_stats_374.keys():
    discard = daily_stats_374.pop('average_humidity')
daily_stats_374.insert(1, 'average_humidity', average_ambience.
 ↪groupby(['age'])['average_humidity'].median())

if 'pressure' in daily_stats_374.keys():
    discard = daily_stats_374.pop('pressure')
daily_stats_374.insert(1, 'pressure', average_ambience.
 ↪groupby(['age'])['pressure'].median())
```

[114]:
```
daily_stats_374
```

[114]:

age	living_birds	pressure	average_humidity	average_temperature
0	32500	0.0	40.0	32.30
1	32480	0.0	41.0	31.40
2	32440	12.0	46.0	31.10
3	32410	23.0	46.0	29.40
4	32365	24.0	44.0	29.10
5	32335	25.0	44.0	28.60
6	32291	27.0	40.0	28.20
7	32259	29.0	41.0	27.60
8	32227	30.0	47.0	27.30
9	32190	27.0	45.0	27.10
10	32129	25.0	44.0	26.90
11	32067	18.0	45.0	26.70
12	32034	27.0	46.0	26.60
13	31984	27.0	57.0	25.60
14	31924	29.0	54.0	25.80
15	31834	29.0	51.0	25.60
16	31744	27.0	50.0	25.40
17	31689	27.0	52.0	25.20
18	31645	27.0	42.0	25.00
19	31620	28.0	39.0	24.80
20	31604	28.0	39.0	24.40
21	31578	29.0	50.0	23.40
22	31547	30.0	59.0	22.70
23	31534	31.0	61.0	22.60
24	31517	31.0	57.0	23.20
25	31500	27.0	54.0	24.40
26	31488	27.0	54.0	23.90
27	31473	27.0	53.0	23.90
28	31469	27.0	51.0	22.90
29	31462	26.0	47.0	22.10
30	31450	21.0	51.0	22.10
31	31437	25.0	56.0	16.80
32	31424	31.0	39.0	16.75
33	31411	31.0	46.0	20.30
34	31389	29.0	54.0	21.80
35	31378	29.0	50.0	21.20
36	31358	29.0	50.0	19.90
37	31338	29.0	46.0	19.15
38	31318	31.0	45.0	20.40
39	31291	31.0	47.0	20.00
40	31271	31.0	47.0	19.90
41	31233	31.0	48.0	21.10
42	31198	36.0	47.0	20.60
43	31188	39.0	47.0	20.20
44	31175	38.0	49.0	20.20

```
45          31152      34.0              54.0                19.80
46          31112      38.0              55.0                19.70

     average_weight  acc_consumption_per_bird  acc_consumption_lp  \
age
0               NaN                  0.009077          262.788043
1               NaN                  0.020474          638.981000
2             0.048                  0.044236         1365.192810
3             0.056                  0.058624         1872.659801
4             0.093                  0.090375         2866.976555
5             0.111                  0.133601         4411.652100
6             0.127                  0.169552         5509.710980
7             0.146                  0.215754         6957.035424
8             0.179                  0.260341         8430.622162
9             0.221                  0.306617         9896.347883
10            0.242                  0.359488        11651.004972
11            0.260                  0.419122        13653.275648
12            0.283                  0.493382        15904.095103
13            0.300                  0.564970        18154.901384
14            0.327                  0.638391        20496.043021
15            0.361                  0.728781        23062.943250
16            0.446                  0.822360        25870.906622
17            0.493                  0.911673        28202.250392
18            0.558                  1.007900        31440.727956
19            0.583                  1.104839        33523.917424
20            0.653                  1.209024        37395.101876
21            0.706                  1.327190        41186.978823
22              NaN                  1.472565        45484.638016
23              NaN                  1.604142        48346.089094
24              NaN                  1.736840        52117.976748
25            0.977                  1.884444        56625.234690
26              NaN                  2.028392        60461.526480
27              NaN                  2.183777        65651.169327
28            0.851                  2.335632        70406.163811
29            0.907                  2.490624        74133.361154
30            1.161                  2.624801        78366.394194
31            1.325                  2.781754        81906.914354
32            1.199                  2.972091        87607.627044
33            1.310                  3.137913        91836.444389
34            1.556                  3.306413        97774.816754
35            1.622                  3.451622       102443.408424
36            1.620                  3.644843       106740.462356
37            1.777                  3.832408       111523.607064
38              NaN                  4.039051       117933.198516
39              NaN                  4.245790       123745.086907
40            1.852                  4.456685       128257.564446
41              NaN                  4.663497       134845.337145
```

```
42              NaN                4.884127       141755.848988
43              NaN                5.016994       145526.899032
44              NaN                5.174338       150302.994535
45              NaN                5.248299       152750.144962
46              NaN                5.255046       152750.144962

     acc_consumption  consumption  consumption_lp                 timestamp  \
age
0              295.0        295.0      262.788043 2021-05-30 00:00:00+00:00
1              665.0        370.0      376.192957 2021-05-31 00:00:00+00:00
2             1435.0        770.0      726.211810 2021-06-01 00:00:00+00:00
3             1900.0        465.0      507.466991 2021-06-02 00:00:00+00:00
4             2925.0       1025.0      994.316754 2021-06-03 00:00:00+00:00
5             4320.0       1395.0     1544.675546 2021-06-04 00:00:00+00:00
6             5475.0       1155.0     1098.058880 2021-06-05 00:00:00+00:00
7             6960.0       1485.0     1447.324443 2021-06-06 00:00:00+00:00
8             8390.0       1430.0     1473.586739 2021-06-07 00:00:00+00:00
9             9870.0       1480.0     1465.725720 2021-06-08 00:00:00+00:00
10           11550.0       1680.0     1754.657090 2021-06-09 00:00:00+00:00
11           13440.0       1890.0     2002.270676 2021-06-10 00:00:00+00:00
12           15805.0       2365.0     2250.819455 2021-06-11 00:00:00+00:00
13           18070.0       2265.0     2250.806281 2021-06-12 00:00:00+00:00
14           20380.0       2310.0     2341.141637 2021-06-13 00:00:00+00:00
15           23200.0       2820.0     2566.900229 2021-06-14 00:00:00+00:00
16           26105.0       2905.0     2807.963372 2021-06-15 00:00:00+00:00
17           28890.0       2785.0     2331.343770 2021-06-16 00:00:00+00:00
18           31895.0       3005.0     3238.477564 2021-06-17 00:00:00+00:00
19           34935.0       3040.0     2083.189468 2021-06-18 00:00:00+00:00
20           38210.0       3275.0     3871.184452 2021-06-19 00:00:00+00:00
21           41910.0       3700.0     3791.876947 2021-06-20 00:00:00+00:00
22           46455.0       4545.0     4297.659193 2021-06-21 00:00:00+00:00
23           50585.0       4130.0     2861.451079 2021-06-22 00:00:00+00:00
24           54740.0       4155.0     3771.887654 2021-06-23 00:00:00+00:00
25           59360.0       4620.0     4507.257942 2021-06-24 00:00:00+00:00
26           63870.0       4510.0     3836.291789 2021-06-25 00:00:00+00:00
27           68730.0       4860.0     5189.642848 2021-06-26 00:00:00+00:00
28           73500.0       4770.0     4754.994484 2021-06-27 00:00:00+00:00
29           78360.0       4860.0     3727.197343 2021-06-28 00:00:00+00:00
30           82550.0       4190.0     4233.033039 2021-06-29 00:00:00+00:00
31           87450.0       4900.0     3540.520160 2021-06-30 00:00:00+00:00
32           93395.0       5945.0     5700.712690 2021-07-01 00:00:00+00:00
33           98565.0       5170.0     4228.817345 2021-07-02 00:00:00+00:00
34          103785.0       5220.0     5938.372365 2021-07-03 00:00:00+00:00
35          108305.0       4520.0     4668.591670 2021-07-04 00:00:00+00:00
36          114295.0       5990.0     4297.053931 2021-07-05 00:00:00+00:00
37          120100.0       5805.0     4783.144708 2021-07-06 00:00:00+00:00
38          126495.0       6395.0     6409.591452 2021-07-07 00:00:00+00:00
```

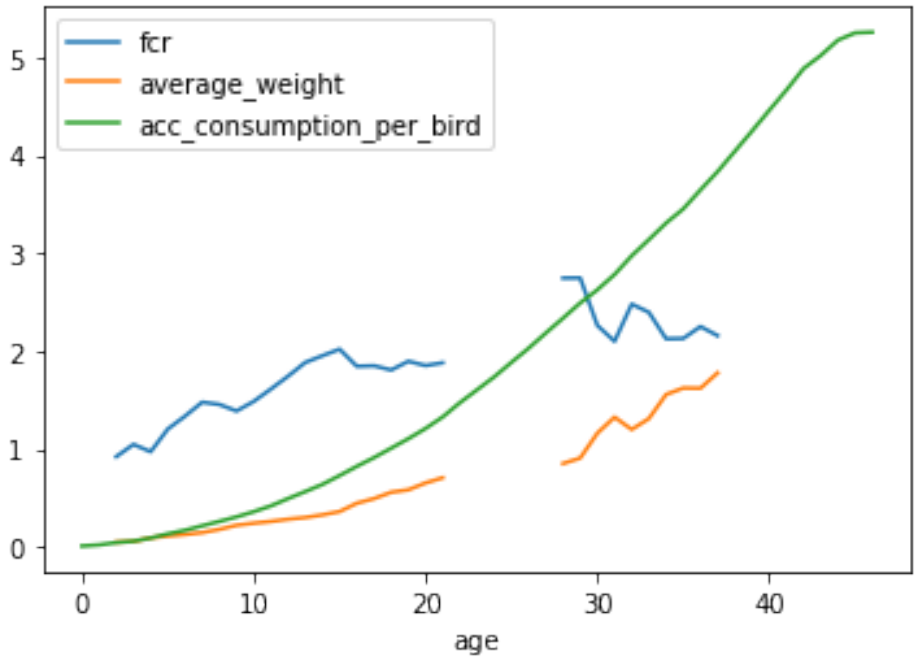
```
39          132855.0       6360.0     5811.888392 2021-07-08 00:00:00+00:00
40          139365.0       6510.0     4512.477539 2021-07-09 00:00:00+00:00
41          145655.0       6290.0     6587.772698 2021-07-10 00:00:00+00:00
42          152375.0       6720.0     6910.511843 2021-07-11 00:00:00+00:00
43          156470.0       4095.0     3771.050044 2021-07-12 00:00:00+00:00
44          161310.0       4840.0     4776.095504 2021-07-13 00:00:00+00:00
45          163495.0       2185.0     2447.150427 2021-07-14 00:00:00+00:00
46          163495.0          0.0        0.000000 2021-07-15 00:00:00+00:00
```

```
          fcr
age
0         NaN
1         NaN
2    0.921573
3    1.046855
4    0.971779
5    1.203616
6    1.335054
7    1.477765
8    1.454417
9    1.387407
10   1.485489
11   1.612009
12   1.743399
13   1.883233
14   1.952267
15   2.018783
16   1.843857
17   1.849235
18   1.806273
19   1.895092
20   1.851492
21   1.879872
22        NaN
23        NaN
24        NaN
25   1.928807
26        NaN
27        NaN
28   2.744573
29   2.746002
30   2.260811
31   2.099437
32   2.478809
33   2.395354
34   2.124944
35   2.128004
```

```
36   2.249903
37   2.156673
38        NaN
39        NaN
40   2.406417
41        NaN
42        NaN
43        NaN
44        NaN
45        NaN
46        NaN
```

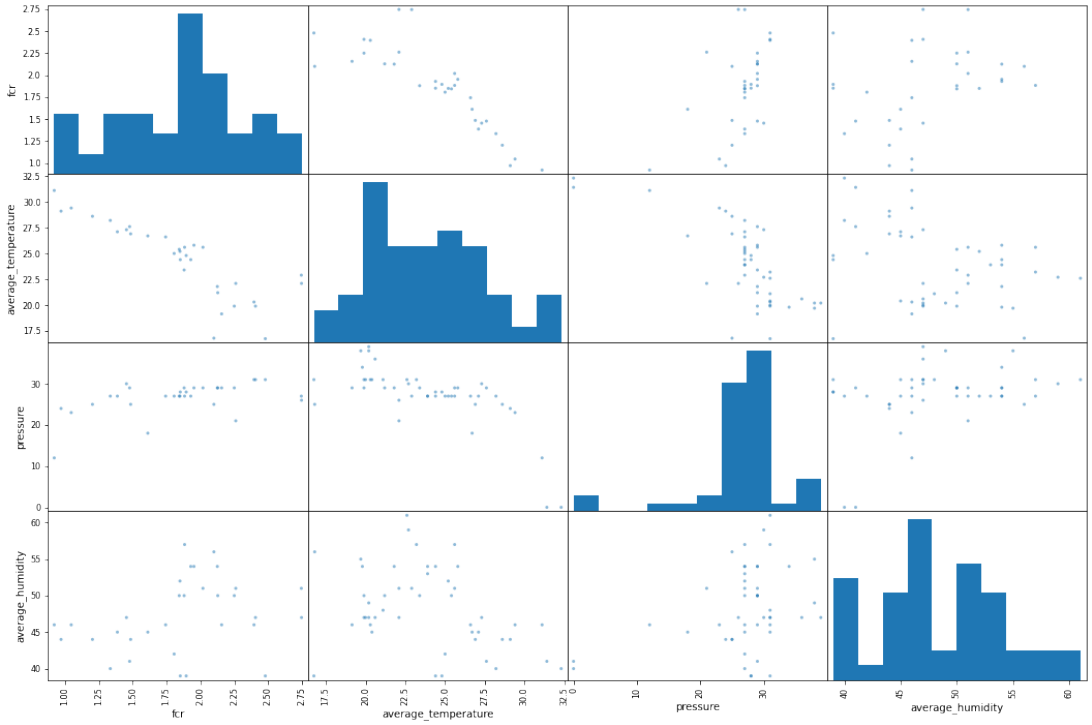
[117]:
```
daily_stats_374['fcr'].plot(legend=True)
daily_stats_374['average_weight'].plot(legend=True)
daily_stats_374['acc_consumption_per_bird'].plot(legend=True)
```

[117]: <AxesSubplot:xlabel='age'>



[116]:
```
from pandas.plotting import scatter_matrix

attributes = ["fcr", "average_temperature", "pressure", "average_humidity"]
scatter_matrix(daily_stats_374[attributes], figsize=(18, 12))
```

[116]:
```
array([[<AxesSubplot:xlabel='fcr', ylabel='fcr'>,
        <AxesSubplot:xlabel='average_temperature', ylabel='fcr'>,
        <AxesSubplot:xlabel='pressure', ylabel='fcr'>,
        <AxesSubplot:xlabel='average_humidity', ylabel='fcr'>],
       [<AxesSubplot:xlabel='fcr', ylabel='average_temperature'>,
        <AxesSubplot:xlabel='average_temperature',
    ylabel='average_temperature'>,
        <AxesSubplot:xlabel='pressure', ylabel='average_temperature'>,
        <AxesSubplot:xlabel='average_humidity', ylabel='average_temperature'>],
       [<AxesSubplot:xlabel='fcr', ylabel='pressure'>,
        <AxesSubplot:xlabel='average_temperature', ylabel='pressure'>,
        <AxesSubplot:xlabel='pressure', ylabel='pressure'>,
        <AxesSubplot:xlabel='average_humidity', ylabel='pressure'>],
       [<AxesSubplot:xlabel='fcr', ylabel='average_humidity'>,
        <AxesSubplot:xlabel='average_temperature', ylabel='average_humidity'>,
        <AxesSubplot:xlabel='pressure', ylabel='average_humidity'>,
        <AxesSubplot:xlabel='average_humidity', ylabel='average_humidity'>]],
      dtype=object)
```



[ ]:

APÊNDICE B – Carregamento dos dados para aprendizagem

inobram-platform-load-data

November 21, 2021

1 Carregamento de dados, preparação, análise e exportação para CSV

Este notebook irá focar em preparar os dados para a aprendizagem de máquina

1.1 Consulta ao banco de dados PostgreSQL

A função a seguir será utilizada ao longo deste notebook sempre que novos dados precisarem ser obtidos da base de dados:

[1]:
```
import pandas as pd
import sqlalchemy
import psycopg2

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "inobram-platform-local-21-07-30"
DB_USER = "postgres"
DB_PASSWORD = "postgres"
CON = f"postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/
 ↪{DB_NAME}"

def load_db_data(sql, con=CON):
    return pd.read_sql_query(sql, con)
```

1.2 Definindo quais lotes atendem as características necessárias

Serão considerados lotes com as seguintes características:

- Estado do lote: encerrado;
- A quantidade de aves alojadas tenha sido fornecida;
- A mortalidade diária tenha sido fornecida;
- O peso dos silos de ração tenha sido registrado;

[2]:
```
sql = '''SELECT DISTINCT b.id
         FROM batch_batch AS b
         INNER JOIN reading_reading AS r
             ON r.mac = b.device
         INNER JOIN mortality_mortality AS m
```

