

MARCUS VINÍCIUS SIMÕES

Sistema de supervisão em Python capaz de identificar características referentes à laminação de metal.

Marcus Vinícius Simões

Sistema de supervisão em Python capaz de identificar características referentes à laminação de metal.

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador (a): Prof. Dr. Paloma Maria Silva Rocha Rizol

S593s	Simões, Marcus Vinícius Sistema de supervisão em Python capaz de identificar características referentes à laminação de metal / Marcus Vinícius Simões – Guaratinguetá, 2022. 62 f : il. Bibliografia: f. 51 Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia e Ciências de Guaratinguetá, 2022. Orientadora: Prof ^a . Dr ^a . Paloma Maria Silva Rocha Rizol 1. Laminação (Metalurgia). 2. Controle de processo. 3. Aprendizado do computador. I. Título.
-------	--

CDU 621.771

Luciana Máximo
Bibliotecária CRB-8/3595

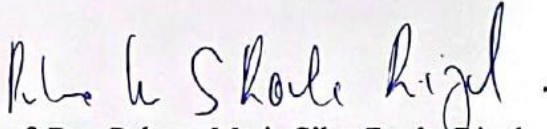
MARCUS VINÍCIUS SIMÕES

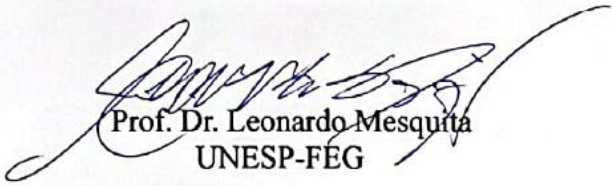
**ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO
PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE
“GRADUADO EM ENGENHARIA ELÉTRICA”**

**APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE
GRADUAÇÃO EM ENGENHARIA ELÉTRICA**


Prof. Dr. Daniel Julien Barros da Silva Sampaio
Coordenador

BANCA EXAMINADORA:


Prof. Dra. Paloma Maria Silva Rocha Rizol
Orientador/UNESP-FEG


Prof. Dr. Leonardo Mesquita
UNESP-FEG


Eng João Otávio Belizário Tonhão
Membro Externo

DADOS CURRICULARES

MARCUS VINÍCIUS SIMÕES

NASCIMENTO 18.08.1998 – Bragança Paulista / SP

FILIAÇÃO Marcos Eli Simões
Carmen Silvia Montagnana Simões

Dedico este trabalho a todos que me acompanharam e se esforçaram de alguma forma a me ajudar a chegar até aqui. Em especial aos meus pais e meu irmão.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer aos meus pais, *Carmen Silvia Montagnana Simões* e *Marcos Eli Simões*, que batalharam durante anos para que eu tivesse a oportunidade de seguir meus sonhos, além de terem me dado atenção, cuidado e educação com o maior amor que existe nesse mundo. Agradeço também ao meu irmão, *Pedro Augusto Simões*, pela companhia, compreensão, inúmeros dias de conversas, diversão e por nunca ter deixado de estar ao meu lado;

à minha orientadora, *Prof. Dr. Paloma Maria Silva Rocha Rizol* que se dispôs a me acompanhar durante a execução desse trabalho. Sem a sua orientação, dedicação e auxílio, o estudo aqui apresentado seria praticamente impossível;

ao *João Otávio Belizário Tonhão* pela idealização inicial do trabalho, pela ajuda e pela disposição ao longo do desenvolvimento desse sistema;

aos meus amigos *Bruno Kenzo Suzuki*, *Bruno Hideki Yoshida*, *Rodrigo Carvalho Guerra*, *Guilherme Cavalcante da Silva* e *Fábio Augusto Abreu Gama dos Santos*, que apesar das dificuldades apresentadas ao longo dessa graduação, permaneceram comigo até o fim. Pode se dizer com absoluta certeza que sem vocês, muitos obstáculos seriam maiores do que foram;

aos meus amigos e muito queridos companheiros de apartamento *Arthur Cardoso Leite* e *Alan Vitor Pereira Rodrigues* pelas conversas, músicas, tempos de estudo, reuniões, conselhos e tempo dividido durante boa parte da graduação;

aos meus amigos de infância *Matheus Finamor*, *Artur Alves da Fonseca Pedrosa*, *Pedro Vinícius de Paiva* e meus amigos de escola *Leonardo Amaro de Oliveira* e *Eliézer Taffuri* pelas várias noites de videogame, conversa, inúmeros conselhos e reflexões nesses últimos longos e cheios de reviravoltas, seis anos;

às funcionárias da Biblioteca do Campus de Guaratinguetá pela dedicação, presteza e principalmente pela vontade de ajudar;

e finalmente à todas as pessoas que contribuíram de forma direta ou indireta à minha instrução e saúde durante todo esse período.

RESUMO

A necessidade de melhoria do processo industrial da laminação de metal de forma a otimizar o tempo de entrega de um produto ao cliente foi o ponto de partida da idealização de um sistema de supervisão em Python aplicado para essa necessidade. Sendo assim, os processos de laminação de metal, que necessitam de uma análise de alta precisão da produção, podem se aproveitar das tecnologias de *machine learning* para auxiliar no controle de conformidade do produto, aumentando a velocidade da produção e melhorando a confiabilidade do processo que atualmente é responsabilidade de um especialista humano. O programa que analisa esses dados e toma as decisões, deve também conter uma interface gráfica para interação máquina-humano de forma que, ao utilizar o sistema, o supervisor seja capaz de ler os dados, entender a tomada de decisão da máquina e até interferir em alguns casos. Dessa maneira, desenvolveu-se um sistema de supervisão que é diretamente conectado ao banco de dados de produção, contendo elementos gráficos relevantes do produto e permitindo com que o usuário especialista seja capaz de acompanhar os dados em tempo real e até mesmo alterar algumas características da produção, caso julgue necessário.

PALAVRAS-CHAVE: Aprendizado de máquina. Python. Interface Gráfica. Metal. Controle de Processos.

ABSTRACT

The need to improve the industrial process of metal lamination in order to optimize the delivery time of a product to the customer was the starting point for the idealization of a supervisory system in Python applied to this need. Thus, metal rolling processes, which require a high-precision production analysis, can take advantage of machine learning technologies to help control product compliance, increasing production speed and improving process reliability. that is currently the responsibility of a human expert. The program that analyzes this data and makes decisions must also contain a graphical interface for machine-human interaction so that, when using the system, the supervisor is able to read the data, understand the machine's decision making and even interfere in some cases. In this way, a supervisory system was developed that is directly connected to the production database, containing relevant graphic elements of the product and allowing the expert to be able to follow the data in real time and even change some characteristics of the product if deemed necessary.

KEYWORDS: Machine learning. Python. Graphic interface. Metal. Process control.