```
            ON m.batch_id = b.id
        WHERE b.quantity_housed > 0
        AND b.state = 'closed'
        AND b.id IN (
           SELECT DISTINCT batch_id FROM mortality_mortality
        )
        AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') > b.initial_date
        AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') < b.final_date
        AND r.sps1 > 0
        ORDER BY b.id ASC'''
valid_batches_df = load_db_data(sql)

batches_id_list = valid_batches_df['id'].values.tolist()
```

Na base de dados da data de 30 de julho de 2021, 59 lotes atendem as características desejadas:

[3]:
```
len(batches_id_list)
```

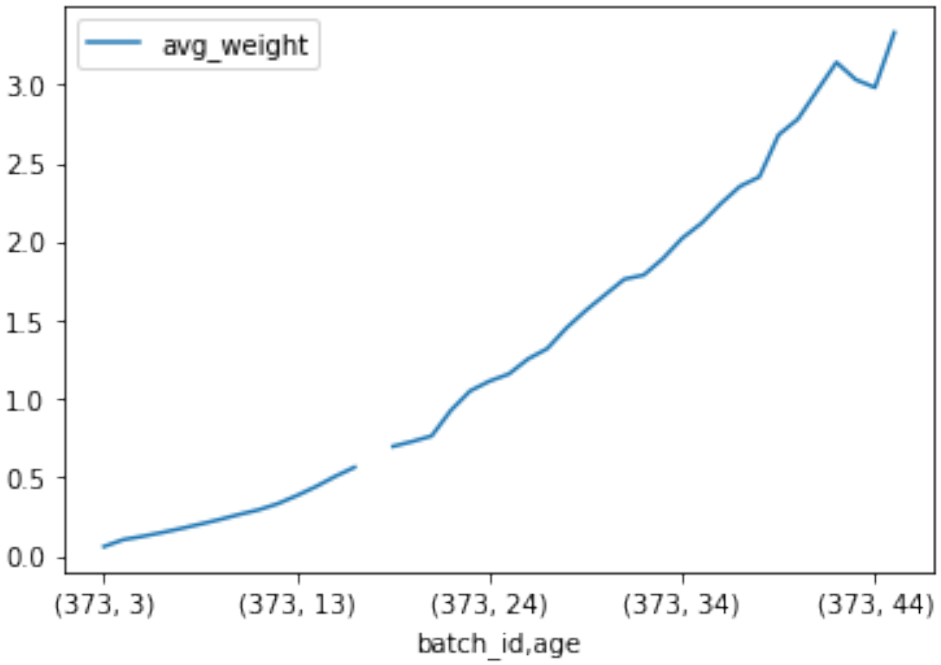
[3]: 59

1.3 Peso médio das aves

O peso médio diário das aves pode ser obtido da tabela de análise de peso. Aqui também está sendo feita a conversão de gramas para Kilogramas

[4]:
```
sql='''SELECT b.id AS batch_id,
            wa.age,
            wa.aw / 1000 AS avg_weight,
            TO_DATE(wa.rec_at, 'YYYY-MM-DD HH24:MI:SS') AS date
    FROM weight_analysis_weightanalysis AS wa
    INNER JOIN batch_batch AS b
    ON b.device = wa.mac
    WHERE b.device = wa.mac
      AND TO_DATE(LEFT(wa.rec_at, 10), 'YYYY-MM-DD') >= b.initial_date
      AND TO_DATE(LEFT(wa.rec_at, 10), 'YYYY-MM-DD') <= b.final_date
      AND b.id IN {}
    ORDER BY batch_id, age;'''.format(tuple(batches_id_list))
avg_weight_df = load_db_data(sql=sql).set_index(['batch_id', 'age'])
```

[5]:
```
avg_weight_df.query("batch_id == 373")["avg_weight"].plot(legend=True).figure.
 ↪savefig("avg-weight.svg")
```



Criando um Dataframe contendo os dados diários, que inicialmente contém apenas os pesos médios:

[6]:
```
daily_stats_df = avg_weight_df
```

1.4 Consumo de ração

A base de dados apresenta as leituras de peso do silo minuto a minuto.

[7]:
```
sql='''SELECT b.id AS batch_id,
              r.age,
              r.sps1 AS silo_weight,
              TO_TIMESTAMP(r.rec_at, 'YYYY-MM-DD HH24:MI:SS') AS timestamp
    FROM reading_reading AS r
    INNER JOIN batch_batch AS b
        ON b.device = r.mac
    WHERE b.device = r.mac
        AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') > b.initial_date
        AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') < b.final_date
        AND b.id IN {}
    ORDER BY batch_id, age, timestamp'''.format(tuple(batches_id_list))
silo_weight_minute_df = load_db_data(sql=sql)
```

Inicialmente, é criado um dataframe contendo as pesagens de silo indexadas por id do lote e idade:

[8]:
```
silo_weight_df = silo_weight_minute_df.set_index(['batch_id', 'age'])
```

É obtida então a diferença de peso em relação a leitura do minuto anterior (linha anterior):

[9]:
```
silo_weight_df['difference'] = silo_weight_df['silo_weight'].diff()
```

Destas diferenças, consideram-se apenas as diferenças negativas, ou seja, quando o peso do silo diminuiu, e desconsideram-se as diferenças positivas, ocorridas durante o abastecimento do silo. Também serão desconsideradas quaisquer variações de peso maiores do que 100Kg/min, já que apenas irão ocorrer na prática caso haja um problema com o silo.

Somarão-se estas diferenças ocorridas durante o mesmo dia, obtendo-se o consumo diário, será utilizado o valor absoluto destas diferenças:

[10]:
```
daily_consumption_df = silo_weight_df['difference'][(silo_weight_df.difference␣
↪< 0) & (silo_weight_df.difference > -100)].groupby(['batch_id', 'age']).
↪sum().abs()
```

Serão adicionados ao Dataframe de dados diários, o consumo acumulado de ração:

[11]:
```
daily_stats_df['acc_consumption'] = daily_consumption_df.groupby('batch_id').
↪cumsum()
```

1.5 Ambiência média do dia

A base de dados apresenta os dados coletados minuto a minuto

[12]:
```
sql='''SELECT b.id AS batch_id,
            r.age,
            r.at AS temperature,
            r.ah AS humidity,
            r.p1 AS pressure,
            r.air_co21 AS co2,
            TO_TIMESTAMP(r.rec_at, 'YYYY-MM-DD HH24:MI:SS') AS rec_at
    FROM reading_reading AS r
    INNER JOIN batch_batch AS b
    ON b.device = r.mac
    WHERE b.device = r.mac
      AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') >= b.initial_date
      AND TO_DATE(LEFT(r.rec_at, 10), 'YYYY-MM-DD') <= b.final_date
      AND b.id IN {}
    ORDER BY batch_id, age;'''.format(tuple(batches_id_list))
avg_ambience_df = load_db_data(sql=sql).set_index(['batch_id', 'age'])
```

É calculada a mediana dos valores de temperatura, umidade e pressão para cada dia, e estes valores são então adicionados ao Dataframe de dados diários:

[13]:
```
daily_stats_df['temperature'] = avg_ambience_df.groupby(['batch_id',␣
↪'age'])['temperature'].median()
daily_stats_df['humidity'] = avg_ambience_df.groupby(['batch_id',␣
↪'age'])['humidity'].median()
```

```
daily_stats_df['pressure'] = avg_ambience_df.groupby(['batch_id',␣
↪'age'])['pressure'].median()
```

1.6 Quantidade de aves vivas

A sequir será calculada a quantidade de aves vivas em cada dia do lote. Isto será feito subtraindo a mortalidade diária da quantidade de aves alojadas no lote.

Obtenção da quantidade de aves alojadas:

[14]:
```
sql='''SELECT id AS batch_id, quantity_housed, initial_date, final_date
       FROM batch_batch
       WHERE id IN {};'''.format(tuple(batches_id_list))
quantity_housed_df = load_db_data(sql=sql).set_index('batch_id')
```

Obtenção da mortalidade:

[15]:
```
sql='''SELECT batch_id, age, quantity AS daily_mortality
       FROM mortality_mortality
       WHERE batch_id IN {}
       ORDER BY batch_id, age;'''.format(tuple(batches_id_list))
mortality_df = load_db_data(sql=sql).set_index(['batch_id', 'age'])
```

Calculo da mortalidade acumulada:

[16]:
```
mortality_df['acc_mortality'] = mortality_df.
↪groupby(['batch_id'])['daily_mortality'].cumsum()
```

Obtenção da quantidade de aves vivas, subtraindo a mortalidade acumulada da quantidade de aves alojadas:

[17]:
```
mortality_df['living_birds'] = quantity_housed_df['quantity_housed'].
↪sub(mortality_df['acc_mortality'])
```

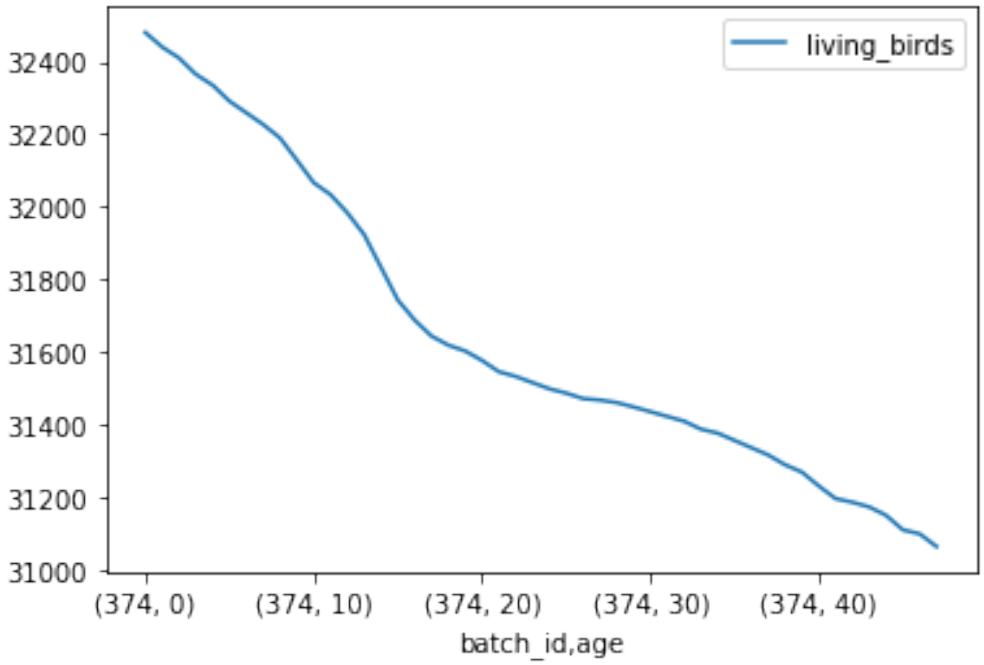
1.6.1 Cálculo do percentual de aves vivas

Saber a proporção de aves vivas em relação a quantidade de aves alojadas é necessário, pois a quantidade de aves alojadas muda de um lote para outro. Dividindo a quantidade de aves vivas pela quantidade de aves alojadas no início do lote teremos esta relação.

[18]:
```
#mortality_df['lb_ratio'] = mortality_df['living_birds'].
↪div(quantity_housed_df['quantity_housed'])
```

[19]:
```
daily_stats_df['living_birds'] = mortality_df['living_birds']
```

[20]:
```
mortality_df.query("batch_id == 374")["living_birds"].plot(legend=True).figure.
↪savefig("living-birds.svg")
```



1.7 Conversão alimentar

Calculada dividindo-se o consumo acumulado de ração pela quantidade de aves vivas, e dividindo pelo peso médio das aves. Desta forma, caso a mortalidade tenha sido alta no lote, a conversão alimentar será maior.

Quanto menor a conversão alimentar, melhor a produtividade do lote.

[21]:
```
daily_stats_df['acc_consumption_per_bird'] = daily_stats_df['acc_consumption'] /
 ↪ daily_stats_df['living_birds']
daily_stats_df['feed_conversion_rate'] =␣
 ↪daily_stats_df['acc_consumption_per_bird'] / daily_stats_df['avg_weight']
```

1.8 Análise exploratória dos dados

Visualizando os registros com menor taxa de conversão alimentar para a idade 42, onde as aves geralmente estão próximas da data de abate:

[74]:
```
daily_stats_df.query('batch_id == 246').head()
```

[74]:

		avg_weight	date	acc_consumption	temperature	humidity
batch_id	age					
246	0	0.055	2021-01-30	NaN	33.3	54.0
	1	0.070	2021-01-31	320.0	31.8	65.0
	2	0.090	2021-02-01	785.0	31.0	76.0

```
          3         0.106  2021-02-02           1430.0         30.6      76.0
          4         0.125  2021-02-03           2065.0         30.3      79.0

                pressure  living_birds  acc_consumption_per_bird  \
batch_id age
246      0          26.0         33980                       NaN
         1           0.0         33780                  0.009473
         2           0.0         33629                  0.023343
         3          18.0         33507                  0.042678
         4          21.0         33321                  0.061973

                feed_conversion_rate
batch_id age
246      0                       NaN
         1                  0.135329
         2                  0.259366
         3                  0.402619
         4                  0.495783
```

[70]:
```
daily_stats_df.query('age == 42').sort_values('date', ascending=True)
```

[70]:

batch_id	age	avg_weight	date	acc_consumption	temperature	humidity
160	42	2.884	2021-01-08	265105.0	23.2	99.0
161	42	2.578	2021-01-08	145970.0	23.1	99.0
162	42	3.087	2021-01-12	160120.0	23.4	99.0
405	42	2.577	2021-01-12	148300.0	23.2	99.0
264	42	2.577	2021-01-12	148300.0	23.2	99.0
164	42	3.266	2021-01-13	157685.0	24.4	97.0
275	42	2.604	2021-01-13	149812.0	24.7	0.0
166	42	3.859	2021-01-14	151745.0	22.5	94.0
168	42	3.456	2021-01-16	171300.0	22.4	99.0
169	42	2.944	2021-01-16	148285.0	22.4	99.0
286	42	2.529	2021-01-19	153113.0	22.9	99.0
270	42	2.529	2021-01-19	153113.0	22.9	99.0
259	42	3.247	2021-01-19	169960.0	22.8	80.0
281	42	3.247	2021-01-19	169960.0	22.8	80.0
178	42	2.659	2021-01-21	136355.0	22.1	99.0
246	42	2.786	2021-03-13	150410.0	22.6	99.0
244	42	3.180	2021-03-13	270970.0	22.6	84.0
265	42	2.804	2021-03-16	152525.0	22.2	99.0
406	42	2.804	2021-03-16	152525.0	22.2	99.0
249	42	3.188	2021-03-17	168985.0	22.2	93.0
276	42	2.755	2021-03-17	158057.0	22.2	18.0
252	42	2.734	2021-03-20	196530.0	23.3	99.0
254	42	3.059	2021-03-21	157145.0	22.2	96.0
287	42	3.067	2021-03-24	171605.0	21.6	88.0

288	42	2.788	2021-03-24	165790.0	21.6	99.0
352	42	2.211	2021-05-14	147745.0	20.6	68.0
351	42	3.012	2021-05-14	297735.0	20.8	91.0
350	42	3.316	2021-05-14	162355.0	21.6	73.0
349	42	2.685	2021-05-14	142030.0	21.6	77.0
407	42	2.685	2021-05-14	142030.0	21.6	77.0
346	42	1.031	2021-05-18	162585.0	21.5	NaN
347	42	2.800	2021-05-18	153865.0	21.5	55.0
342	42	3.155	2021-05-21	170530.0	19.6	86.0
343	42	0.247	2021-05-21	146920.0	20.4	76.0
341	42	2.717	2021-05-26	155960.0	20.4	99.0
340	42	3.122	2021-05-26	158375.0	20.6	83.0
373	42	3.141	2021-07-10	349125.0	20.4	56.0
371	42	3.229	2021-07-15	136910.0	20.6	62.0
369	42	3.028	2021-07-19	163920.0	18.9	48.0
370	42	2.700	2021-07-19	128980.0	16.8	41.0

		pressure	living_birds	acc_consumption_per_bird	\
batch_id	age				
160	42	41.0	32589	8.134800	
161	42	38.0	33020	4.420654	
162	42	42.0	32907	4.865834	
405	42	41.0	33299	4.453587	
264	42	41.0	33299	4.453587	
164	42	43.0	32847	4.800591	
275	42	43.0	33132	4.521671	
166	42	40.0	32020	4.739069	
168	42	51.0	32048	5.345107	
169	42	50.0	33001	4.493349	
286	42	51.0	33258	4.603795	
270	42	51.0	33258	4.603795	
259	42	44.0	32656	5.204557	
281	42	44.0	32656	5.204557	
178	42	41.0	32845	4.151469	
246	42	37.0	32520	4.625154	
244	42	42.0	32048	8.455130	
265	42	46.0	33308	4.579230	
406	42	46.0	33308	4.579230	
249	42	45.0	32700	5.167737	
276	42	44.0	33056	4.781492	
252	42	48.0	33433	5.878324	
254	42	53.0	32992	4.763124	
287	42	47.0	32563	5.269938	
288	42	54.0	33310	4.977184	
352	42	32.0	32131	4.598207	
351	42	31.0	31987	9.308000	
350	42	41.0	31729	5.116928	

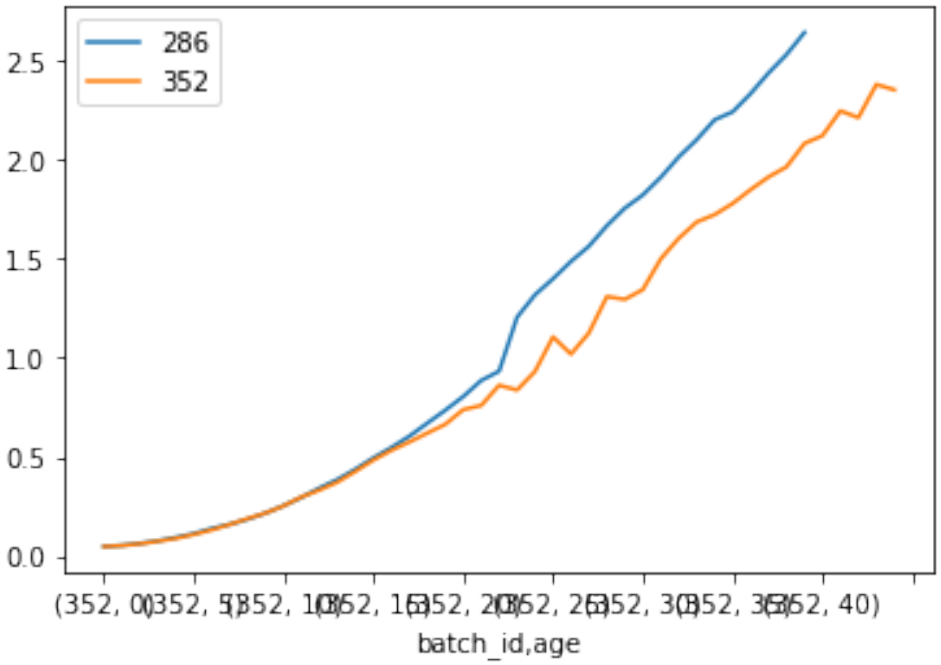
349	42	36.0	31268	4.542344
407	42	36.0	31268	4.542344
346	42	45.0	31874	5.100866
347	42	37.0	32319	4.760822
342	42	52.0	31655	5.387143
343	42	51.0	31850	4.612873
341	42	44.0	32173	4.847543
340	42	37.0	31657	5.002843
373	42	35.0	30924	11.289775
371	42	30.0	31328	4.370212
369	42	28.0	31456	5.211089
370	42	29.0	31698	4.069026

		feed_conversion_rate
batch_id	age	
160	42	2.820666
161	42	1.714761
162	42	1.576234
405	42	1.728206
264	42	1.728206
164	42	1.469869
275	42	1.736433
166	42	1.228056
168	42	1.546617
169	42	1.526273
286	42	1.820401
270	42	1.820401
259	42	1.602882
281	42	1.602882
178	42	1.561290
246	42	1.660141
244	42	2.658846
265	42	1.633106
406	42	1.633106
249	42	1.620997
276	42	1.735569
252	42	2.150082
254	42	1.557085
287	42	1.718271
288	42	1.785217
352	42	2.079696
351	42	3.090305
350	42	1.543102
349	42	1.691748
407	42	1.691748
346	42	4.947494
347	42	1.700294

```
342       42               1.707494
343       42              18.675599
341       42               1.784153
340       42               1.602448
373       42               3.594325
371       42               1.353426
369       42               1.720967
370       42               1.507047
```

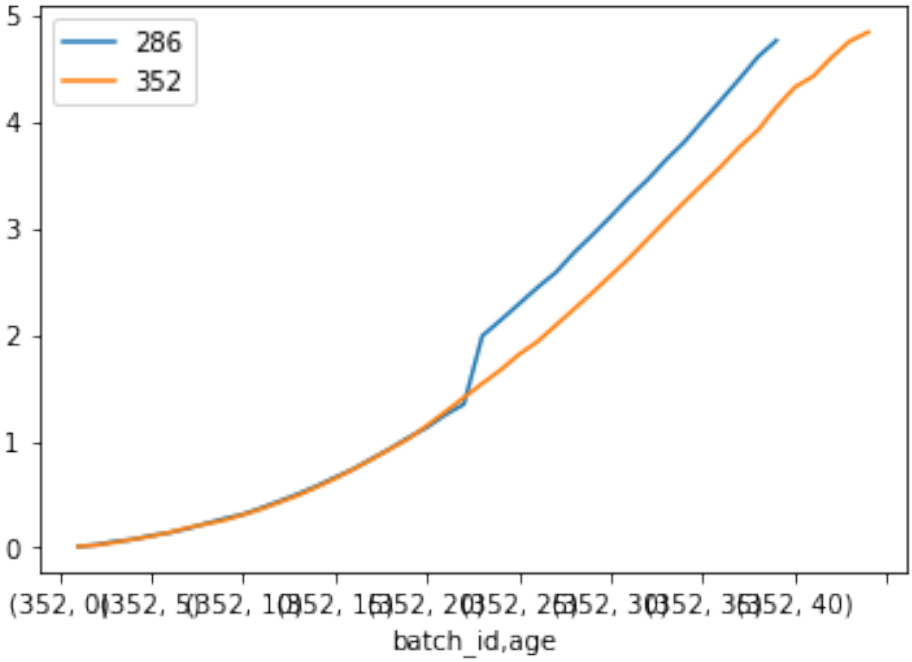
[57]:
```
daily_stats_df.query('batch_id == 286')['avg_weight'].plot(legend=True,␣
↪label=286)
daily_stats_df.query('batch_id == 352')['avg_weight'].plot(legend=True,␣
↪label=352)
```

[57]: <AxesSubplot:xlabel='batch_id,age'>



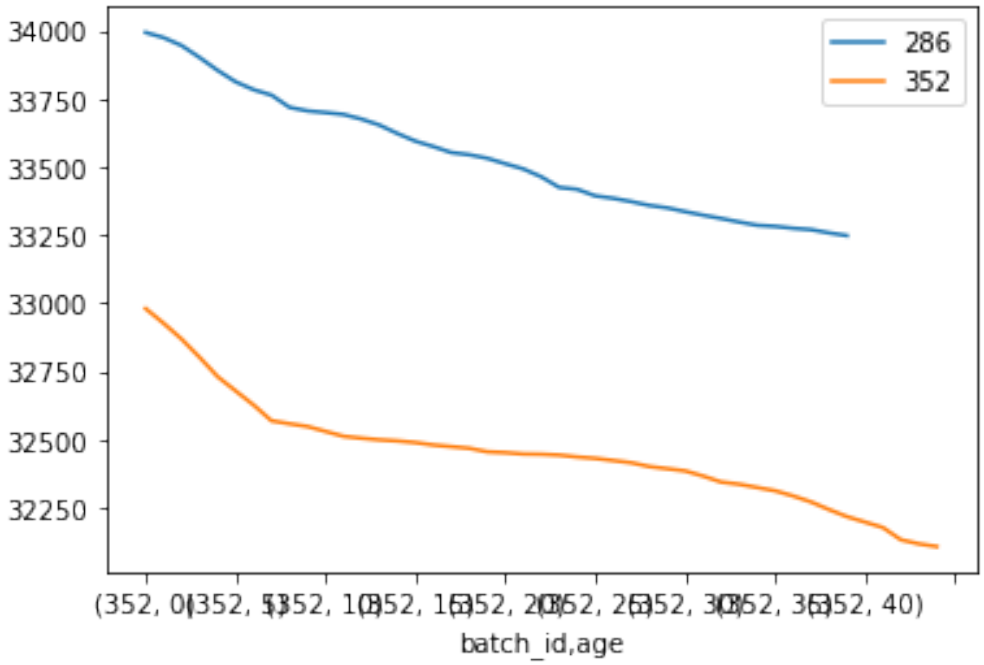
[58]:
```
daily_stats_df.query('batch_id == 286')['acc_consumption_per_bird'].
↪plot(legend=True, label=286)
daily_stats_df.query('batch_id == 352')['acc_consumption_per_bird'].
↪plot(legend=True, label=352)
```

[58]: <AxesSubplot:xlabel='batch_id,age'>



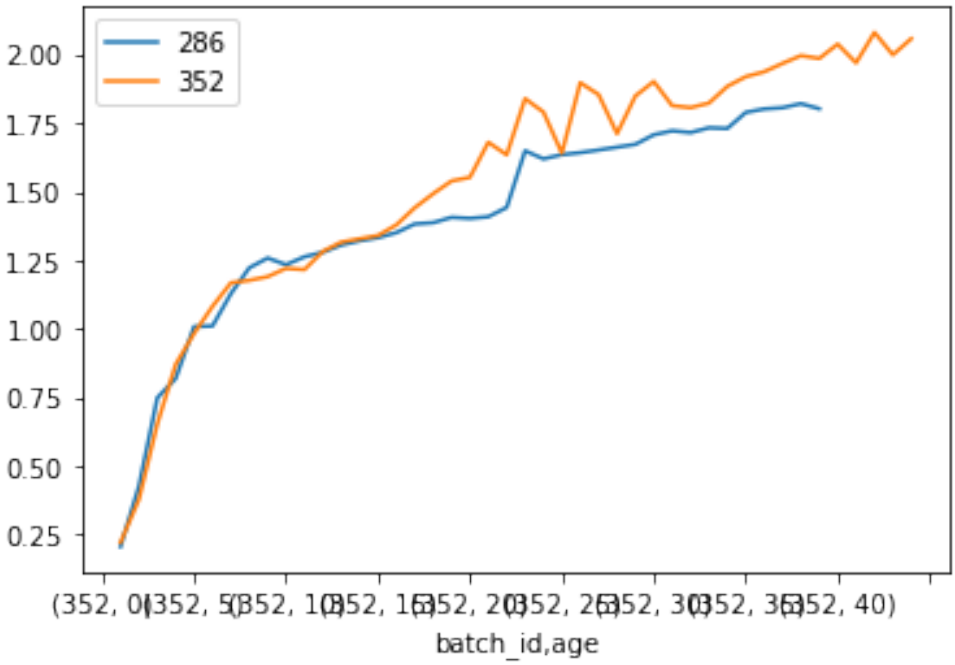
[59]:
```
daily_stats_df.query('batch_id == 286')['living_birds'].plot(legend=True,␣
↪label=286)
daily_stats_df.query('batch_id == 352')['living_birds'].plot(legend=True,␣
↪label=352)
```

[59]: <AxesSubplot:xlabel='batch_id,age'>



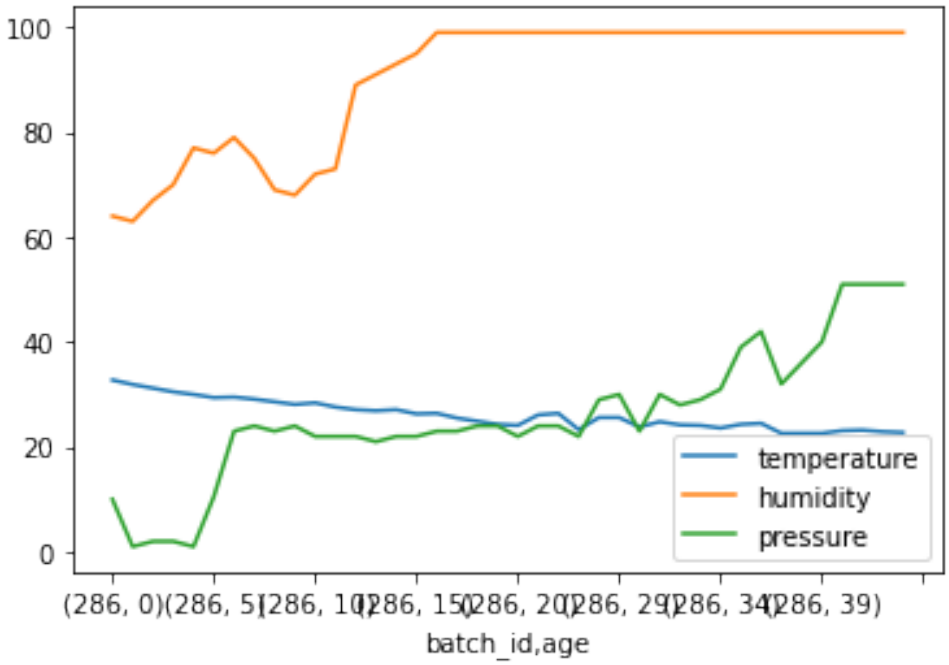
[60]:
```
daily_stats_df.query('batch_id == 286')['feed_conversion_rate'].
 ↪plot(legend=True, label=286)
daily_stats_df.query('batch_id == 352')['feed_conversion_rate'].
 ↪plot(legend=True, label=352)
```

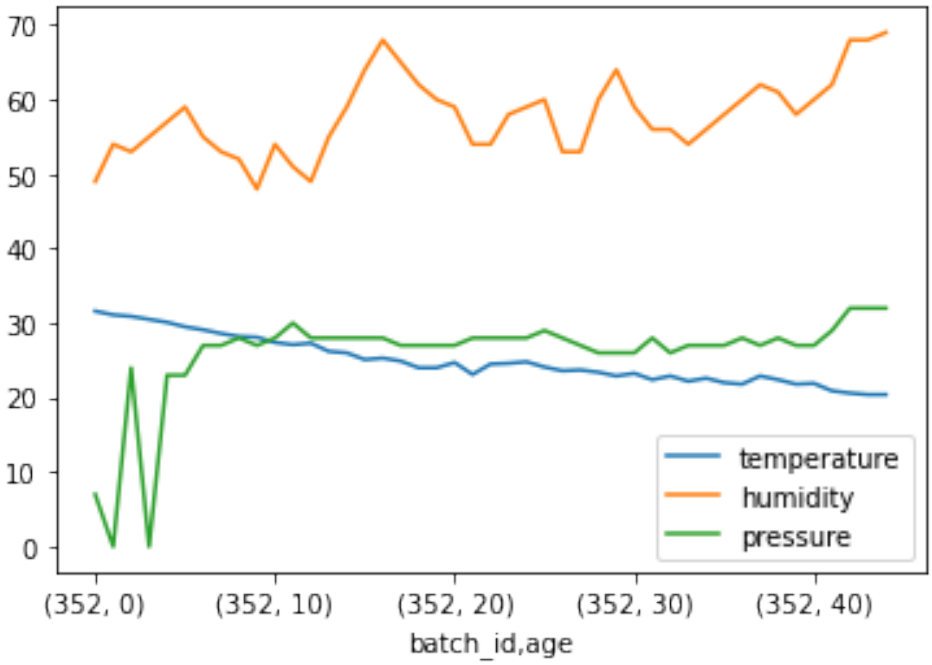
[60]: <AxesSubplot:xlabel='batch_id,age'>



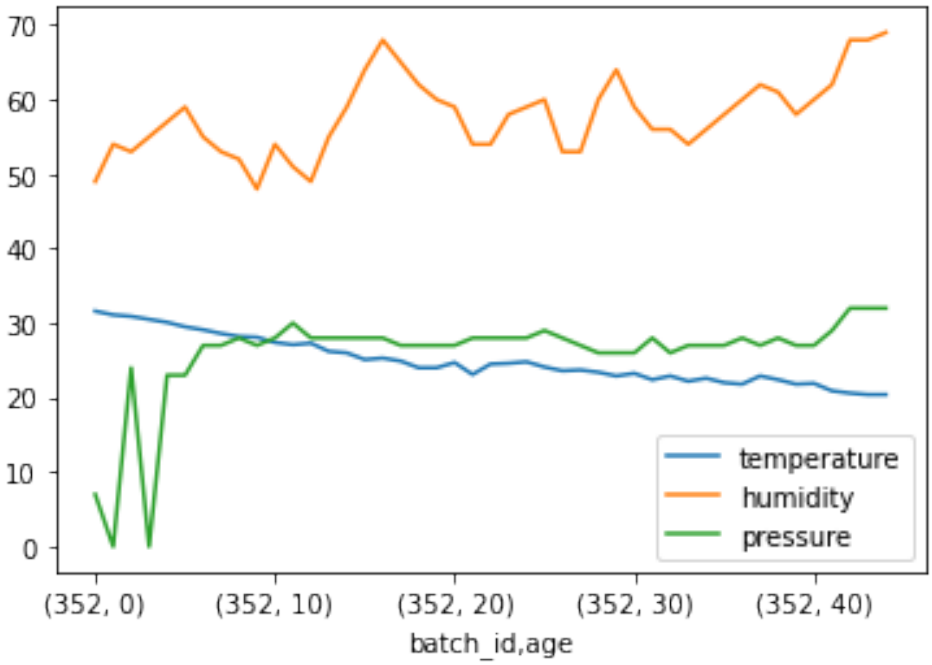
[61]:
```
daily_stats_df.query('batch_id == 286')[['temperature', 'humidity',␣
↪'pressure']].plot(legend=True, label=286)
daily_stats_df.query('batch_id == 352')[['temperature', 'humidity',␣
↪'pressure']].plot(legend=True, label=352)
```

[61]:
```
<AxesSubplot:xlabel='batch_id,age'>
```





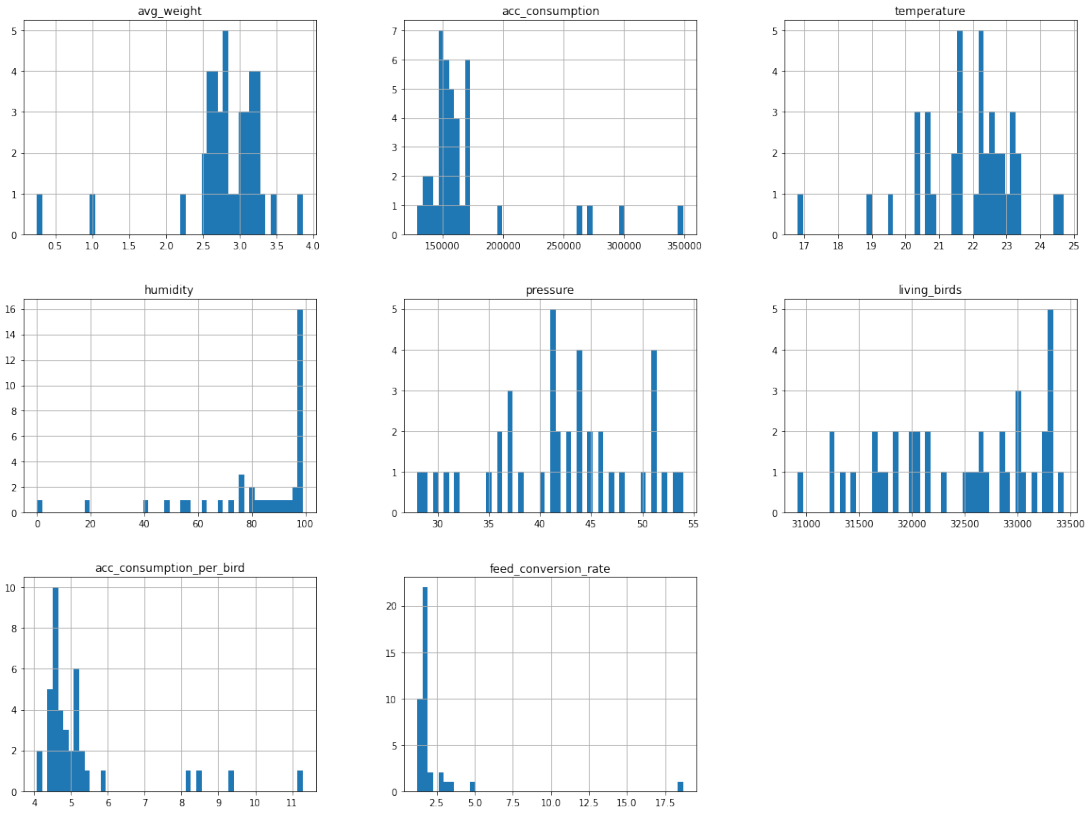
[73]:
```
daily_stats_df.query('batch_id == 352')[['temperature', 'humidity',␣
↪'pressure']].plot(legend=True).figure.savefig('ambiencia.svg')
```



Visualizando histogramas dos lotes na idade 42

[23]:
```
daily_stats_df.query('age == 42').hist(bins=50, figsize=(20,15))
```

[23]:
```
array([[<AxesSubplot:title={'center':'avg_weight'}>,
        <AxesSubplot:title={'center':'acc_consumption'}>,
        <AxesSubplot:title={'center':'temperature'}>],
       [<AxesSubplot:title={'center':'humidity'}>,
        <AxesSubplot:title={'center':'pressure'}>,
        <AxesSubplot:title={'center':'living_birds'}>],
       [<AxesSubplot:title={'center':'acc_consumption_per_bird'}>,
        <AxesSubplot:title={'center':'feed_conversion_rate'}>,
        <AxesSubplot:>]], dtype=object)
```



1.9 Exportando os dados para um arquivo CSV

[24]:
```
daily_stats_df.to_csv('inobram-poultry-daily.csv', index=True)
```

[25]:
```
daily_stats_df = pd.read_csv('inobram-poultry-daily.csv').
↪set_index(['batch_id', 'age'])
```

APÊNDICE C – Aprendizagem de Máquina

inobram-platform-ml-keras

November 21, 2021

1 LSTM com keras utilizando janela móvel

1.1 Instalando bibliotecas necessárias para aprendizagem

[762]:
```
#!pip install tensorflow
```

[763]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

from joblib import dump, load
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
from pylab import rcParams
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from tensorflow import keras

from copy import deepcopy

%matplotlib inline
%config InLineBackend.figure_format='retina'

register_matplotlib_converters()
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 22, 10

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
```