LISTA DE ILUSTRAÇÕES

Figura 1 - Scopus (supervisory and system) AND TITLE-ABS-KEY (python)	13
Figura 2 - Scopus (supervisory and system and python and industrial)	13
Figura 3 - Linha do processamento de dados anterior ao sistema de supervisão	14
Figura 4 – Exemplo de janela Tkinter com um botão	16
Figura 5 – Janela da Interface de Produção sem elementos gráficos	21
Figura 6 – Janela com a Frame da Treeview e Scrollbar.	24
Figura 7 – Janela com Frame e Treeview definidos sem dados	26
Figura 8 – Treeview com dados.....	29
Figura 9 – Treeview completa com os dados	30
Figura 10 – Janela com Treeview e informação da produção	32
Figura 11 – Janela com todos os elementos gráficos.....	35
Figura 12 – Tela de configuração da tabela pública de dados results.....	41
Figura 13 – Tabela results com os dados	41
Figura 14 – Programa com linha selecionada em modo janela	45
Figura 15 – Programa com linha selecionada em tela cheia.....	45
Figura 16 – Exemplo de mudança de Descarte Espessura no programa anterior.....	47
Figura 17 – Exemplo de mudança de Descarte Espessura no banco de dados anterior	47
Figura 18 – Exemplo de mudança de Descarte Espessura no programa após.....	48
Figura 19 – Exemplo de mudança de Descarte Espessura no banco de dados após	48
Figura 20 – Exemplo de nova linha no banco de dados	49
Figura 21 – Exemplo de nova linha no programa	49

LISTA DE QUADROS

Quadro 1 – Exemplo do código de uma janela com um elemento de botão Tkinter.....	17
Quadro 2 – Exemplo de conexão à base PostgreSQL	19
Quadro 3 – Definição das propriedades da janela	20
Quadro 4 – Definição do frame e propriedades iniciais da Treeview	22
Quadro 5 – Definição e atribuição da Scrollbar na Treeview.....	23
Quadro 6 – Definição das colunas da Treeview	25
Quadro 7 – Formatação das colunas da Treeview	25
Quadro 8 – Definição dos cabeçalhos da Treeview.....	26
Quadro 9 – Inserção dos dados na Treeview	28
Quadro 10 – Definir as cores da linha pela etiqueta.....	29
Quadro 11 – Código da seção de contagem de bobinas	31
Quadro 12 – Definição da moldura e propriedades da área de verificação dos dados.....	33
Quadro 13 – Espaço de verificação dos dados	34
Quadro 14 – Definição do botão Editar.....	35
Quadro 15 – Função de seleção de dados.....	36
Quadro 16 – Função de edição de dados	38
Quadro 17 – Função update.....	39
Quadro 18 – Função update_loop.....	39
Quadro 19 – Configuração de conexão ao banco de dados PostgreSQL	42
Quadro 20 – Treeview conectada ao banco de dados.....	43
Quadro 21 – Função edit_record com conexão ao banco de dados.....	44

SUMÁRIO

1	INTRODUÇÃO	11
1.1	CONTEXTUALIZAÇÃO	11
1.2	JUSTIFICATIVA	11
1.3	DELIMITAÇÃO.....	13
1.4	OBJETIVO	15
1.5	ESTRUTURA DO TRABALHO	15
2	REFERENCIAL TEÓRICO	16
2.1	PYTHON E A BIBLIOTECA TKINTER	16
2.2	POSTGRESQL E A BIBLIOTECA PSYCOPG2	18
3	DESENVOLVIMENTO	20
3.1	PROPRIEDADES DA JANELA DA INTERFACE	20
3.2	DEFINIÇÃO DA <i>TREEVIEW</i>	21
3.2.1	Definição do <i>Frame</i> e propriedades da <i>Treeview</i>	21
3.2.2	<i>Scrollbar</i> da <i>Treeview</i>	23
3.2.3	Elementos da <i>Treeview</i>	24
3.3	SEÇÃO DE CONTAGEM DE BOBINAS	30
3.4	ÁREA DE VERIFICAÇÃO DOS DADOS	33
3.5	FUNÇÕES DO PROGRAMA	35
3.5.1	Seleção dos dados	35
3.5.2	Edição dos dados	37
3.5.3	Funções de atualização da Janela	38
3.6	CONEXÃO COM O BANCO DE DADOS	40
4	RESULTADOS	45
5	CONCLUSÃO	50

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Com o avanço da tecnologia da informação, decisões automáticas e mais rápidas passaram a ser desejadas em muitas etapas da indústria. Esse fenômeno da integração da tecnologia informacional e a grande necessidade de dados para acompanhar a tomada de decisão de forma automática através de um sistema integrado e inteligente está presente no escopo da Indústria 4.0 (SACOMANO, 2018).

Dentre os vários tipos de produção industrial, estão os processos de laminação de metal. Atualmente, os processos de laminação consistem de muitos sensores para aquisição de dados referentes ao produto e é especialmente importante que esses sejam captados com precisão e analisados de forma crítica, pois qualquer variação no processo de fabricação pode acarretar em defeitos graves no produto final.

De forma simplificada, o processo de laminação de metal consiste de três partes chave para seu funcionamento, sendo elas: Os atuadores, cilindros de laminação e sistemas de lubrificação e arrefecimento.

Para garantir que o processo seja feito com precisão, o sistema que compõe os atuadores é equipado com uma grande quantidade de sensores. Esses sensores fazem uma checagem das características do material produzido, levantando dados de diâmetro dos cilindros de laminação, a dilatação térmica, espessura, comprimento, quantidade de descarte e etc.

Apesar dessa instrumentação ser principalmente utilizada para definir o fim de curso de atuadores, ela também é bastante necessária no controle da qualidade do produto, podendo gerar um relatório das características do material produzido e, se comparado ao pedido de um cliente, ou seja, um alvo (target), pode ser feito um estudo do desvio da produção e analisar se esse produto está dentro da conformidade.

1.2 JUSTIFICATIVA

Nos dias de hoje, o responsável pela análise dos arquivos de produção é um especialista humano que tem a responsabilidade de tomar a decisão quanto ao destino do metal produzido. Entre as decisões estão: aprovação do material, redirecionamento do material para um outro cliente ou a sucateamento. Se o material está conforme o pedido de um primeiro cliente, então

ele será aprovado, se não, analisa-se se algum processo pode tirar essa imperfeição e se com o novo processo ele pode ser redirecionado a outro cliente e por fim, se nada puder ser feito, então ele é sucitado.

Durante essa etapa do processo, o material fica parado, esperando a decisão do especialista e isso aumenta o tempo com que a produção passa de um processo para outro (chamado de *lead time*) além de ficar a cargo de um profissional que necessita de uma vasta experiência para diminuir a chance do material ser aprovado incorretamente.

Sendo assim, surge a necessidade de implementar um sistema que seja ajudar o especialista a tomar a decisão adequada, integrando os dados dos processos de laminação à uma classificação automática do material, pois em contraste com seres humanos, se desenvolvido corretamente, os sistemas de inspeção automatizados são mais precisos, consistentes e mais rápidos (YACOB, SEMERE e NORDGREN, 2019).

Para o desenvolvimento desse sistema, foi optado a utilização da linguagem *Python* de programação por algumas razões principais, essas são: O desenvolvimento de qualquer projeto em *Python* pode ser feito gratuitamente; a linguagem é mais intuitiva e tem vasta aplicação em análise de dados; é bastante relevante na indústria nos dias atuais.

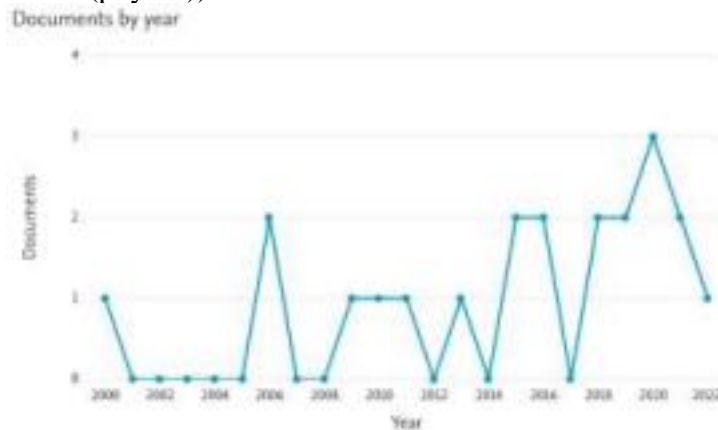
Para Dubois (2007) Um fator importante na utilização da linguagem *Python* de programação entre engenheiros e cientistas é sua sintaxe intuitiva e clara, que torna o código fácil de entender e de manter.

Para justificar a relevância do tema, foi feita a pesquisa na base de dados da Scopus com as palavras chave: *supervisory and system and python*. Nessa pesquisa, obteve-se 21 resultados. E em uma segunda pesquisa no Scopus com as palavras chave: *supervisory and system and python and industrial* foram obtidos 7 resultados.