1.2 Carregando os dados do arquivo CSV

Carregando os dados:

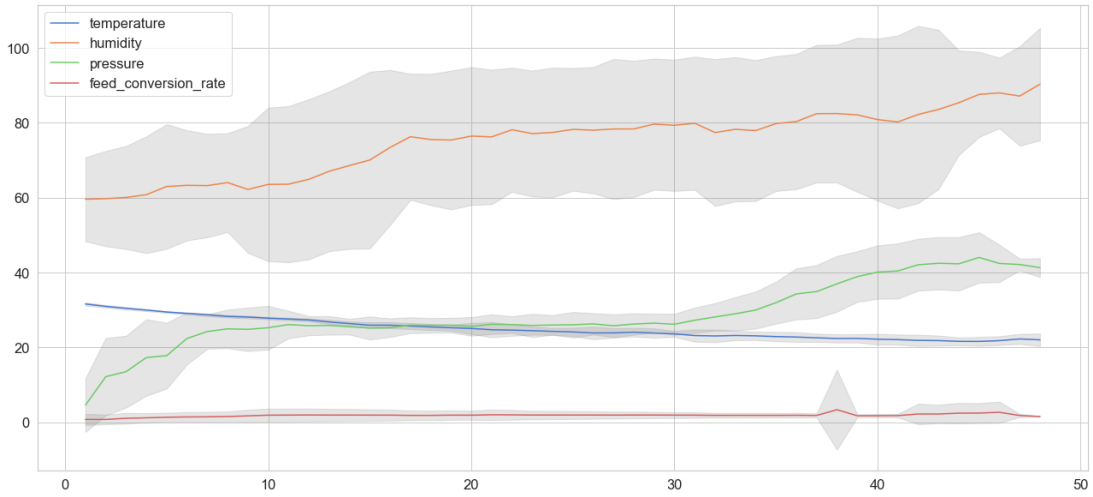
[764]:
```
daily_stats_df = pd.read_csv('inobram-poultry-daily.csv')
daily_stats_df['age_col'] = daily_stats_df['age']
daily_stats_df.set_index(['batch_id', 'age'], inplace=True)
```

[765]:
```
daily_stats_df.drop('date', axis='columns', inplace=True)
daily_stats_df.drop('acc_consumption', axis='columns', inplace=True)
daily_stats_df.drop('living_birds', axis='columns', inplace=True)
daily_stats_df.dropna(inplace=True)
```

[766]:
```
daily_stats_mean = daily_stats_df.groupby('age_col').mean()
daily_stats_sd = daily_stats_df.groupby('age_col').std()
```

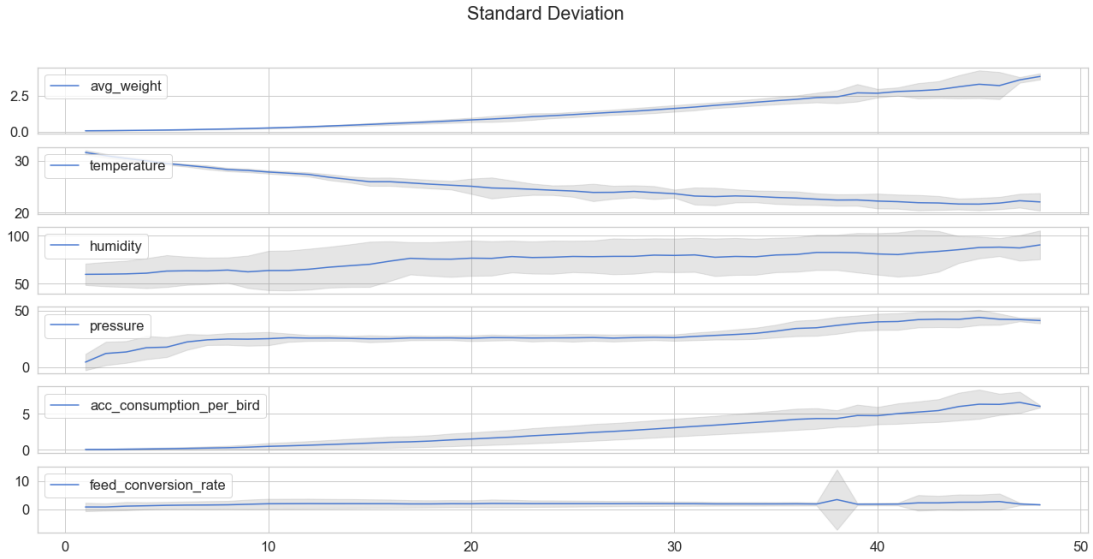
[767]:
```
daily_stats_l = daily_stats_mean.sub(daily_stats_sd)
daily_stats_h = daily_stats_mean.add(daily_stats_sd)
```

[768]:
```
for column in ['temperature', 'humidity', 'pressure', 'feed_conversion_rate']:␣
↪#daily_stats_mean.columns:
    plt.plot(daily_stats_mean[column], '-', label=column)
    plt.fill_between(daily_stats_l.index.to_numpy(), daily_stats_l[column].
↪to_numpy(), daily_stats_h[column].to_numpy(), color='gray', alpha=0.2)
plt.legend(loc='upper left')
plt.savefig('daily-stats-mean.svg')
```



[769]:
```
fig, axs = plt.subplots(ncols=1, nrows=len(daily_stats_df.columns)-1,␣
↪sharex=True)
fig.suptitle('Standard Deviation')

columns = daily_stats_mean.columns.tolist()
```

```
for column in columns:
    axs[columns.index(column)].plot(daily_stats_mean[column], '-', label=column)
    axs[columns.index(column)].legend(loc="upper left")
    axs[columns.index(column)].fill_between(daily_stats_l.index.to_numpy(),␣
 ↪daily_stats_l[column].to_numpy(), daily_stats_h[column].to_numpy(),␣
 ↪color='gray', alpha=0.2)
```



1.3 Separando em conjuntos de treino e teste

Os conjuntos serão separados baseados na id do lote, ou seja, ao invés de separar uma série temporal em blocos, algumas séries serão utilizadas para treino e outras para teste:

[770]:
```
train_ids, test_ids = train_test_split(daily_stats_df.index.levels[0],␣
 ↪test_size=.20, random_state = 42)
```

[771]:
```
train = daily_stats_df.loc[train_ids]
test = daily_stats_df.loc[test_ids]
```

[772]:
```
bad_samples = train.query('feed_conversion_rate >= 2.4')['feed_conversion_rate']
```

[773]:
```
bad_training_ids = bad_samples.index.get_level_values(0).tolist()
```

[774]:
```
train_ids = set(train_ids) - set(bad_training_ids)
```

[775]:
```
print('Train batches: {train}, test: {test}'.format(train=len(train_ids),␣
 ↪test=len(test_ids)))
```

```
Train batches: 34, test: 12
```

[776]:
```
train = daily_stats_df.loc[train_ids]
```

1.4 Parâmetros

[777]:
```
F_COLUMNS = ['temperature', 'pressure', 'humidity', 'age_col']
O_COLUMNS = ['feed_conversion_rate']
TIME_STEPS = 2
UNITS = 170
DROPOUT_RATE = 0.1
EPOCHS = 50
BATCH_SIZE = 2000
USE_BIAS=True
```

1.5 Normalização

[778]:
```
f_transformer = RobustScaler()
o_transformer = RobustScaler()

f_transformer = f_transformer.fit(train.loc[:, F_COLUMNS].to_numpy())
o_transformer = o_transformer.fit(train.loc[:, O_COLUMNS].to_numpy())

train_normalized = deepcopy(train)
test_normalized = deepcopy(test)

train_normalized.loc[:, F_COLUMNS] = f_transformer.transform(train.loc[:,␣
↪F_COLUMNS].to_numpy())
train_normalized.loc[:, O_COLUMNS] = o_transformer.transform(train.loc[:,␣
↪O_COLUMNS].to_numpy())

test_normalized.loc[:, F_COLUMNS] = f_transformer.transform(test.loc[:,␣
↪F_COLUMNS].to_numpy())
test_normalized.loc[:, O_COLUMNS] = o_transformer.transform(test.loc[:,␣
↪O_COLUMNS].to_numpy())
```

1.6 Criando pedaços das séries temporais

Também conhecidos como *chunks*

Obs: Possivelmente será necessário criar os *chunks* iterando sobre os lotes e dias de lote, para manter a ordem temporal e não misturar na mesma série dados de lotes diferentes.

[779]:
```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for batch_id in X.index.levels[0].tolist():
        try:
            X_batch = X.loc[batch_id, :]
            y_batch = y.loc[batch_id, :]
```

```
            for i in range(len(X_batch) - time_steps):
                values = X_batch.iloc[i: (i + time_steps)].to_numpy()
                Xs.append(values)
                ys.append(y_batch.iloc[i + time_steps])
        except:
            pass
    return np.array(Xs), np.array(ys)
```

[780]:
```
X_train, y_train = create_dataset(train_normalized.loc[:, F_COLUMNS],␣
↪train_normalized.loc[:, O_COLUMNS], time_steps=TIME_STEPS)
X_test, y_test = create_dataset(test_normalized.loc[:, F_COLUMNS],␣
↪test_normalized.loc[:, O_COLUMNS], time_steps=TIME_STEPS)
```

[781]:
```
# [amostras, time_steps, características]
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(1198, 2, 4) (1198, 1)
(431, 2, 4) (431, 1)
```

1.7 Modelo LSTM no Keras

[782]:
```
model = keras.Sequential()
model.add(
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            units=UNITS,
            use_bias=USE_BIAS,
            input_shape=(X_train.shape[1], X_train.shape[2])
        )
    )
)
model.add(keras.layers.Dropout(rate=DROPOUT_RATE))
model.add(keras.layers.Dense(units=len(O_COLUMNS)))
```

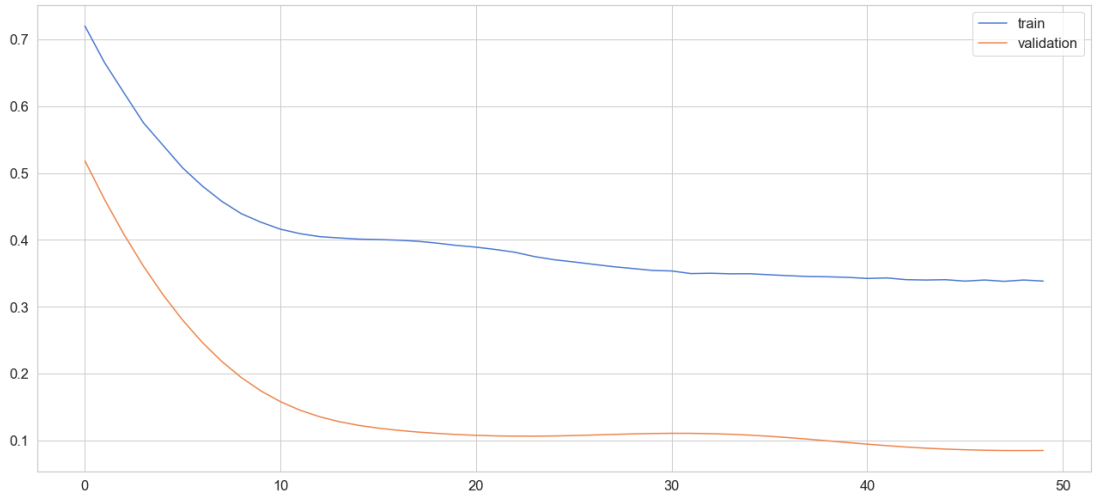
[783]:
```
model.compile(loss='mean_squared_error', optimizer='adam')
```

[ ]:
```
history = model.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1,
    shuffle=False
)
```

[785]:
```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
```

```
plt.legend()
```

[785]: <matplotlib.legend.Legend at 0x162a95970>

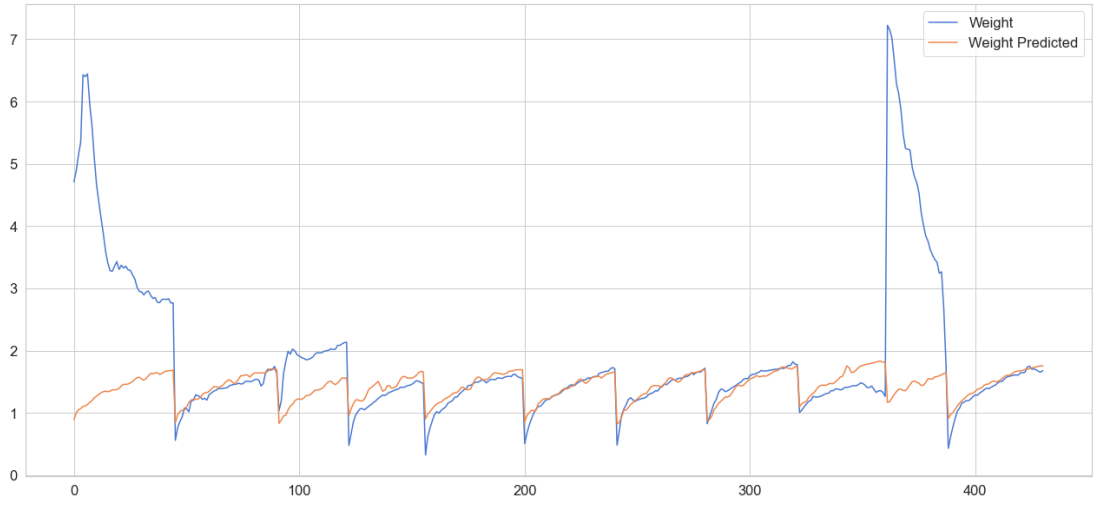


[786]:
```
y_pred = model.predict(X_test)
```

[787]:
```
y_train_inv = o_transformer.inverse_transform(y_train)
y_test_inv = o_transformer.inverse_transform(y_test)
y_pred_inv = o_transformer.inverse_transform(y_pred)
```

[788]:
```
plt.plot(y_test_inv[:, 0].flatten(), label= 'Weight')
plt.plot(y_pred_inv[:, 0].flatten(), label= 'Weight Predicted')
plt.legend()
```

[788]: <matplotlib.legend.Legend at 0x1659f9c40>



1.8 Selecionar um único lote de teste para verificar as previsões

Aqui será selecionado um único lote do conjunto de teste para verificar o comportamento das previsões

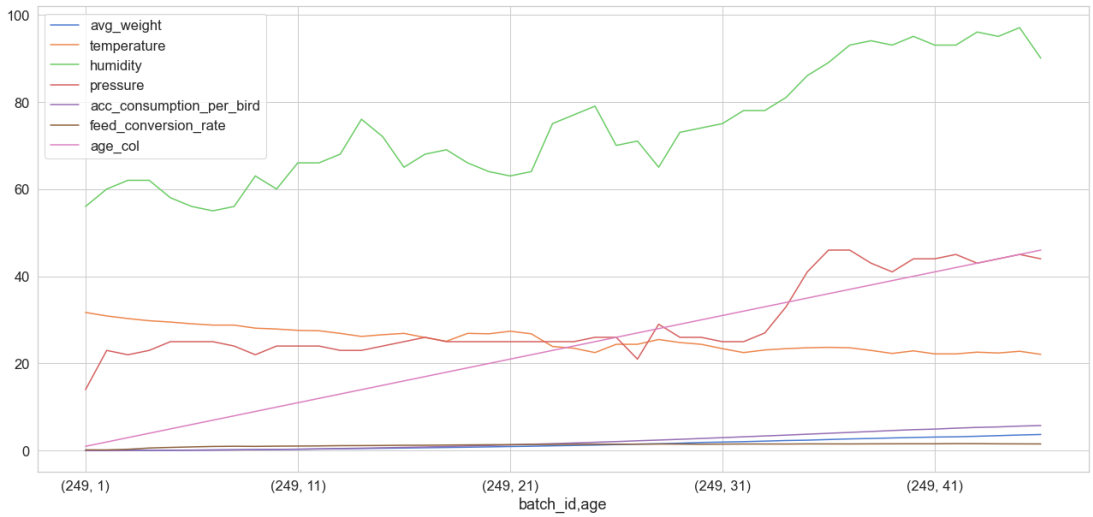
[789]:
```
test_ids
```

[789]:
```
Int64Index([160, 167, 341, 249, 264, 392, 346, 287, 371, 247, 407, 164],
dtype='int64', name='batch_id')
```

[790]:
```
test_batch = daily_stats_df.query('batch_id == 249')
```

[791]:
```
test_batch.plot()
```

[791]:
```
<AxesSubplot:xlabel='batch_id,age'>
```



[792]:
```
test_batch_normalized = deepcopy(test_batch)

test_batch_normalized.loc[:, F_COLUMNS] = f_transformer.transform(test_batch.
 ↪loc[:, F_COLUMNS].to_numpy())
test_batch_normalized.loc[:, O_COLUMNS] = o_transformer.transform(test_batch.
 ↪loc[:, O_COLUMNS].to_numpy())
```

[793]:
```
print(test_batch_normalized.shape)
```

```
(46, 7)
```

[794]:
```
X_test_batch, y_test_batch = create_dataset(test_batch_normalized.loc[:,␣
↪F_COLUMNS], test_batch_normalized.loc[:, O_COLUMNS], time_steps=TIME_STEPS)
```

[795]:
```
X_test_batch.shape
```

[795]:
```
(44, 2, 4)
```

[796]:
```
y_test_batch.shape
```

[796]:
```
(44, 1)
```

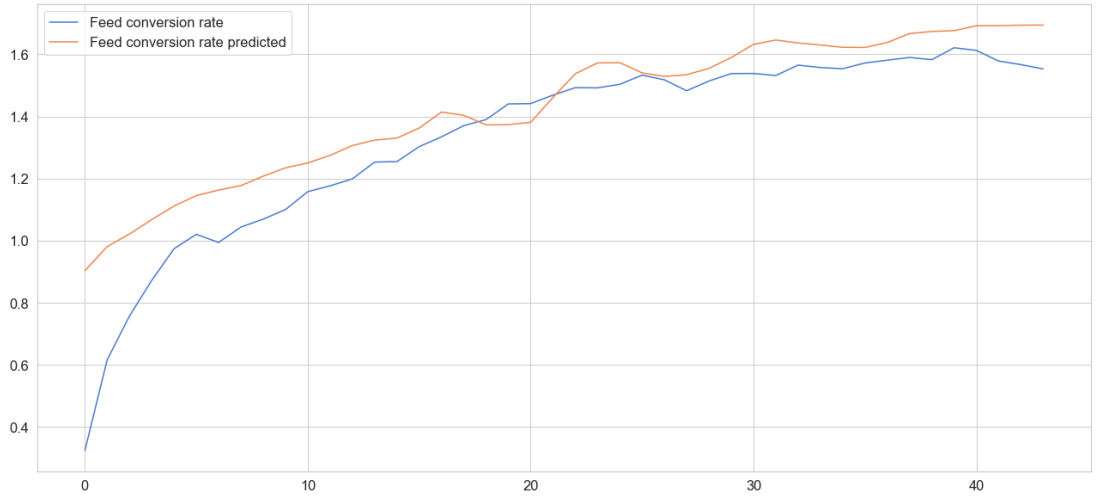
[797]:
```
y_test_batch_pred = model.predict(X_test_batch)
```

[798]:
```
y_test_batch_denormalized = o_transformer.inverse_transform(y_test_batch)
y_test_batch_pred_denormalized = o_transformer.
↪inverse_transform(y_test_batch_pred)
```

[799]:
```
y_test_batch_pred_denormalized[38]
```

[799]:
```
array([1.6734941], dtype=float32)
```

[800]:
```
plt.plot(y_test_batch_denormalized.flatten(), label= 'Feed conversion rate')
plt.plot(y_test_batch_pred_denormalized.flatten(), label= 'Feed conversion rate␣
↪predicted')
plt.legend()
plt.savefig('prediction.svg')
```



[801]:
```
plt.savefig('predicted-vs-real.png')
```

```
<Figure size 1584x720 with 0 Axes>
```

1.9 Salvando o modelo

Possibilita que em produção apenas seja necessário carregar o modelo salvo e fazer as previsões, sem ser necessário executar o treinamento novamente.

[802]:
```
model.save('./keras_model_fcr')
```

```
WARNING:absl:Found untraced functions such as
lstm_cell_387_layer_call_and_return_conditional_losses,
lstm_cell_387_layer_call_fn,
lstm_cell_388_layer_call_and_return_conditional_losses,
lstm_cell_388_layer_call_fn, lstm_cell_387_layer_call_fn while saving (showing 5
of 10). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: ./keras_model_fcr/assets

INFO:tensorflow:Assets written to: ./keras_model_fcr/assets
```

[803]:
```
dump(f_transformer, 'f_transformer.joblib')
dump(o_transformer, 'o_transformer.joblib')
```

[803]:
```
['o_transformer.joblib']
```

2 Modelo alternativo: Múltiplas camadas LSTM

2.1 Parâmetros

[1004]:
```
F_COLUMNS = ['temperature', 'pressure', 'humidity', 'age_col']
O_COLUMNS = ['feed_conversion_rate']
TIME_STEPS = 2
UNITS = 8
DROPOUT_RATE = 0.095
EPOCHS = 488
BATCH_SIZE = 2000
USE_BIAS=True
```

2.2 Modelo LSTM no Keras com múltiplas camadas LSTM

[1005]:
```
initializer = keras.initializers.HeNormal()
model = keras.Sequential()
model.add(
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            units=UNITS,
            use_bias=USE_BIAS,
            activation='softsign',
```

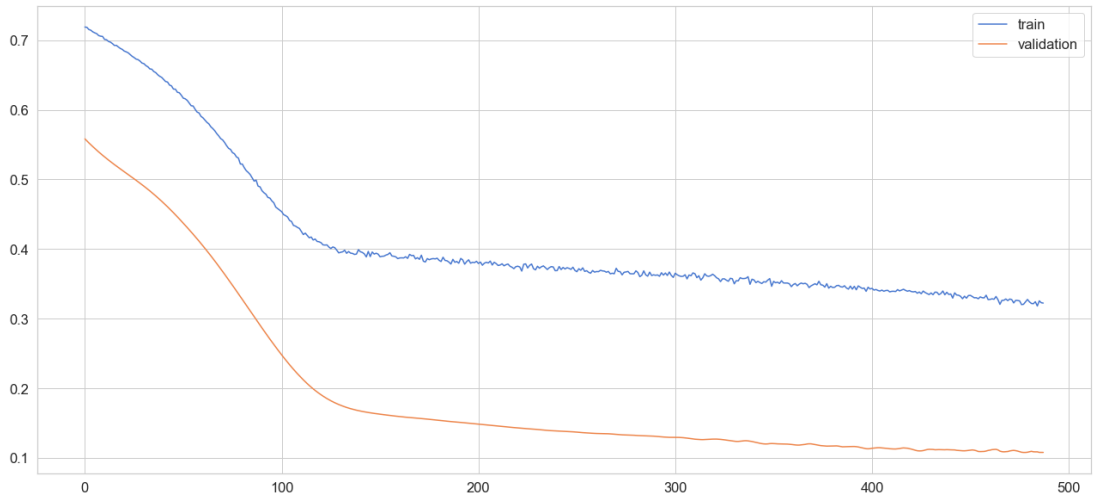
```
            recurrent_activation='sigmoid',
            kernel_initializer=initializer,
            input_shape=(X_train.shape[1], X_train.shape[2]),
            return_sequences=True
        )
    )
)
model.add(
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            units=UNITS,
            use_bias=USE_BIAS,
            activation='softsign',
            recurrent_activation='sigmoid',
            kernel_initializer=initializer,
            input_shape=(X_train.shape[1], X_train.shape[2]),
            return_sequences=True
        )
    )
)
model.add(
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            units=UNITS,
            use_bias=USE_BIAS,
            activation='softsign',
            recurrent_activation='sigmoid',
            kernel_initializer=initializer,
            input_shape=(X_train.shape[1], X_train.shape[2])
        )
    )
)
model.add(keras.layers.Dropout(rate=DROPOUT_RATE))
model.add(keras.layers.Dense(units=len(O_COLUMNS), kernel_regularizer=keras.
 ↪regularizers.l1_l2(l1=0.003, l2=0.003)))
```

[1006]:
```
model.compile(loss='mean_squared_error', optimizer='adam')
```

[ ]:
```
history = model.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1,
    shuffle=False
)
```

[1008]:
```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
plt.legend()
```

[1008]: <matplotlib.legend.Legend at 0x165fc4400>

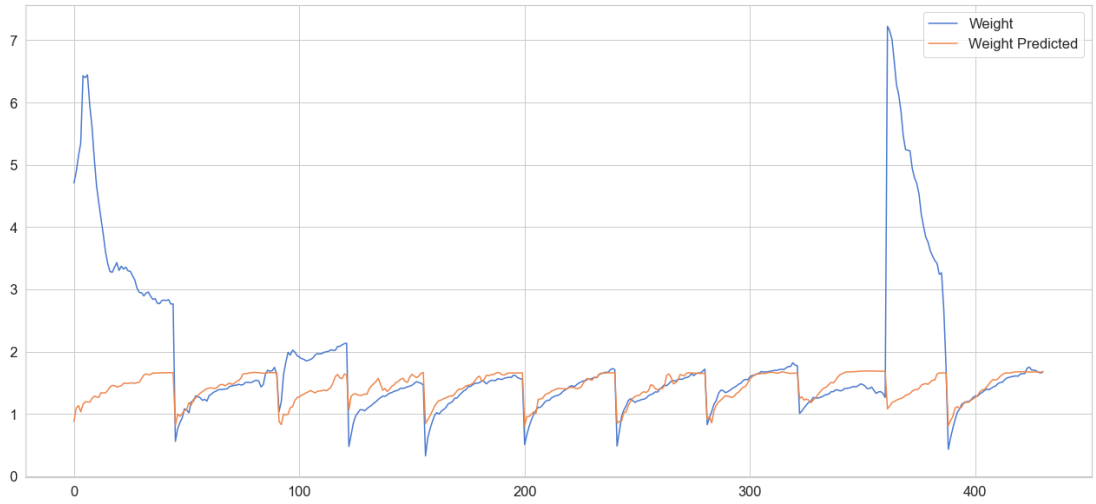


[1009]:
```
y_pred = model.predict(X_test)
```

[1010]:
```
y_train_inv = o_transformer.inverse_transform(y_train)
y_test_inv = o_transformer.inverse_transform(y_test)
y_pred_inv = o_transformer.inverse_transform(y_pred)
```

[1011]:
```
plt.plot(y_test_inv[:, 0].flatten(), label= 'Weight')
plt.plot(y_pred_inv[:, 0].flatten(), label= 'Weight Predicted')
plt.legend()
```

[1011]: <matplotlib.legend.Legend at 0x16c506640>



2.3 Selecionar um único lote de teste para verificar as previsões

Aqui será selecionado um único lote do conjunto de teste para verificar o comportamento das previsões