Nota-se que apesar de uma crescente nos últimos anos, esse assunto ainda é muito pouco explorado, dando uma oportunidade muito grande para se trabalhar em soluções que envolvam sistema supervisorio e a linguagem de programação em Python.

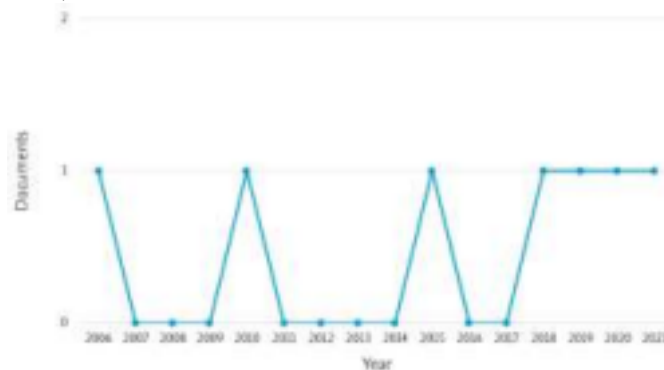
Observa-se isto através da quantidade de documentos acadêmicos por ano, indicado no gráfico da Figura 1 e da Figura 2.

Figura 1 – Gráfico do Scopus utilizando a chave de pesquisa: (supervisory and system) AND TITLE-ABS-KEY (phyton))



Fonte: Scopus (2022).

Figura 2 - Gráfico do Scopus utilizando a chave de pesquisa: (supervisory and system and python and industrial)



Fonte: Scopus (2022).

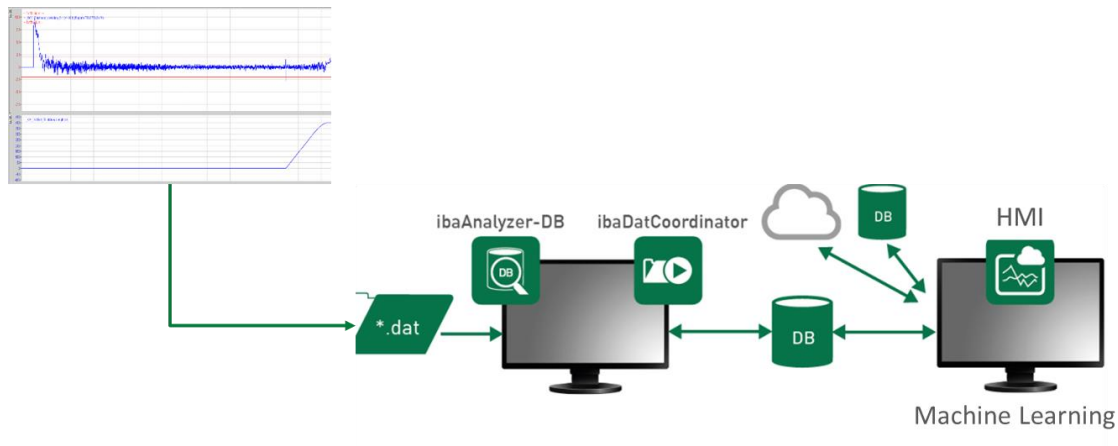
1.3 DELIMITAÇÃO

Um arquivo é criado com os dados crus (*raw*) referentes às características do material produzido, sendo elas, por exemplo, largura e comprimento da bobina de metal. Esse arquivo .dat passa por um tratamento inicial, do qual é possível filtrar os dados a partir de uma máquina equipada com algoritmos de limpeza e tratamento de dados para fins de documentação e reporte interno, alimentando assim, um banco de dados interno. Após esse passo, uma outra máquina (dessa vez equipada com algoritmos de aprendizado de máquina) os utiliza para tomar as decisões referentes ao direcionamento desse produto. Por exemplo: caso haja um desvio na espessura de um produto, esse algoritmo é capaz de decidir se esse material deve ser sucataado ou redirecionado a outro cliente. Essas informações alimentam um novo banco de dados POSTGRESQL em nuvem. O sistema desse trabalho, de natureza de

supervisão, se conectará à essa base e trará as informações ao supervisor a partir de um programa executável de computador, realizando assim a interface máquina-humano (HMI).

A Figura 3 exemplifica como é o fluxo dos dados de produção dessa indústria de laminação de metal.

Figura 3 - Linha do processamento de dados anterior ao sistema de supervisão



Fonte: Machine Learning Python Supervision System for Quality Control (2022).

Esse programa deve conter informações visuais relevantes, como a lista das bobinas de metal produzidas e suas características mais importantes, além de fazer uma redundância visual clara sobre a classificação das bobinas, por exemplo, alterando a cor da linha das bobinas sucata para vermelho, ou as redirecionadas para amarelo e assim por diante.

Em conjunto a isso, o sistema de supervisão precisa atualizar a lista acompanhando a inserção de novos dados de bobinas produzidas em tempo real e permitir que o usuário seja capaz de fazer alterações nos dados que julgar que sejam necessários.

Dessa forma, além da escolha da linguagem de programação em *Python*, também foi necessário decidir como os dados da fabricação seriam armazenados, já que o sistema teria que consultar informações geradas por um software de controle que não é executado em segundo plano, mas de forma externa ao computador do supervisor. Sendo assim, optou-se por armazenar os dados pelo gerenciador *PostgreSQL*.

1.4 OBJETIVO

O objetivo deste trabalho é desenvolver um sistema de supervisão usando linguagem de programa Python, contendo uma interface gráfica que mostra ao usuário todos os dados relevantes das características das bobinas de metal, conectado ao banco de dados no qual os resultados das decisões do algoritmo de aprendizado de máquina estão sendo registrados. Concomitante a isso, o sistema também permite com que sejam feitas alterações nos dados por parte do usuário, no caso do contexto dessa aplicação, o usuário que é um especialista julgue necessário.

Dessa forma, não se elimina completamente a necessidade de um especialista, mas seu trabalho é facilitado ao colocá-lo em uma posição de supervisão e não de tomada de decisão. Diminuindo o *lead time* da produção e aumentando a assertividade da classificação das bobinas de metal, melhorando a satisfação com o cliente e até diminuindo desperdício ao conseguir reclassificar o material que usualmente poderia ser descartado.

1.5 ESTRUTURA DO TRABALHO

O presente trabalho foi dividido em 5 capítulos, sendo o Capítulo 1 mais introdutório, com uma contextualização do problema e uma breve apresentação das ferramentas. O Capítulo 2 apresenta uma explicação introdutória, porém mais detalhada das bibliotecas que foram utilizadas para o desenvolvimento do sistema, enquanto o Capítulo 3 mostra como foi feita a construção do código, quais as dificuldades enfrentadas e alguns comentários. Finalmente, no Capítulo 4 será apresentada a interface gráfica completa e no Capítulo 5, a conclusão desse trabalho.

2 REFERENCIAL TEÓRICO

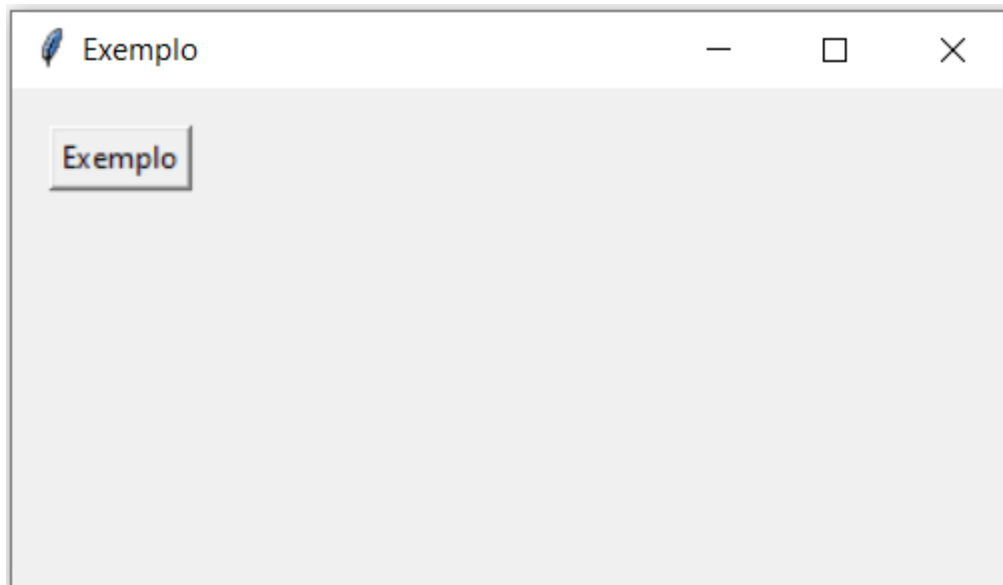
2.1 PYTHON E A BIBLIOTECA TKINTER

Para entender o desenvolvimento da interface gráfica, é necessário ser contextualizado à biblioteca integrada ao espaço de trabalho *Python* chamada *Tkinter*.