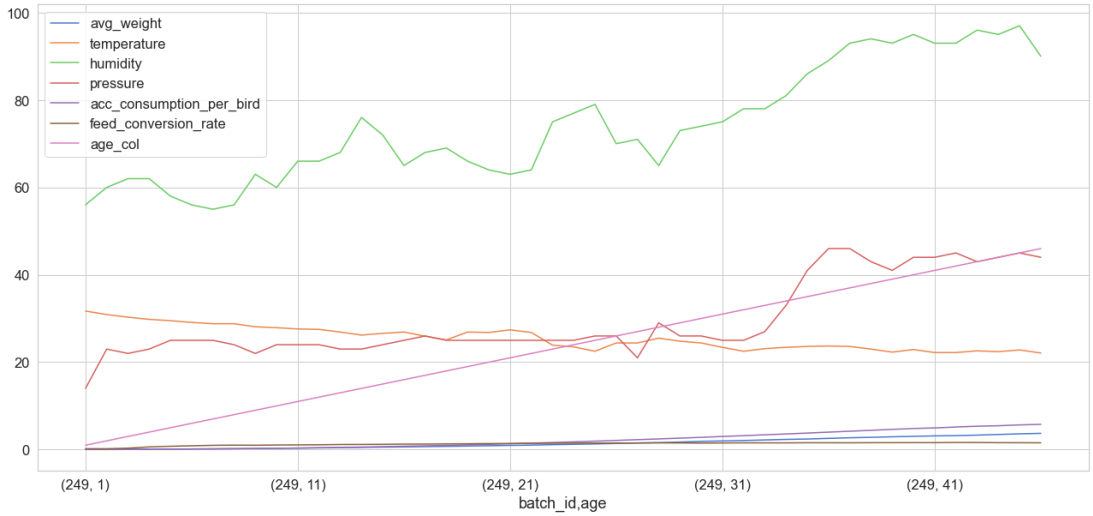
[1012]:
```
test_ids
```

[1012]:
```
Int64Index([160, 167, 341, 249, 264, 392, 346, 287, 371, 247, 407, 164],
dtype='int64', name='batch_id')
```

[1013]:
```
test_batch = daily_stats_df.query('batch_id == 249')
```

[1014]:
```
test_batch.plot()
```

[1014]:
```
<AxesSubplot:xlabel='batch_id,age'>
```



[1015]:
```
test_batch_normalized = deepcopy(test_batch)

test_batch_normalized.loc[:, F_COLUMNS] = f_transformer.transform(test_batch.
 ↪loc[:, F_COLUMNS].to_numpy())
test_batch_normalized.loc[:, O_COLUMNS] = o_transformer.transform(test_batch.
 ↪loc[:, O_COLUMNS].to_numpy())
```

[1016]:
```
print(test_batch_normalized.shape)
```

```
(46, 7)
```

[1017]:
```
X_test_batch, y_test_batch = create_dataset(test_batch_normalized.loc[:,␣
 ↪F_COLUMNS], test_batch_normalized.loc[:, O_COLUMNS], time_steps=TIME_STEPS)
```

[1018]:
```
X_test_batch.shape
```

[1018]: (44, 2, 4)

[1019]:
```
y_test_batch.shape
```

[1019]: (44, 1)

[1020]:
```
y_test_batch_pred = model.predict(X_test_batch)
```

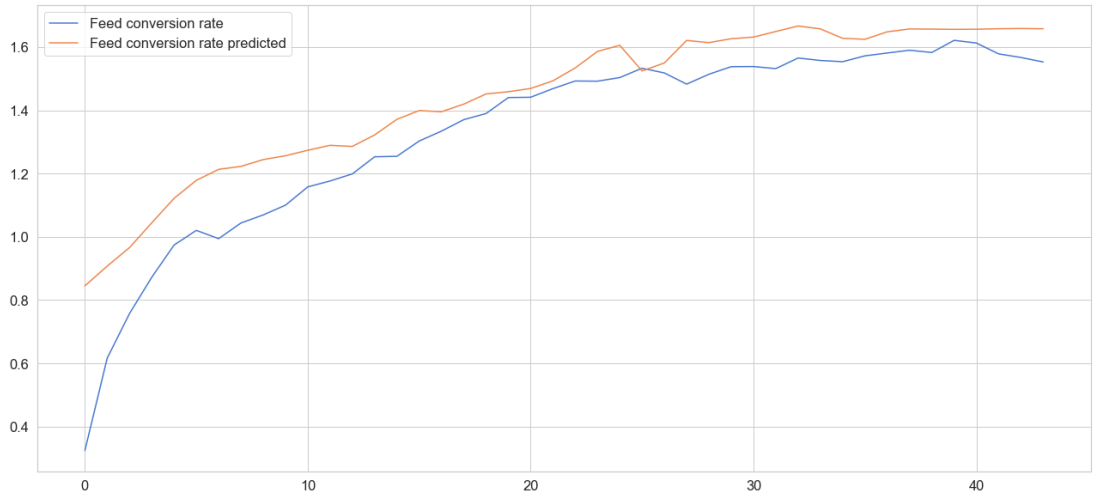
[1021]:
```
y_test_batch_denormalized = o_transformer.inverse_transform(y_test_batch)
y_test_batch_pred_denormalized = o_transformer.
 ↪inverse_transform(y_test_batch_pred)
```

[1022]:
```
y_test_batch_pred_denormalized[38]
```

[1022]:
```
array([1.656528], dtype=float32)
```

[1023]:
```
plt.plot(y_test_batch_denormalized.flatten(), label= 'Feed conversion rate')
plt.plot(y_test_batch_pred_denormalized.flatten(), label= 'Feed conversion rate␣
 ↪predicted')
plt.legend()
```

[1023]:
```
<matplotlib.legend.Legend at 0x16c417ca0>
```



APÊNDICE D – Algoritmo Genético - Busca e Otimização

inobram-platform-ga

November 26, 2021

1 Algoritmo Genético

Aqui o modelo criado no notebook 'inobram-platform-ml-keras' será carregado, e utilizado para fazer previsões dentro da etapa de fit do algoritmo genético

O algoritmo genético será empregado para encontrar candidados a planos de ação capazes de fornecer as menores taxas de conversão alimentar

[1]:
```
from copy import deepcopy
from joblib import dump, load
from operator import itemgetter
from pandas.plotting import register_matplotlib_converters
from pylab import rcParams
from tensorflow import keras

import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random

%matplotlib inline
%config InLineBackend.figure_format='retina'
register_matplotlib_converters()
rcParams['figure.figsize'] = 22, 10

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
```

```
2021-11-26 08:30:34.802801: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory
2021-11-26 08:30:34.802862: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dlerror if you do not have a GPU set up on your machine.
```

1.1 Carregando o modelo de aprendizagem e os escalonadores

Isto torna possível executar a previsão sem precisar retreinar o modelo. Além disto, os escalonadores empregados durante o treinamento do modelo serão utilizados para deixar os dados gerados no algoritmo genético nas escalas adequadas

[2]:
```
model = keras.models.load_model('./keras_model_fcr')
f_transformer = load('./f_transformer.joblib')
o_transformer = load('./o_transformer.joblib')
```

```
2021-11-26 08:30:37.170727: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object
file: No such file or directory
2021-11-26 08:30:37.170744: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit:
UNKNOWN ERROR (303)
2021-11-26 08:30:37.170762: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not
appear to be running on this host (geanceretta-inobram):
/proc/driver/nvidia/version does not exist
2021-11-26 08:30:37.171076: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
/home/gean/projects/tcc/.venv/lib64/python3.9/site-packages/sklearn/base.py:324:
UserWarning: Trying to unpickle estimator RobustScaler from version 0.22.2 when
using version 1.0.1. This might lead to breaking code or invalid results. Use at
your own risk. For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-
maintainability-limitations
  warnings.warn(
```

1.2 Parâmetros

Aqui é necessário utilizar os mesmos parâmetros empregados durante o treinamento.

[3]:
```
F_COLUMNS = ['temperature', 'pressure', 'humidity', 'age_col']
O_COLUMNS = ['feed_conversion_rate']
TIME_STEPS = 2
GENERATIONS = 200
NOT_EVOLVING_TIMEOUT = 10
AGE = 40
SCORE_PENALTY = 0.9

# Based on Cobb:
MAX_TEMPERATURE = 34
```

```
MIN_TEMPERATURE = 14
MAX_HUMIDITY = 70
MIN_HUMIDITY = 30
```

1.3 Criando janelas de temporais

[4]:
```
def create_features_dataset(X, time_steps=1):
    Xs = []
    for batch_id in X.index.levels[0].tolist():
        try:
            X_batch = X.loc[batch_id, :]
            for i in range(len(X_batch) - time_steps):
                values = X_batch.iloc[i: (i + time_steps)].to_numpy()
                Xs.append(values)
        except:
            pass # index not in use in this slice
    return np.array(Xs)
```

1.4 Função que pretendemos otimizar

Nosso algoritmo genético irá tentar otimizar entradas para uma função que retorna a previsão da conversão alimentar para a idade 40. A função recebe como entrada um plano de ação: Temperatura, Umidade, Pressão e Idade do lote:

[5]:
```
def predict_fcr(action_plan_df):
    ap = deepcopy(action_plan_df)
    ap.loc[:, F_COLUMNS] = f_transformer.transform(ap.loc[:, F_COLUMNS].
 ↪to_numpy())
    X = create_features_dataset(ap.loc[:, F_COLUMNS], time_steps=TIME_STEPS)
    y = model.predict(X)

    empty_fcr_period = []
    for empty_fcr in range(TIME_STEPS):
        empty_fcr_period.append([None])

    y = empty_fcr_period + o_transformer.inverse_transform(y).tolist()
    y = [item for sublist in y for item in sublist]
    return y
```

1.5 Função de avaliação da solução

Função que irá avaliar os resultados e atribuir um peso para cada um deles. Neste caso, quanto melhor for o plano de ação para a idade de interesse, maior será o retorno da função fitness().

1.5.1 Temperatura adequada

Literatura da Cobb indica que até que a temperatura possa ser recomendada por idade até que densidade atinja 28kg/m2, a partir de então, a temperatura é recomendada de acordo com a densidade.

Como não possuímos dados suficientes para conhecer a densidade de cada idade, consideraremos a tabela de idade até a idadae que a densidade aumenta gradativamente, e seguiremos a tabela para densidade fornecida por COBB

[6]:
```
def fitness(action_plan_df, age):
    index = age - 1 #index starts from zero
    score = 0
    try:
        y = predict_fcr(action_plan_df)
        score = 1/y[index]
    except Exception as e:
        #print('Fitness: ', e)
        pass

    if action_plan_df['temperature'].max() > MAX_TEMPERATURE or \
       action_plan_df['temperature'].min() < MIN_TEMPERATURE or \
       action_plan_df['humidity'].max() > MAX_HUMIDITY or \
       action_plan_df['humidity'].min() < MIN_HUMIDITY:
        score *= SCORE_PENALTY

    return score
```

1.6 Gerador de soluções

Na primeira geração, estaremos utilizando o nosso conjunto de planos de ação real, e futuramente gerando mutações a partir dele.

Nossas soluções serão uma lista, cada elemento da lista será um DataFrame contendo um plano de ação de um lote real.

[7]:
```
def get_innitial_action_plans():
    daily_stats_df = pd.read_csv('inobram-poultry-daily.csv')
    daily_stats_df['age_col'] = daily_stats_df['age']
    daily_stats_df.set_index(['batch_id', 'age'], inplace=True)
    daily_stats_df.drop('date', axis='columns', inplace=True)
    daily_stats_df.drop('acc_consumption', axis='columns', inplace=True)
    daily_stats_df.drop('living_birds', axis='columns', inplace=True)
    daily_stats_df.dropna(inplace=True)
    action_plans_df = daily_stats_df.loc[:, F_COLUMNS]

    solutions = []

    for batch_id in action_plans_df.index.levels[0].tolist():
```

```
        try:
            solutions.append(action_plans_df.query(f'batch_id == {batch_id}'))
        except:
            pass

    return solutions
```

1.7 Mutação

Como as séries temporais seguem uma tendência de aumento ou declínio com o tempo, não faria muito sentido aleatóriamente acrescentar ou subtrair valores em um dia, se essa tendência não fosse acompanhada nos dias seguintes.

Como tentativa de criar uma mutação que impactasse toda a curva suavemente, foi criada uma função que devolve um segmento aleatório de uma curva seno para um intervalo entre -pi e pi. A lista gerada por esta função pode ser utilizada na mutação dos elementos da série temporal de um Plano de Ação:

[8]:
```
def random_curve_multiplier(lenght):
    phase_min = random.choice(range(0, 31)) # ~ int(pi/2 * 10)
    phase_max = random.choice(range(0, 31))
    phase_steps = abs((phase_max - phase_min)/lenght)
    multipliers = []
    for age in range(lenght):
        multipliers.append(math.sin((phase_min + age*phase_steps)/10))
    return multipliers
```

[9]:
```
def random_column_mutation(original_solutions):
    mutated_solutions = deepcopy(original_solutions)

    for solution in mutated_solutions:
        random_column = random.choice(['temperature', 'humidity', 'pressure'])
        random_curve_weight = random.uniform(0, 0.15)
        curve_multipliers = random_curve_multiplier(50)
        for index_tuple in solution.index:
            try:
                age = index_tuple[1]
                solution.at[index_tuple, random_column] +=␣
↪random_curve_weight*curve_multipliers[age]
            except Exception as e:
                #print('Random column mutation error: ', index_tuple, e)
                pass

    return mutated_solutions
```

[10]:
```
def random_percentage_mutation(original_solutions):
    mutated_solutions = deepcopy(original_solutions)
    temperature_elements = []
    pressure_elements = []
    humidity_elements = []
    for solution in mutated_solutions:
        temperature_elements += solution['temperature'].tolist()
        pressure_elements += solution['pressure'].tolist()
        humidity_elements += solution['humidity'].tolist()

    for solution in mutated_solutions:
        for index_tuple in solution.index:
            try:
                solution.at[index_tuple, 'temperature'] = random.
 ↪choice(temperature_elements) * random.uniform(0.99, 1.01)
                solution.at[index_tuple, 'pressure'] = random.
 ↪choice(pressure_elements) * random.uniform(0.99, 1.01)
                solution.at[index_tuple, 'humidity'] = random.
 ↪choice(humidity_elements) * random.uniform(0.99, 1.01)
                #solution.at[index_tuple, 'temperature'] *= random.uniform(0.
 ↪99, 1.01)
                #solution.at[index_tuple, 'pressure'] *= random.uniform(0.99, 1.
 ↪01)
                #solution.at[index_tuple, 'humidity'] *= random.uniform(0.99, 1.
 ↪01)
            except Exception as e:
                # print('Random percentage mutation error: ', index_tuple, e)
                pass

    return mutated_solutions
```

[11]:
```
def crossover_mutation(original_solutions):
    mutated_solutions = deepcopy(original_solutions)
    for solution in mutated_solutions:
        random_solution = mutated_solutions[random.
 ↪choice(range(len(mutated_solutions)))]
        random_batch_id = int(random_solution.index[0][0])
        random_column = random.choice(['temperature', 'humidity', 'pressure'])
        for index_tuple in solution.index:
            try:
                solution.at[index_tuple, random_column] = (random_solution.
 ↪at[(random_batch_id, index_tuple[1]), random_column] + solution.
 ↪at[index_tuple, random_column])/2
            except Exception as e:
                #print('Crossover mutation error: ', index_tuple, e)
```

```
                pass
    return mutated_solutions
```

[12]:
```
def crossover(original_solutions):
    mutated_solutions = deepcopy(original_solutions)
    for solution in mutated_solutions:
        random_solution = mutated_solutions[random.
↪choice(range(len(mutated_solutions)))]
        random_batch_id = int(random_solution.index[0][0])
        random_column = random.choice(['temperature', 'humidity', 'pressure'])
        for index_tuple in solution.index:
            try:
                solution.at[index_tuple, random_column] = random_solution.
↪at[(random_batch_id, index_tuple[1]), random_column]
            except Exception as e:
                #print('Crossover mutation error: ', index_tuple, e)
                pass
    return mutated_solutions
```

1.8 Classificador de Soluções

A rotina a seguir irá submeter as soluções à avaliação e gerar uma lista ordenada da melhor para a pior.

[13]:
```
solutions = get_innitial_action_plans()
previous_score = None
not_evolving_gen_counter = 0

for generation in range(GENERATIONS):
    ranked_solutions = []
    for solution in solutions:
        try:
            score = fitness(solution, AGE)
            ranked_solutions.append( (score, solution) )
        except:
            print('bad solution: ', solution.head())
    ranked_solutions = sorted(ranked_solutions, key=itemgetter(0), reverse=True)
    best_score = ranked_solutions[0][0]
    best_solutions = [i[1] for i in ranked_solutions][:15]
    print(f'Generation: {generation}, Best score at age {AGE}: {best_score},␣
↪FCR: {1/best_score}')

    #solutions = best_solutions + crossover_mutation(best_solutions) +␣
↪random_column_mutation(best_solutions) +␣
↪random_percentage_mutation(best_solutions)
    solutions = best_solutions + crossover(best_solutions) +␣
↪crossover_mutation(best_solutions) + random_column_mutation(solutions)
```

```
    if best_score == previous_score:
        not_evolving_gen_counter += 1
        if not_evolving_gen_counter == NOT_EVOLVING_TIMEOUT:
            break
    else:
        not_evolving_gen_counter = 0
        previous_score = best_score
```

```
Generation: 0, Best score at age 40: 0.5713590589741756, FCR: 1.7502129077911377
Generation: 1, Best score at age 40: 0.5918837090666371, FCR: 1.689521074295044
Generation: 2, Best score at age 40: 0.5919341202776864, FCR: 1.6893771886825562
Generation: 3, Best score at age 40: 0.5933264244248305, FCR: 1.685412883758545
Generation: 4, Best score at age 40: 0.5961273010494719, FCR: 1.6774940490722656
Generation: 5, Best score at age 40: 0.596269463271987, FCR: 1.6770941019058225
Generation: 6, Best score at age 40: 0.6004589154957565, FCR: 1.6653928756713867
Generation: 7, Best score at age 40: 0.6014444518691464, FCR: 1.6626639366149902
Generation: 8, Best score at age 40: 0.6019866415727907, FCR: 1.6611664295196533
Generation: 9, Best score at age 40: 0.602900453960912, FCR: 1.658648610115051
Generation: 10, Best score at age 40: 0.6032795373753777, FCR:
1.6576063632965088
Generation: 11, Best score at age 40: 0.6041790487505752, FCR:
1.6551384925842283
Generation: 12, Best score at age 40: 0.605725875328872, FCR: 1.650911808013916
Generation: 13, Best score at age 40: 0.6058333595134329, FCR: 1.65061891078949
Generation: 14, Best score at age 40: 0.6064390366906617, FCR:
1.6489703655242922
Generation: 15, Best score at age 40: 0.6069265500629275, FCR:
1.6476458311080933
Generation: 16, Best score at age 40: 0.6083388303220278, FCR:
1.6438207626342773
Generation: 17, Best score at age 40: 0.6094486611227907, FCR:
1.6408272981643677
Generation: 18, Best score at age 40: 0.6094486611227907, FCR:
1.6408272981643677
Generation: 19, Best score at age 40: 0.6100161641112517, FCR: 1.63930082321167
Generation: 20, Best score at age 40: 0.6111966045928373, FCR:
1.6361347436904905
Generation: 21, Best score at age 40: 0.6112665723357653, FCR:
1.6359474658966064
Generation: 22, Best score at age 40: 0.612708499713608, FCR: 1.6320974826812746
Generation: 23, Best score at age 40: 0.6128520995822709, FCR:
1.6317150592803953
Generation: 24, Best score at age 40: 0.6133896292749621, FCR:
1.6302851438522339
Generation: 25, Best score at age 40: 0.6138331784104731, FCR: 1.629107117652893
Generation: 26, Best score at age 40: 0.614348629295524, FCR: 1.6277402639389038
```

```
Generation: 27, Best score at age 40: 0.6148712111161361, FCR:
1.6263568401336672
Generation: 28, Best score at age 40: 0.6154234419720059, FCR:
1.6248974800109866
Generation: 29, Best score at age 40: 0.6169346585729647, FCR:
1.6209172010421753
Generation: 30, Best score at age 40: 0.6182311616036998, FCR:
1.6175179481506348
Generation: 31, Best score at age 40: 0.6185909080953951, FCR:
1.6165772676467893
Generation: 32, Best score at age 40: 0.6194011501360029, FCR: 1.614462614059448
Generation: 33, Best score at age 40: 0.6196856187730745, FCR: 1.613721489906311
Generation: 34, Best score at age 40: 0.6200879443918708, FCR:
1.6126744747161865
Generation: 35, Best score at age 40: 0.6208189894675642, FCR:
1.6107754707336426
Generation: 36, Best score at age 40: 0.6217570895220104, FCR:
1.6083451509475708
Generation: 37, Best score at age 40: 0.622516006844805, FCR: 1.6063843965530396
Generation: 38, Best score at age 40: 0.6227865599809941, FCR:
1.6056865453720093
Generation: 39, Best score at age 40: 0.6243425638732513, FCR: 1.601684808731079
Generation: 40, Best score at age 40: 0.6245682021036029, FCR:
1.6011061668395996
Generation: 41, Best score at age 40: 0.6246149399179318, FCR: 1.600986361503601
Generation: 42, Best score at age 40: 0.6264937147548143, FCR:
1.5961852073669436
Generation: 43, Best score at age 40: 0.6275319017147405, FCR:
1.5935444831848147
Generation: 44, Best score at age 40: 0.6275319017147405, FCR:
1.5935444831848147
Generation: 45, Best score at age 40: 0.6282671098070346, FCR:
1.5916796922683716
Generation: 46, Best score at age 40: 0.6283439587773918, FCR:
1.5914850234985352
Generation: 47, Best score at age 40: 0.629648122166301, FCR: 1.588188648223877
Generation: 48, Best score at age 40: 0.6302954637719458, FCR:
1.5865575075149536
Generation: 49, Best score at age 40: 0.6309081681493589, FCR:
1.5850167274475098
Generation: 50, Best score at age 40: 0.6318764118830555, FCR:
1.5825879573822024
Generation: 51, Best score at age 40: 0.6318943562470335, FCR:
1.5825430154800415
Generation: 52, Best score at age 40: 0.6326398756411256, FCR: 1.580678105354309
Generation: 53, Best score at age 40: 0.6334269964114446, FCR:
1.5787138938903809
Generation: 54, Best score at age 40: 0.6344142677504573, FCR:
```

```
1.5762571096420288
Generation: 55, Best score at age 40: 0.6350156849920666, FCR:
1.5747642517089844
Generation: 56, Best score at age 40: 0.6362998854849413, FCR: 1.571586012840271
Generation: 57, Best score at age 40: 0.6363536091485056, FCR: 1.571453332901001
Generation: 58, Best score at age 40: 0.638120525901609, FCR: 1.567102074623108
Generation: 59, Best score at age 40: 0.6387021489688388, FCR:
1.5656750202178955
Generation: 60, Best score at age 40: 0.6393679587828103, FCR:
1.5640445947647095
Generation: 61, Best score at age 40: 0.6395729414075318, FCR:
1.5635433197021484
Generation: 62, Best score at age 40: 0.6409229109971741, FCR:
1.5602500438690186
Generation: 63, Best score at age 40: 0.6409553301490625, FCR: 1.560171127319336
Generation: 64, Best score at age 40: 0.6420994425743841, FCR:
1.5573911666870117
Generation: 65, Best score at age 40: 0.6422533158096942, FCR:
1.5570180416107178
Generation: 66, Best score at age 40: 0.6430190083890728, FCR:
1.5551639795303345
Generation: 67, Best score at age 40: 0.6441086299386529, FCR: 1.552533149719238
Generation: 68, Best score at age 40: 0.645698633308707, FCR: 1.5487101078033447
Generation: 69, Best score at age 40: 0.6471116020752664, FCR:
1.5453284978866577
Generation: 70, Best score at age 40: 0.6476032311113589, FCR:
1.5441553592681885
Generation: 71, Best score at age 40: 0.6476036310730249, FCR: 1.544154405593872
Generation: 72, Best score at age 40: 0.648116187653277, FCR: 1.5429332256317139
Generation: 73, Best score at age 40: 0.648846092145865, FCR: 1.5411975383758545
Generation: 74, Best score at age 40: 0.6493109582401326, FCR:
1.5400941371917725
Generation: 75, Best score at age 40: 0.6494148103533434, FCR:
1.5398478507995605
Generation: 76, Best score at age 40: 0.6501787358133462, FCR:
1.5380386114120483
Generation: 77, Best score at age 40: 0.650216533231307, FCR: 1.5379492044448855
Generation: 78, Best score at age 40: 0.6502440021121322, FCR: 1.53788423538208
Generation: 79, Best score at age 40: 0.650699668638844, FCR: 1.5368072986602783
Generation: 80, Best score at age 40: 0.650824566303572, FCR: 1.5365123748779297
Generation: 81, Best score at age 40: 0.6518755216258095, FCR:
1.5340352058410645
Generation: 82, Best score at age 40: 0.6528166162627805, FCR: 1.531823754310608
Generation: 83, Best score at age 40: 0.6528453722353657, FCR:
1.5317562818527222
Generation: 84, Best score at age 40: 0.652870116592844, FCR: 1.531698226928711
Generation: 85, Best score at age 40: 0.652903298315876, FCR: 1.5316203832626343
Generation: 86, Best score at age 40: 0.6530510563588612, FCR:
```

```
1.5312738418579102
Generation: 87, Best score at age 40: 0.65455714475214, FCR: 1.5277504920959473
Generation: 88, Best score at age 40: 0.6545757364457709, FCR:
1.5277070999145508
Generation: 89, Best score at age 40: 0.6546237017629468, FCR:
1.5275951623916626
Generation: 90, Best score at age 40: 0.6547753061133691, FCR:
1.5272414684295654
Generation: 91, Best score at age 40: 0.6547920701903708, FCR:
1.5272023677825928
Generation: 92, Best score at age 40: 0.654848399687744, FCR: 1.5270709991455076
Generation: 93, Best score at age 40: 0.6552694909015693, FCR:
1.5260896682739256
Generation: 94, Best score at age 40: 0.6552694909015693, FCR:
1.5260896682739256
Generation: 95, Best score at age 40: 0.6563736871130352, FCR:
1.5235223770141602
Generation: 96, Best score at age 40: 0.6564322410387217, FCR:
1.5233864784240723
Generation: 97, Best score at age 40: 0.6564665564391978, FCR:
1.5233068466186523
Generation: 98, Best score at age 40: 0.6564897265004395, FCR: 1.523253083229065
Generation: 99, Best score at age 40: 0.6565384865834172, FCR:
1.5231399536132815
Generation: 100, Best score at age 40: 0.6565629464234001, FCR:
1.5230832099914553
Generation: 101, Best score at age 40: 0.6565968129497681, FCR:
1.5230046510696411
Generation: 102, Best score at age 40: 0.6566337155095615, FCR:
1.5229190587997437
Generation: 103, Best score at age 40: 0.6566505748837774, FCR:
1.522879958152771
Generation: 104, Best score at age 40: 0.6566777675889288, FCR:
1.5228168964385986
Generation: 105, Best score at age 40: 0.6566858384471085, FCR:
1.5227981805801392
Generation: 106, Best score at age 40: 0.6567088183464318, FCR:
1.5227448940277102
Generation: 107, Best score at age 40: 0.6567613131424401, FCR:
1.5226231813430786
Generation: 108, Best score at age 40: 0.6567969485740521, FCR: 1.52254056930542
Generation: 109, Best score at age 40: 0.6568400967899776, FCR:
1.5224405527114868
Generation: 110, Best score at age 40: 0.65688227334956, FCR: 1.5223428010940552
Generation: 111, Best score at age 40: 0.65688227334956, FCR: 1.5223428010940552
Generation: 112, Best score at age 40: 0.6569094338059402, FCR:
1.5222798585891724
Generation: 113, Best score at age 40: 0.6569272847986861, FCR:
```

```
1.5222384929656982
Generation: 114, Best score at age 40: 0.6569472461354676, FCR:
1.5221922397613525
Generation: 115, Best score at age 40: 0.6569889219082563, FCR:
1.5220956802368164
Generation: 116, Best score at age 40: 0.6569889219082563, FCR:
1.5220956802368164
Generation: 117, Best score at age 40: 0.6570140328276436, FCR:
1.5220375061035156
Generation: 118, Best score at age 40: 0.6570281328328752, FCR:
1.5220048427581787
Generation: 119, Best score at age 40: 0.6570434170987307, FCR:
1.5219694375991821
Generation: 120, Best score at age 40: 0.6570772302739942, FCR:
1.5218911170959473
Generation: 121, Best score at age 40: 0.6570877300502007, FCR:
1.521866798400879
Generation: 122, Best score at age 40: 0.6571143927545308, FCR:
1.5218050479888916
Generation: 123, Best score at age 40: 0.6571563986240512, FCR:
1.5217077732086182
Generation: 124, Best score at age 40: 0.657187237233273, FCR:
1.5216363668441775
Generation: 125, Best score at age 40: 0.6572237942584208, FCR:
1.5215517282485962
Generation: 126, Best score at age 40: 0.6572393450992169, FCR:
1.5215157270431519
Generation: 127, Best score at age 40: 0.6572820879558027, FCR:
1.5214167833328247
Generation: 128, Best score at age 40: 0.6572820879558027, FCR:
1.5214167833328247
Generation: 129, Best score at age 40: 0.6573073243003745, FCR:
1.5213583707809448
Generation: 130, Best score at age 40: 0.6573669206660773, FCR:
1.5212204456329346
Generation: 131, Best score at age 40: 0.6573953576581243, FCR:
1.5211546421051025
Generation: 132, Best score at age 40: 0.6574193661645104, FCR:
1.5210990905761719
Generation: 133, Best score at age 40: 0.6574487867038191, FCR:
1.5210310220718384
Generation: 134, Best score at age 40: 0.6574583708456373, FCR:
1.521008849143982
Generation: 135, Best score at age 40: 0.6574833630673812, FCR:
1.5209510326385498
Generation: 136, Best score at age 40: 0.6574935666236938, FCR:
1.5209274291992188
Generation: 137, Best score at age 40: 0.6575218599590921, FCR:
```

```
1.5208619832992554
Generation: 138, Best score at age 40: 0.657734575975814, FCR:
1.5203701257705688
Generation: 139, Best score at age 40: 0.658234065914837, FCR:
1.5192164182662964
Generation: 140, Best score at age 40: 0.6583539675351736, FCR:
1.518939733505249
Generation: 141, Best score at age 40: 0.6596503556058235, FCR:
1.5159546136856077
Generation: 142, Best score at age 40: 0.6601674112948598, FCR:
1.5147672891616821
Generation: 143, Best score at age 40: 0.6602779357687752, FCR:
1.5145137310028076
Generation: 144, Best score at age 40: 0.6605386748151346, FCR:
1.5139158964157104
Generation: 145, Best score at age 40: 0.660702554413237, FCR:
1.5135403871536255
Generation: 146, Best score at age 40: 0.6609101999384204, FCR:
1.5130648612976074
Generation: 147, Best score at age 40: 0.6613656131206437, FCR:
1.5120229721069336
Generation: 148, Best score at age 40: 0.6617782146536803, FCR:
1.5110802650451662
Generation: 149, Best score at age 40: 0.6632366083235222, FCR:
1.507757544517517
Generation: 150, Best score at age 40: 0.6632449460887672, FCR:
1.5077385902404785
Generation: 151, Best score at age 40: 0.6633388784170092, FCR:
1.507525086402893
Generation: 152, Best score at age 40: 0.6633412388684099, FCR:
1.507519721984863
Generation: 153, Best score at age 40: 0.6640169655595057, FCR:
1.5059856176376343
Generation: 154, Best score at age 40: 0.6640253229566712, FCR:
1.5059666633605957
Generation: 155, Best score at age 40: 0.6640282664925605, FCR:
1.5059599876403809
Generation: 156, Best score at age 40: 0.6655222650171775, FCR:
1.5025793313980103
Generation: 157, Best score at age 40: 0.6655478740825181, FCR:
1.5025215148925781
Generation: 158, Best score at age 40: 0.6655633988877931, FCR:
1.5024864673614502
Generation: 159, Best score at age 40: 0.6655730626527194, FCR:
1.5024646520614624
Generation: 160, Best score at age 40: 0.6655874796105214, FCR:
1.502432107925415
Generation: 161, Best score at age 40: 0.6656167379208998, FCR:
```