De acordo com Grayson (2000), *Tkinter* nada mais é do que um módulo da linguagem *Python*, repleta de ferramentas que facilitam a criação de uma interface de usuário simples, mas com fácil entendimento. Esse módulo acrescenta interfaces orientadas a objetos, ou seja, nele as referências de *widget* são objetos e nós conduzimos os *widgets* usando métodos e seus atributos.

Para entender mais claramente, observa-se a Figura 4.

Figura 4 – Exemplo de janela Tkinter com um botão



Fonte: Autoria própria (2022).

O código de programação referente é apresentado no Quadro 1:

Quadro 1 – Exemplo do código de uma janela com um elemento de botão Tkinter

```
#exemplo

from tkinter import *
from tkinter import ttk

#Definição da Janela
window=Tk()
window.title("Exemplo")
window.geometry("400x200")

#Adição de botões
button_edit=Button(data_frame, text="Exemplo")
button_edit.grid(padx=15, pady=15, sticky=EW)

window.mainloop()
```

Fonte: Autoria própria (2022).

Fica mais claro, agora, entender como é feita a programação utilizando a ferramenta. A janela *window* é um objeto da classe `Tk()`, sendo assim, os métodos de `Tk` podem ser utilizados para alterar as propriedades de *window*, que nada mais é do que a janela da interface.

O título da janela é definido como “Exemplo” e de dimensões 400x200 pixels.

Em seguida, adiciona-se mais um objeto, o botão que herda os métodos da classe *Button* e este é definido com o texto “Exemplo”, deslocado 15 pixels para a direita e para baixo e em seguida recebe a propriedade *Sticky* que faz com que esse botão não perca a sua posição quando a janela for colocada em modo de tela cheia, com a variável *EW* (*East and West*) que permite que o formato do botão mude apenas para a horizontal de forma a mostrar o texto todo dentro do botão.

Enfim, isso tudo é mostrado na tela quando colocado dentro do laço (*loop*) `window.mainloop()`.

Para o sistema de supervisão desenvolvido nesse trabalho, foram necessários vários objetos para os diferentes elementos gráficos do programa e cada um desses elementos necessitaram de características que foram definidas através de classes e métodos. A aplicação de cada um deles será explicada durante a apresentação desse trabalho.

2.2 POSTGRESQL E A BIBLIOTECA PSYCOPG2

O PostgreSQL é uma base de dados muito popular entre os cientistas de dados. E não é à toa, de acordo com STONES (2006), PostgreSQL é muito confiável, estável, gratuito e *open source*, além de, como descrito no nome, suportar a linguagem *query* do SQL e, no ano de lançamento do livro, mostrar ter melhorado de desempenho a cada nova atualização. Nos contextos atuais, com máquinas mais poderosas e tecnologias de leitura de memória mais rápidas, vê-se que a diferença de performance entre PostgreSQL e as outras bases de dados disponíveis torna-se irrelevante, principalmente para a aplicação desse trabalho.

A linguagem *query* SQL nada mais é do que um conjunto de instruções que são feitas por um usuário humano e que facilitam a filtragem dos dados ou edição dos mesmos através do próprio interpretador presente no PostgreSQL.

Para realizar a conexão da aplicação em Python com o PostgreSQL, é necessária uma biblioteca chamada *psycopg2*.

A biblioteca *psycopg2* é uma biblioteca que oferece um *framework* para a conexão com o servidor do PostgreSQL, ou seja, uma certa interface entre o que está acontecendo internamente na sua máquina e o programador. Isso significa que quando você utiliza uma função, ela está na realidade fazendo inúmeras outras operações que você tem controle limitado ou nenhum controle a respeito, porém essa distanciação é muito importante para garantir um código de alto nível, de fácil entendimento e melhor, de fácil aprendizado.

Para essa aplicação, foram utilizadas principalmente as funções de conexão à base, o Quadro 2 exemplifica um código de conexão à uma base do PostgreSQL.

Quadro 2 – Exemplo de conexão à base PostgreSQL

```

conn = psycopg2.connect(dbname="nome da base",
user="usuário", password="senha", host="administrador")

cur = conn.cursor()

cur.execute("SELECT * FROM public.results")
dados=cur.fetchall()

conn.commit()
cur.close()
conn.close()

```

Fonte: A autoria própria (2022).

Percebe-se através desse código que, a variável *conn* recebe um valor de endereço, que é recebido através da função de conexão *psycopg2.connect*. Note que essa função recebe alguns valores em caracteres, esses valores são respectivos à configuração da própria base (feitas no programa PostgreSQL), são elas:

- Nome da base de dados
- Usuário registrado na base
- Senha de segurança para conexão da base
- Host da base (pode ser local ou um ip da internet)

Então é criado um cursor que utiliza esses dados da conexão para, de forma muito simplificada, “apontar” as funções que serão utilizadas para editar a base na direção correta. Note que é por isso que as funções “execute” e “fetchall” são métodos de “cur” (o cursor).

A função *execute* executa um texto que é convertido para um código *query* de SQL que apenas seleciona os dados da tabela publica (por isso o *public*) chamada “results”.

Então é feita uma variável de tabela chamada dados que recebe todos os dados presente nessa tabela através da função “fetchall”.

Finalmente, certifica-se o banco que os dados foram recebidos através da função “commit” e por segurança e boas práticas de programação, interrompe-se a conexão através de *cur.close()* e *conn.close()*. Um conjunto muito parecido de funções será visto durante algumas seções desse trabalho e outras funções da biblioteca *psycopg2* que não são mostradas no Quadro 2 serão descritas na seção de Desenvolvimento.

3 DESENVOLVIMENTO

O sistema apresentado foi desenvolvido de forma modular, ou seja, cada elemento gráfico foi separado e escrito em partes para que a interpretação do código ficasse mais simples, além de favorecer qualquer mudança futura na estrutura gráfica ou até mesmo nos nomes das variáveis presentes no programa, além desses, as funções que realizam as conexões com o banco de dados foram implementadas após comprovado o funcionamento ideal da janela do sistema. O desenvolvimento será explicado seguindo a ordem desses módulos, cada um em detalhes e com suas justificativas para as decisões de cor, tamanho e proporção na janela.

Com uma ideia do que o programa deveria mostrar, as primeiras definições começaram pelas propriedades da janela que seria mostrada para o usuário.

3.1 PROPRIEDADES DA JANELA DA INTERFACE

Decidiu-se inicialmente utilizar um proporção de 4:3 para a janela gráfica pois assim se assemelharia mais com as janelas de aplicações mais antigas de engenharia e possibilitaria a execução em modo janela do programa em praticamente qualquer computador, porém essa ideia foi logo abandonada quando os nomes das variáveis começaram a ser aplicadas na interface, fazendo com que boa parte do texto fosse cortado, então definiu-se uma proporção mais usual aos dias de hoje, a 16:9, em resolução de 1280 x 720 pixels.

O Quadro 3 explicita a parte do código desenvolvida para as propriedades da janela.

Quadro 3 – Definição das propriedades da janela

```
#Definição da Janela
window=Tk()
window.title("Interface de Produção")
window.geometry("1280x720")
```

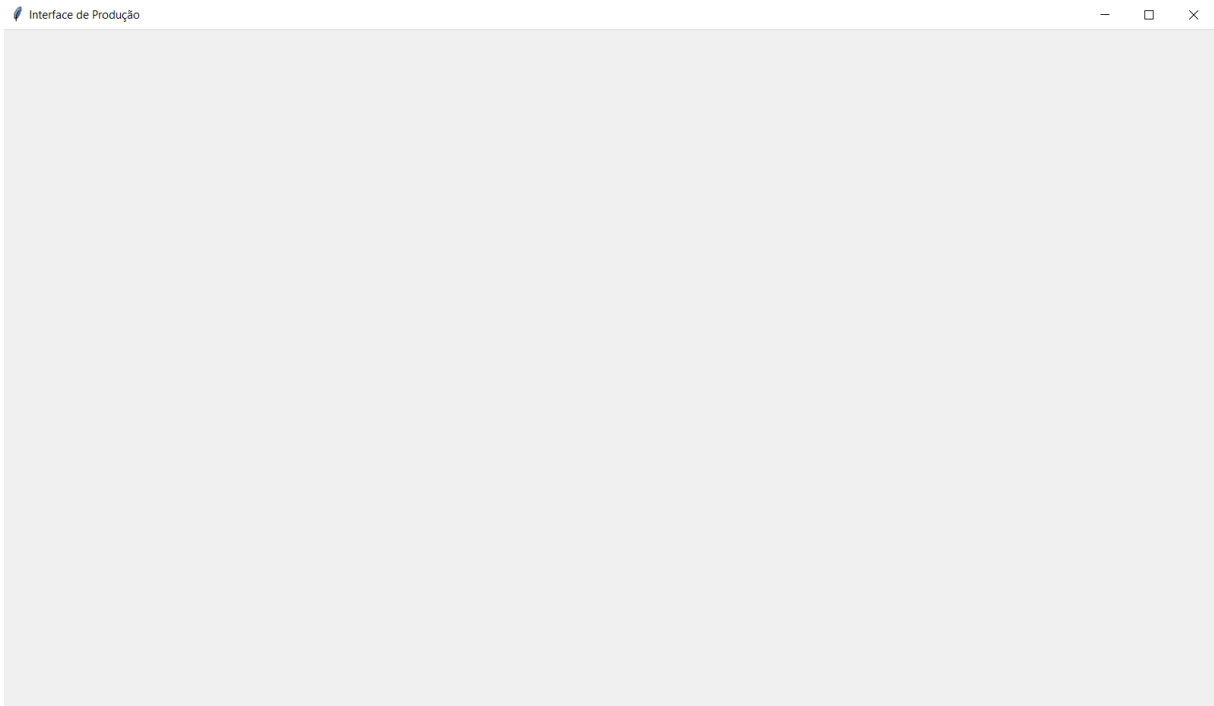
Fonte: Autoria própria (2022).

Observa-se que *window* é criado como um objeto da classe *Tk()* e recebe as propriedades de título, ou seja, o texto que é mostrado na barra superior da janela, igual ao texto “Interface de produção” através da função *title*.

Então, define-se a geometria através do método *geometry* da classe *window*.

A Figura 5 mostra como a janela é exibida enquanto com essas propriedades.

Figura 5 – Janela da Interface de Produção sem elementos gráficos



Fonte: Autoria própria (2022).

3.2 DEFINIÇÃO DA *TREEVIEW*

3.2.1 Definição do *Frame* e propriedades da *Treeview*

Com as propriedades da janela definidas, era necessário colocar um elemento gráfico que fosse capaz de mostrar as propriedades das bobinas de metal e que também permitisse a troca de cor da linha respectiva à bobina.

Inicialmente pensou-se em utilizar um *frame* da janela e adicionar nele vários *grids* para que o programa inserisse de forma programável os dados em cada célula dessa tabela criada dentro do espaço delimitado na janela, mas o desenvolvimento do código foi muito trabalhoso e rapidamente notou-se que esse tipo de abordagem para mostrar os dados não era a mais correta, muito menos a mais eficiente. Dessa maneira, ao consultar a documentação da biblioteca *Tkinter*, descobriu-se um objeto chamado *Treeview*.

A *Treeview* é uma ferramenta da biblioteca gráfica que permite a exibição de itens de forma hierárquica. Quando dados são adicionados nela, ela os aloca preenchendo colunas que serão descritas nas funções de criação de colunas, além de permitir título de *headings* para as

colunas, customização de estilo e cor de linhas e etiquetagem de elementos de linha ou coluna que serão muito úteis ao longo do desenvolvimento desse trabalho.

Já sabendo que essa ferramenta seria utilizada, faz-se a definição da *Treeview* para inseri-la na janela.

Para fazer essa inserção, é necessário a criação de um *frame* (ou moldura) e definir algumas propriedades do *frame* e da lista.

A definição inicial do *frame* e da *Treeview* são vistas no Quadro 4.

Quadro 4 – Definição do *frame* e propriedades iniciais da *Treeview*

```
#Frame no qual entrará o Treeview
tree_frame = Frame(window)
tree_frame.pack(fill="both", expand=1, padx=15, pady=15)

#Treeview
my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended", height=12)
my_tree.pack(fill="both", expand=1)

#Configuração do estilo da treeview
style = ttk.Style()
style.theme_use('default')
style.configure("Treeview", background="white", foreground="black",
fieldbackground="white")
```

Fonte: Autoria própria (2022).

É então definida a classe *tree_frame* e adicionada no espaço da janela, são atribuídas as propriedades de distanciamento de 15 pixels para as bordas e adiciona-se a capacidade de expandir essa moldura quando a janela é colocada em modo de tela cheia ou tem sua proporção redimensionada pelo mouse.

Logo após, é definida a *treeview* com o nome *my_tree* e ela é alocada para o espaço da *tree_frame* que foi definida anteriormente. Atribui-se uma altura de 12 (proporção que será utilizada para delimitar o espaço na janela quando outras frames forem adicionadas), aplica-se novamente a capacidade de expandir junto com a janela e coloca-se uma propriedade de *yscrollcommand* que permitirá com que, quando vários elementos estiverem alocados à essa lista, seja possível utilizar uma barra lateral para mudar os elementos disponíveis em tela, sem com que as informações anteriores sejam perdidas ou que as linhas mudem de tamanho para colocar todos os elementos dentro de uma *frame* só.

Por fim, é criado o método *style* como objeto de *ttk.Style()* para definir o estilo da *Treeview*. É colocado o tema padrão da própria biblioteca e é configurado a cor branca como fundo da lista e preto como a cor entre o cabeçalho e a lista de elementos.

3.2.2 Scrollbar da Treeview

Para que seja possível fazer com que uma barra lateral seja capaz de mudar o foco da lista em tela, é necessário definir uma função que faça isso. Por conveniência, já existem métodos na própria biblioteca *Tkinter* capazes de definir essa função de forma a utilizar comandos de mais alto nível.

É definido uma classe chamada *tree_scroll*, com o método *Scrollbar* e atribuído ao *Frame* no qual está definida a lista.

Essa barra é colocada à direita e com a propriedade de preencher verticalmente a moldura, para imitar o que se vê normalmente em páginas da internet, aonde a barra é colocada à direita e preenche o eixo vertical da página.

Esse código teve que ser escrito anterior à definição da *Treeview* para que programa conseguisse interpretar a função atribuída à propriedade *yscrollcommand*. Logo após isso, era necessário configurar essa barra após a definição da lista, pois ela seria atribuída à classe *my_tree*, ou seja, a lista tinha que existir para que se pudesse configurar a *scroll bar* de modo a arrastar seus elementos na tela.

O código está presente no Quadro 5.

Quadro 5 – Definição e atribuição da *Scrollbar* na *Treeview*

```
#Scrollbar da Treeview
tree_scroll = Scrollbar(tree_frame) #Anterior à definição da Treeview
tree_scroll.pack(side="right", fill="y")

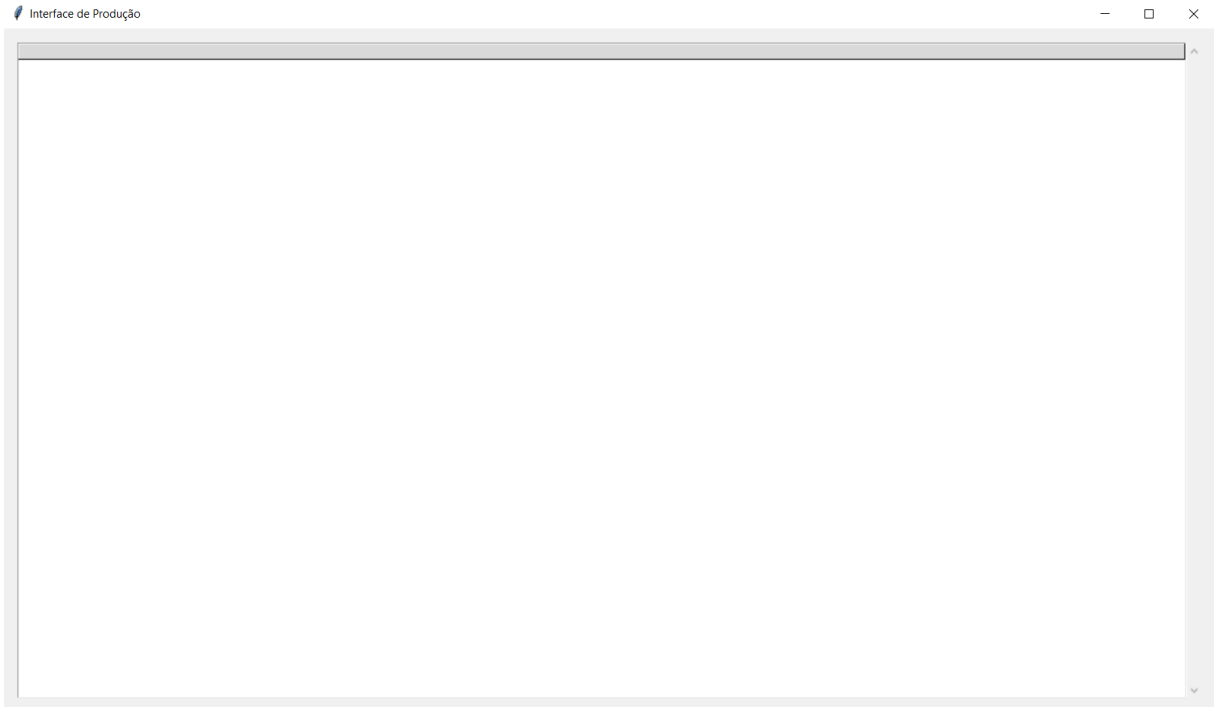
#Treeview
my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set, selectmode="extended",
height=12)
my_tree.pack(fill="both", expand=1)

#Configuração da Scrollbar
tree_scroll.config(command=my_tree.yview) #Deve ser definido após a criação da classe my_tree
```

Fonte: Autoria própria (2022).

Dessa forma, a janela deve mostrar a moldura da lista, o espaço da lista e uma barra lateral. Um exemplo de como pode-se visualizar essa janela está presente na Figura 6.

Figura 6 – Janela com a *Frame* da *Treeview* e *Scrollbar*.



Fonte: Autoria própria (2022).

3.2.3 Elementos da *Treeview*

Com a moldura da lista pronta, é necessário definir os cabeçalhos das colunas que receberão os dados referentes à produção das bobinas de metal. Para os fins desse trabalho foram utilizadas 10 variáveis, sendo elas:

- Ot
- Espessura
- Largura
- Comprimento Final
- Descarte Espessura
- Descarte Temperatura
- Descarte Final
- Resultado Final
- Espessura do Descarte
- Resultado Modelo

Elas foram atribuídas, respectivamente, à *Strings*, contando de *String1* até *String10*, e então essas variáveis foram definidas para um vetor, usando a linha de código presente no Quadro 6.

Quadro 6 – Definição das colunas da *Treeview*

```
my_tree['columns'] = (String1, String2, String3, String4, String5,
String6, String7, String8, String9, String10)
```

Fonte: Autoria própria (2022).

Dessa maneira, agora era necessário a formatação das colunas e a criação de seus respectivos cabeçalhos na *Treeview*. Foi escrito um código conforme o Quadro 7.

Quadro 7 – Formatação das colunas da *Treeview*

```
my_tree.column("#0", width=0, stretch=NO)
my_tree.column(String1, anchor=CENTER, width=100)
my_tree.column(String2, anchor=CENTER, width=100)
my_tree.column(String3, anchor=CENTER, width=100)
my_tree.column(String4, anchor=CENTER, width=100)
my_tree.column(String5, anchor=CENTER, width=100)
my_tree.column(String6, anchor=CENTER, width=100)
my_tree.column(String7, anchor=CENTER, width=100)
my_tree.column(String8, anchor=CENTER, width=100)
my_tree.column(String9, anchor=CENTER, width=100)
my_tree.column(String10, anchor=CENTER, width=100)
```

Fonte: Autoria própria (2022).

As colunas foram formatadas utilizando a função *my_tree.column()* com as propriedades de suas *Strings* como o título da coluna, com a posição centralizada e com largura de 100 pixels. Note que na primeira linha não é atribuído nenhuma *String* de título, nem é definido nenhuma largura. Isso é porque os métodos da *Treeview*, por padrão, definem que a primeira coluna é a coluna de ID, ou identificação, das linhas. Dessa forma que o programa consegue identificar quais são as linhas e aonde alocar os dados quando se trabalha com esse tipo de lista. Neste trabalho, o *ot* é o identificador utilizado para fazer as operações das funções do programa, no entanto, esse ID sempre estará disponível e pode ser utilizado, caso necessário. Como não é interessante mostrar esse ID na coluna, então esconde-se a coluna 0.

Depois que foram definidas as colunas, são criados os *Headings*, ou cabeçalhos, para serem mostrados na tela. Essa parte do programa foi escrita conforme o Quadro 8.

Quadro 8 – Definição dos cabeçalhos da *Treeview*

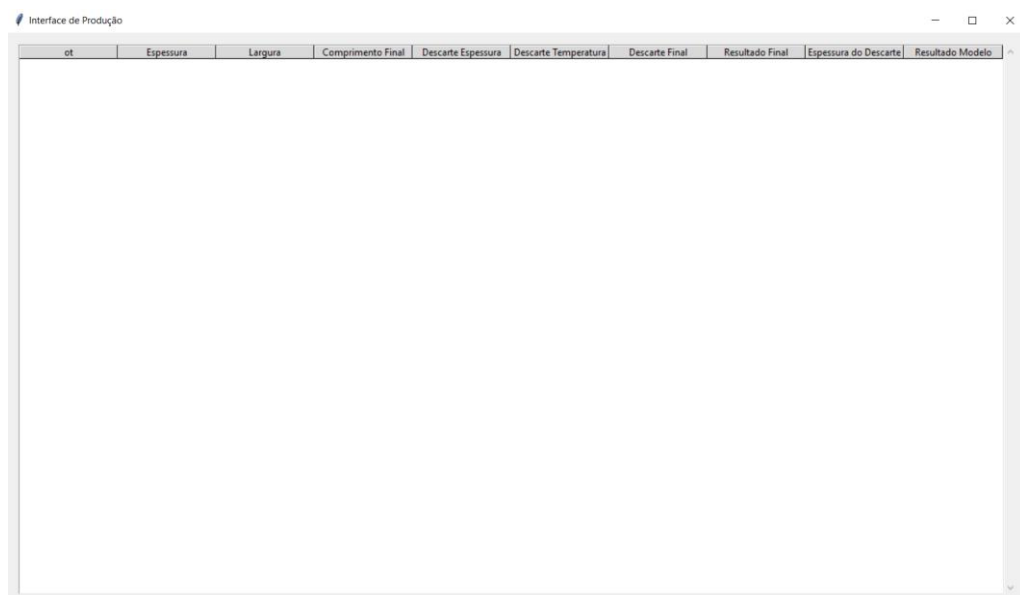
```
#Criação dos Headings
my_tree.heading("#0", text="", anchor=CENTER)
my_tree.heading(String1, text=String1, anchor=CENTER)
my_tree.heading(String2, text=String2, anchor=CENTER)
my_tree.heading(String3, text=String3, anchor=CENTER)
my_tree.heading(String4, text=String4, anchor=CENTER)
my_tree.heading(String5, text=String5, anchor=CENTER)
my_tree.heading(String6, text=String6, anchor=CENTER)
my_tree.heading(String7, text=String7, anchor=CENTER)
my_tree.heading(String8, text=String8, anchor=CENTER)
my_tree.heading(String9, text=String9, anchor=CENTER)
my_tree.heading(String10, text=String10, anchor=CENTER)
```

Fonte: Autoria própria (2022).

É utilizado o método *heading* da classe da já definida *my_tree* e são atribuídos o título, que precisa ser igual ao título da coluna para que os dados sejam correlacionados, o texto apresentado no cabeçalho, que neste caso é igual ao título por questão de conveniência e o texto é alocado no centro do espaço de *Heading*.

A janela com esses elementos é apresentada na Figura 7.

Figura 7 – Janela com *Frame* e *Treeview* definidos sem dados



Fonte: Autoria própria (2022).

Após a construção dos cabeçalhos, era necessário inserir os dados para serem mostrados na *Treeview*.

Primeiramente foi construída uma variável chamada “dados” em formato de matriz, para armazenar dados aleatórios, mas que seriam muito úteis para a visualização. Essa matriz foi declarada com os elementos escritos em código, um a um, e não é muito relevante para a explicação do funcionamento da lógica, portanto, por ocupar um grande pedaço do código e que será substituído por funções de conexão ao banco de dados, decidiu-se por não incluir na explicação. Após isso, os dados foram colocados na *Treeview* através de um laço de *for* conforme o Quadro 9.

Nota-se que as variáveis *cont*, *contse*, *contre*, *contsu* foram definidas como variáveis globais, isso é porque elas ajudarão a definir a quantidade total de bobinas e a quantidade delas que foram sucataadas, seguiram processo ou foram redirecionadas. Além delas, também foram definidas as variáveis *seguir* e *redirecionar*, com a informação de texto referente ao resultado do processo.

O *for* é definido para a variável *record* incrementar até a quantidade de itens dentro da tabela *dados*. Perceba que não mais do que essa definição foi necessária para aplicação porque, por padrão, o interpretador do *Python* já entende de forma mais abstrata.

Durante todo o *loop*, passa-se por um teste condicional:

- Se a informação da coluna 7, que contém o resultado, for *Seguir*, então:
 - Adiciona-se a informação daquela linha em todas as colunas de 0 até 9 a partir da variável *values* do método *insert* da classe *my_tree*. De forma geral, sem definir o parentesco, porque não há a necessidade de “filhos” ou informações complementares da mesma coluna, começando do endereço 0, com o id referente à quantidade de bobinas (começando também em 0) e sem texto.
 - É colocada uma etiqueta, ou *tag*, de acordo com o resultado dado para àquela bobina
 - Acrescenta-se 1 na quantidade total de bobinas
 - Acrescenta-se 1 na quantidade de bobinas que seguiram processo

Quadro 9 – Inserção dos dados na *Treeview*

```

global cont, contse, contre, contsu

cont=0
contse=0
contre=0
contsu=0

#Adicionar dados na Treeview
def ins_dados():

    seguir="Seguir Processo"
    redirecionar="Redirecionar"

    for record in dados:
        if record[7].find(seguir) != -1:
            my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0],
record[1], record[2], record[3], record[4], record[5], record[6], record[7], record[8],
record[9]), tags=('seguir',))
            cont += 1
            contse += 1

        elif record[7].find(redirecionar) != -1:
            my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0],
record[1], record[2], record[3], record[4], record[5], record[6], record[7], record[8],
record[9]), tags=('redirecionado',))
            cont += 1
            contre += 1
        else:
            my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0],
record[1], record[2], record[3], record[4], record[5], record[6], record[7], record[8],
record[9]), tags=('sucitado',))
            cont += 1
            contsu += 1

#Limpar a Treeview
def clear_treeview():
    for item in my_tree.get_children():
        my_tree.delete(item)

```

Fonte: Autoria própria (2022).

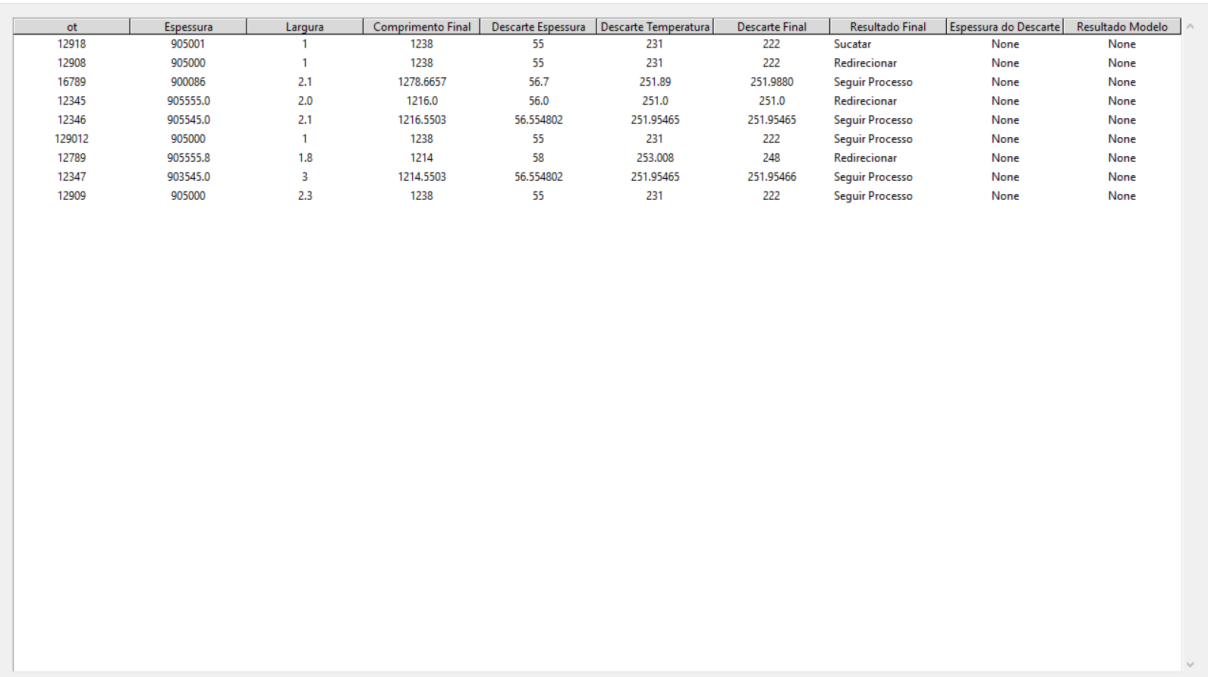
Essa mesma lógica é aplicada para os resultados de redirecionamento e sucatagem. Como não existem mais do que essas 3 opções de resultado, não foi necessário criar uma variável para aplicação da *tag* sucitado pois entende-se que, se o material não seguiu processo

e nem foi redirecionado, logo, foi sucitado. Por fim, a função `ins_dados()` é chamada antes do laço da janela.

Por fim, é definida uma função que destrói a *Treeview*, utilizando-se de um outro laço *for*, no qual *item* começa em 1 e acrescenta de 1 até o número total de itens da lista, deletando-os um por um através do método *delete*. Essa função será aplicada toda vez que for necessário atualizar as informações da *Treeview*, seja pela edição do usuário ou pela inserção de uma nova bobina no banco de dados.

A *Treeview* com os dados inseridos pode ser vista na Figura 8.

Figura 8 – *Treeview* com dados



ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Fonte: Autoria própria (2022).

Então são definidas as cores através da inserção do código presente no Quadro 10 na função descrita anteriormente.

Quadro 10 – Definir as cores da linha pela etiqueta

```
#Adicionando Etiquetas (para mudar a cor da linha)
my_tree.tag_configure('seguir', background="lightgreen")
my_tree.tag_configure('redirecionado', background="yellow")
my_tree.tag_configure('sucitado', background="red")
```

Fonte: Autoria própria (2022).

Utiliza-se o método *tag_configure* para configurar as etiquetas que foram criadas anteriormente pelo método de inserção dos dados na *Treeview* e define-se o plano de fundo ou, *background*, para as cores consideradas mais adequadas para os resultados.

Dessa maneira, finalizou-se a *Treeview* e a janela ficou conforme a Figura 9.

Figura 9 – *Treeview* completa com os dados

Interface de Produção

ct	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Fonte: Autoria própria (2022).

3.3 SEÇÃO DE CONTAGEM DE BOBINAS

De forma muito similar à seção da janela descrita pelos tópicos anteriores, foi criada uma moldura, ou *frame*, chamada *totalb_frame* para uma parte central da interface que mostra o total de bobinas produzidas, assim como o total de bobinas sucata, redirecionadas e que irão seguir ao cliente.

Assim, foi desenvolvido o código que é descrito no Quadro 11.

Quadro 11 – Código da seção de contagem de bobinas

```

#Frame total de bobinas
totalb_frame = LabelFrame(window, text="Totais da Produção")
totalb_frame.pack(fill="x", expand="yes", padx=15)

#Atualizar o total de bobinas no Frame
def totalb_update():
    totp_label=Label(totalb_frame, text=("Total Produzido: {}".format(cont)))
    totp_label.grid(row=0, column=0, padx=15, pady=15, sticky = "nsew")
    totc_label=Label(totalb_frame, text=("Total em Cliente: {}".format(contse)))
    totc_label.grid(row=0, column=1, padx=15, pady=15, sticky = "nsew")
    totred_label=Label(totalb_frame, text=("Total Redirecionado:
{}".format(contre)))
    totred_label.grid(row=0, column=2, padx=15, pady=15, sticky = "nsew")
    totsuc_label=Label(totalb_frame, text=("Total Sucatado: {}".format(consu)))
    totsuc_label.grid(row=0, column=3, padx=15, pady=15, sticky = "nsew")

#Deletar o total de bobinas no frame
def totalb_delete():
    for i in totalb_frame.winfo_children():
        i.destroy()

```

Fonte: Autoria própria (2022).

Após da definição do *Frame*, com o texto “Totais da produção” aparente na parte superior moldura através do método *LabelFrame*, no método *pack* foi definido a distância de 15 pixels da seção superior, que no caso é a *Treeview*, é colocada a propriedade de *fill* para que a moldura se estenda para preencher horizontalmente o espaço da janela e *expand* para que as proporções sejam alteradas conforme o redimensionamento da janela.

Definiu-se uma função chamada *total_update* para construir a parte de texto que indica os números totais da produção. Essa função será útil quando for necessário atualizar as informações da janela de acordo com as informações editadas pelo usuário ou quando o banco de dados receber novas informações.

Para cada texto é declarada um novo rótulo, ou *label*, que divide o espaço da moldura de acordo com as propriedades inseridas em seus métodos. É utilizado o método *Label*, declarando as classes com os nomes de acordo com as informações que elas carregarão: *totp_label* para o total de bobinas produzidas, *totc_label* para o total direcionado ao cliente, *totred_label* para redirecionado e por fim *totsuc_label* para o total sucatado.

Esses rótulos são todos definidos na moldura *total_frame* e em cada um deles o texto exibido segue a sua informação. Note que após os dois pontos, são inseridas duas chaves, elas

indicam um lugar no texto que receberá a informação de uma variável global. Essa variável é então “chamada” pelo método *format* e dentro dos parênteses são colocadas as variáveis que foram calculadas conforme descrito no tópico anterior do trabalho.

Então, para definir a posição correta dessas informações na janela gráfica é utilizado o *grid*, define-se que a linha para todos eles é a linha 0 pois toda a informação ficará na horizontal, acrescenta-se de 0 até 3 nas colunas, para colocar as informações da direita para a esquerda, aplica-se 15 pixels de distância acima e à laterais e por fim, a propriedade *sticky* com a variável *nsew* (para *North*, *South*, *East* e *West*) para que quando a janela seja redimensionada, o *grid* mantenha a proporção para todas as direções.

Por fim, é definida uma função para destruir essa seção, chamado de *total_delete*, que se utiliza de um laço *for* com a variável *i* de acordo com todos os itens (nesse caso, os rótulos) presentes na classe *total_frame* e os deleta. Essa função é muito útil para a atualização das informações na janela, pois para evitar que essa moldura seja sobreposta por outra durante a execução do código, é necessário destruir a moldura e construí-la novamente. Essa operação é tão rápida, que a sensação é de que apenas o texto com as informações das bobinas mudou.

A janela, com essa nova moldura é vista conforme a Figura 10.

Figura 10 – Janela com *Treeview* e informação da produção

Interface de Produção

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Fonte: Autoria própria (2022).

Note que, conforme explicado anteriormente, como a *Treewiew* foi construída com a propriedade *height* igual a 12, o próprio interpretador fez uma proporção automática que entende 12 como uma porção da tela. Como não foi definida essa propriedade para a moldura das bobinas, a janela foi construída de acordo com uma proporção de 12 para 20 no espaço da janela, com a *Treewiew* ocupando um pouco mais que metade da tela.

3.4 ÁREA DE VERIFICAÇÃO DOS DADOS

Com todas essas informações na tela, agora o supervisor tinha que ter como editar os dados que lhes são apresentados, sendo assim, foi desenvolvida a área para verificação e edição dos dados das bobinas.

De forma similar ao tópico anterior, também foi construída uma moldura. O código é explicitado conforme o Quadro 12.

Quadro 12 – Definição da moldura e propriedades da área de verificação dos dados

```
#Definição do Frame e etiqueta para a área de verificação dos dados
dados_frame = LabelFrame(window, text="Verificação dos dados")
dados_frame.pack(fill="x", side="bottom", expand="yes", padx=15, pady=10)
dados_frame.columnconfigure(0, weight=1)
dados_frame.columnconfigure(1, weight=1)
dados_frame.columnconfigure(2, weight=1)
dados_frame.columnconfigure(3, weight=1)
dados_frame.columnconfigure(4, weight=1)
dados_frame.columnconfigure(5, weight=1)
dados_frame.columnconfigure(6, weight=1)
dados_frame.columnconfigure(7, weight=1)
dados_frame.columnconfigure(8, weight=1)
dados_frame.columnconfigure(9, weight=1)
```

Fonte: Autoria própria (2022).

Foi declarada a classe *dados_frame* para a definição da moldura que recebe os dados, é definido o título “Verificação dos dados” para aparecer no topo da moldura. As propriedades são quase idênticas às propriedades da moldura do tópico anterior, com a exceção da distância vertical que agora é de 10 pixels.

Para essa aplicação será necessário colocar um texto acima de uma caixa de texto. Portanto na mesma coluna, entrarão um rótulo