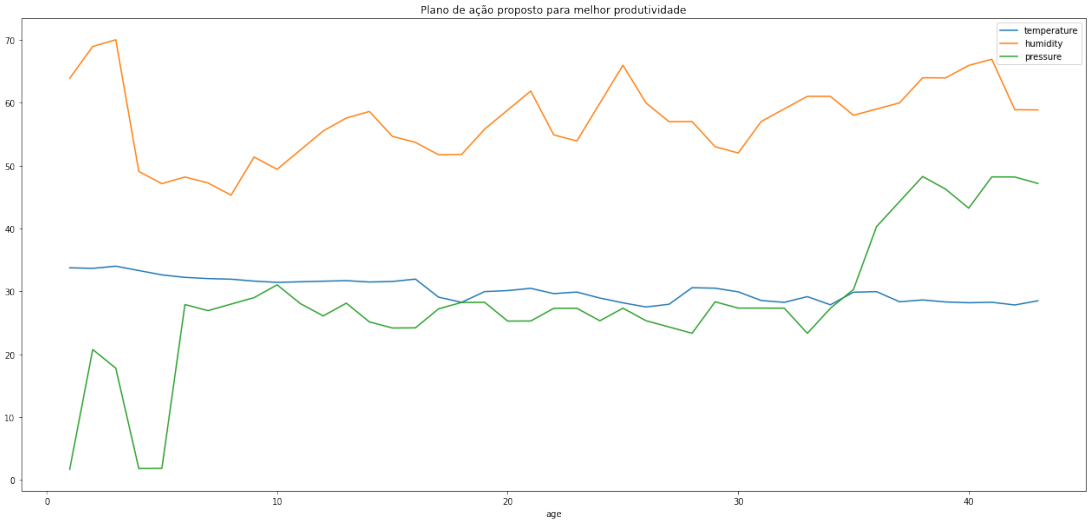
```
1.5023660659790041
Generation: 162, Best score at age 40: 0.6656633769715417, FCR:
1.5022608041763306
Generation: 163, Best score at age 40: 0.66566554270346, FCR: 1.502255916595459
Generation: 164, Best score at age 40: 0.665689630818212, FCR:
1.5022015571594238
Generation: 165, Best score at age 40: 0.6657165207071797, FCR:
1.5021408796310425
Generation: 166, Best score at age 40: 0.6658439206061694, FCR:
1.5018534660339355
Generation: 167, Best score at age 40: 0.6658638989649377, FCR:
1.5018084049224854
Generation: 168, Best score at age 40: 0.6658839842377602, FCR:
1.501763105392456
Generation: 169, Best score at age 40: 0.6658990489875821, FCR:
1.501729130744934
Generation: 170, Best score at age 40: 0.6658990489875821, FCR:
1.501729130744934
Generation: 171, Best score at age 40: 0.6659345727805079, FCR:
1.501649022102356
Generation: 172, Best score at age 40: 0.66593769186594, FCR: 1.5016419887542725
Generation: 173, Best score at age 40: 0.6659652362445236, FCR:
1.5015798807144165
Generation: 174, Best score at age 40: 0.6660024062634622, FCR:
1.5014960765838623
Generation: 175, Best score at age 40: 0.6660236103659998, FCR:
1.5014482736587524
Generation: 176, Best score at age 40: 0.6660236103659998, FCR:
1.5014482736587524
Generation: 177, Best score at age 40: 0.6660468782665234, FCR: 1.50139582157135
Generation: 178, Best score at age 40: 0.6660927313650878, FCR:
1.5012924671173096
Generation: 179, Best score at age 40: 0.6661495408443635, FCR:
1.501164436340332
Generation: 180, Best score at age 40: 0.6661495408443635, FCR:
1.501164436340332
Generation: 181, Best score at age 40: 0.6661592216380913, FCR:
1.5011426210403442
Generation: 182, Best score at age 40: 0.666204032037104, FCR:
1.5010416507720947
Generation: 183, Best score at age 40: 0.6662157779085899, FCR:
1.5010151863098145
Generation: 184, Best score at age 40: 0.666232497971585, FCR:
1.5009775161743164
Generation: 185, Best score at age 40: 0.6662349848880933, FCR:
1.5009719133377075
Generation: 186, Best score at age 40: 0.6662447210802802, FCR:
1.5009499788284302
```

```
Generation: 187, Best score at age 40: 0.6662820281463897, FCR:
1.5008659362792969
Generation: 188, Best score at age 40: 0.6662820281463897, FCR:
1.5008659362792969
Generation: 189, Best score at age 40: 0.6662864206007056, FCR:
1.5008560419082642
Generation: 190, Best score at age 40: 0.6663037263758517, FCR:
1.500817060470581
Generation: 191, Best score at age 40: 0.6663268021411782, FCR:
1.5007650852203371
Generation: 192, Best score at age 40: 0.6663268021411782, FCR:
1.5007650852203371
Generation: 193, Best score at age 40: 0.6663281253413116, FCR:
1.500762104988098
Generation: 194, Best score at age 40: 0.6663360646524727, FCR:
1.5007442235946655
Generation: 195, Best score at age 40: 0.6663450627671731, FCR:
1.500723958015442
Generation: 196, Best score at age 40: 0.6663522614339087, FCR:
1.500707745552063
Generation: 197, Best score at age 40: 0.6663557549605867, FCR:
1.5006998777389526
Generation: 198, Best score at age 40: 0.666358083998723, FCR:
1.5006946325302124
Generation: 199, Best score at age 40: 0.6663690412829709, FCR:
1.5006699562072754
```

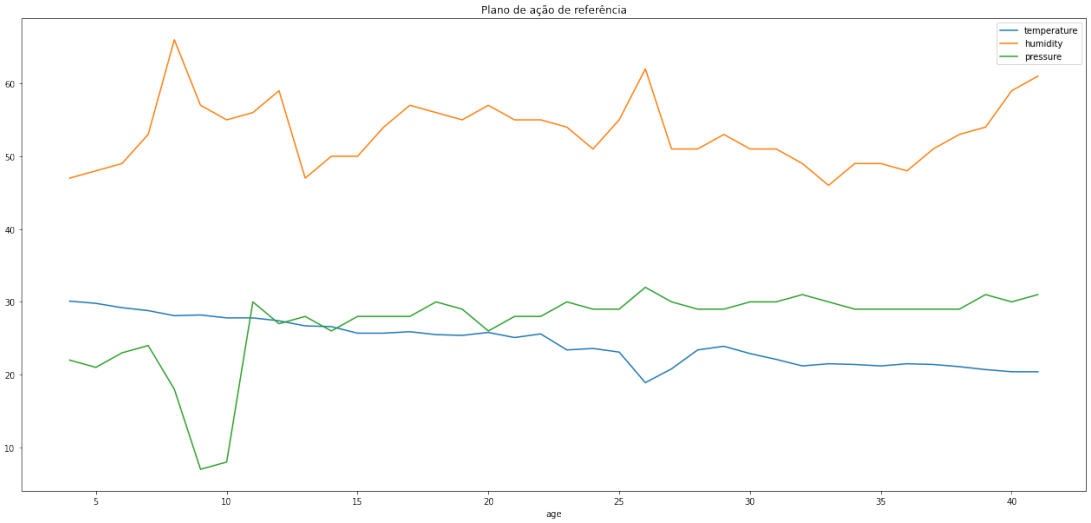
[14]:
```
proposed_action_plan = solutions[0].loc[:, ['temperature', 'pressure',␣
↪'humidity']]
proposed_action_plan.reset_index(level=0, drop=True, inplace=True)
proposed_action_plan['fcr'] = predict_fcr(solutions[0])
```

[15]:
```
reference = get_innitial_action_plans()[45]
reference_action_plan = reference.loc[:, ['temperature', 'pressure',␣
↪'humidity']]
reference_action_plan['fcr'] = predict_fcr(reference)
reference_action_plan.reset_index(level=0, drop=True, inplace=True)
```

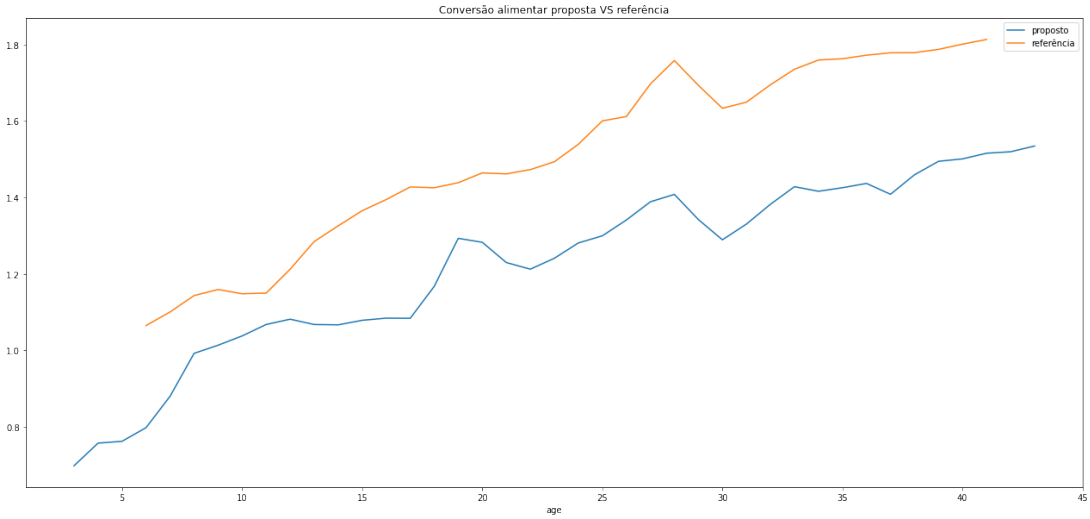
[16]:
```
proposed_action_plan[['temperature', 'humidity', 'pressure']].
↪plot(label='proposed').set_title('Plano de ação proposto para melhor␣
↪produtividade')
plt.legend()
plt.savefig("proposed-action-plan.svg")
```



[17]:
```
reference_action_plan[['temperature', 'humidity', 'pressure']].
 ↪plot(label='reference').set_title('Plano de ação de referência')
plt.legend()
plt.savefig("reference_action_plan.svg")
```



[18]:
```
proposed_action_plan['fcr'].plot(label='proposto').set_title('Conversão␣
 ↪alimentar proposta VS referência')
reference_action_plan['fcr'].plot(label='referência')
plt.legend()
plt.savefig('fcr-proposed-vs-reference.svg')
```



[19]:
```
reference_action_plan['fcr'][40]
```

[19]: 1.8009952306747437

[20]:
```
proposed_action_plan
```

[20]:

age	temperature	pressure	humidity	fcr
1	33.743847	1.708727	63.865681	NaN
2	33.656893	20.749391	68.931495	NaN
3	33.999915	17.789147	69.996015	0.697545
4	33.310318	1.827934	49.059159	0.756728
5	32.618926	1.865693	47.120845	0.761502
6	32.225566	27.902366	48.180994	0.797244
7	32.030065	26.937898	47.239528	0.879415
8	31.932256	27.972232	45.296372	0.991703
9	31.631975	29.005317	51.351453	1.012961
10	31.429062	31.037101	49.404700	1.037274
11	31.523363	28.067533	52.456045	1.067222
12	31.614729	26.096568	55.505422	1.081095
13	31.703015	28.124157	57.552768	1.067161
14	31.488083	25.150259	58.598023	1.066274
15	31.569802	24.174832	54.641129	1.078161
16	31.948047	24.197837	53.682032	1.083813
17	29.081707	27.219236	51.720680	1.083468
18	28.251745	28.238995	51.757026	1.167096
19	29.960783	28.257082	55.791025	1.292691
20	30.124016	25.273468	58.822634	1.282371
21	30.483254	25.288126	61.851816	1.229407

```
22      29.638418  27.301031  54.878536  1.211968
23      29.889435  27.312162  53.902761  1.240492
24      28.936240  25.321500  59.924466  1.280549
25      28.178776  27.329028  65.943625  1.299282
26      27.516998  25.334732  59.960217  1.340624
27      27.950866  24.338603  56.974226  1.388288
28      30.580350  23.340633  56.985638  1.407717
29      30.505429  28.340815  52.994443  1.341813
30      29.926090  27.339149  52.000636  1.288894
31      28.542330  27.335634  57.004214  1.330057
32      28.254153  27.330274  59.005178  1.381817
33      29.161574  23.323076  61.003533  1.427918
34      27.864616  27.314049  60.999287  1.415885
35      29.863308  30.303204  57.992454  1.425205
36      29.957690  40.290556  58.983048  1.436423
37      28.347811  44.276123  59.971089  1.408029
38      28.633726  48.259925  63.956600  1.459094
39      28.315499  46.241985  63.939606  1.494332
40      28.193202  43.222328  65.920137  1.500670
41      28.266914  48.200983  66.898227  1.515504
42      27.836723  48.177981  58.873910  1.519320
43      28.502721  47.153355  58.847227  1.534318
```

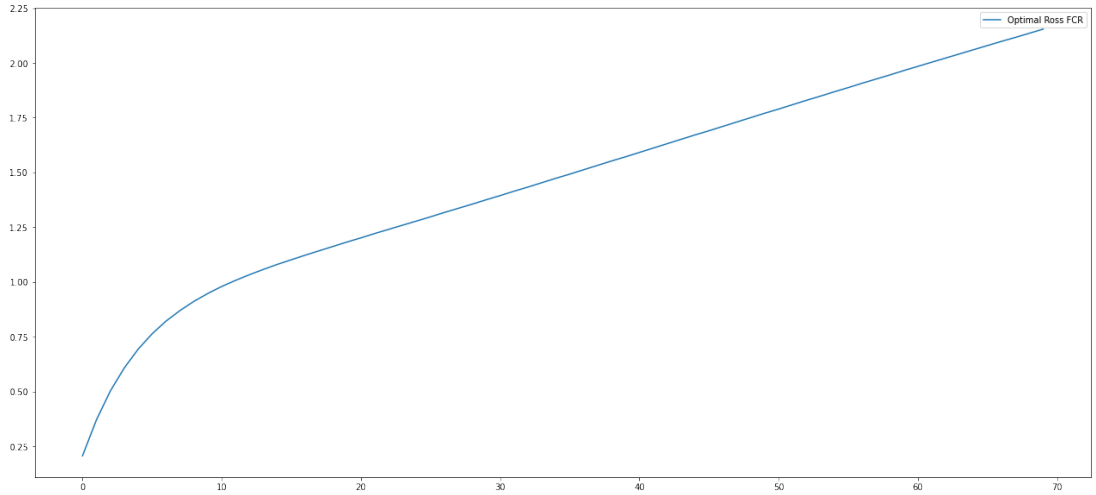
1.9 Desenhando a curva de conversão alimentar ótima sugerida no guia de referência de produtividade Ross

A curva a seguir representa os valores ótimos de conversão alimentar segundo o guia de desempenho Ross:

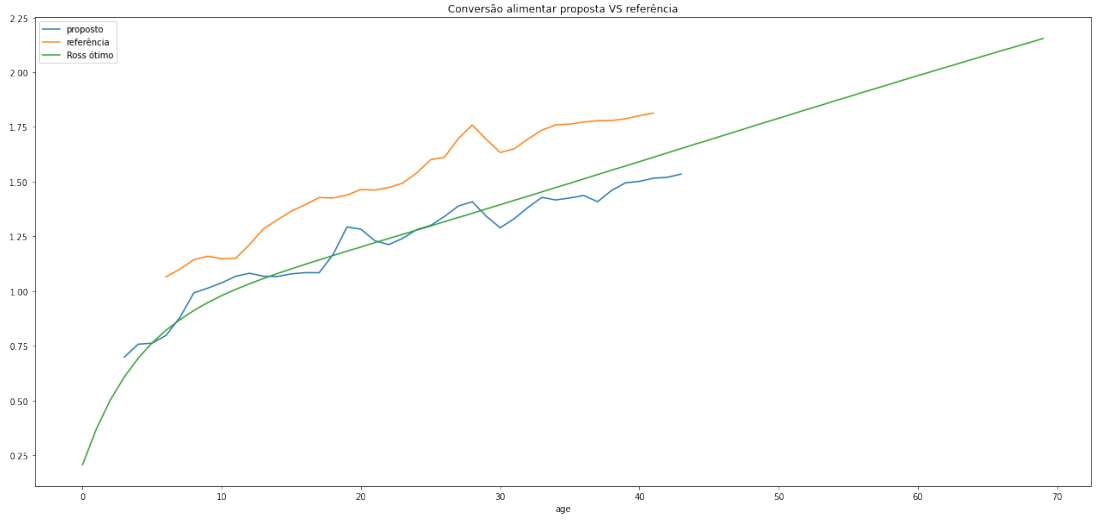
[21]:
```
ross_optimal_fcr_list = [0.206, 0.370, 0.502, 0.607, 0.693, 0.763, 0.821, 0.
↪869, 0.911, 0.947, 0.979, 1.007, 1.033, 1.057, 1.080, 1.101, 1.122, 1.142, 1.
↪162, 1.182, 1.201, 1.221, 1.240, 1.259, 1.278, 1.297, 1.317, 1.336, 1.355, 1.
↪375, 1.394, 1.414, 1.433, 1.453, 1.473, 1.492, 1.512, 1.532, 1.552, 1.571, 1.
↪591, 1.611, 1.631, 1.651, 1.671, 1.690, 1.710, 1.730, 1.750, 1.770, 1.789, 1.
↪809, 1.829, 1.848, 1.868, 1.887, 1.907, 1.926, 1.945, 1.965, 1.984, 2.003, 2.
↪022, 2.041, 2.060, 2.079, 2.098, 2.116, 2.135, 2.154]
ross_optimal_df = pd.DataFrame()
ross_optimal_df['fcr'] = pd.DataFrame(ross_optimal_fcr_list)
```

[22]:
```
ross_optimal_df['fcr'].plot(legend=True, label='Optimal Ross FCR')
```

[22]:
```
<AxesSubplot:>
```



[23]:
```
proposed_action_plan['fcr'].plot(label='proposto').set_title('Conversão␣
 ↪alimentar proposta VS referência')
reference_action_plan['fcr'].plot(label='referência')
ross_optimal_df['fcr'].plot(label='Ross ótimo')
plt.legend()
plt.savefig('fcr-proposed-vs-reference-vs-optimal.svg')
```



[ ]: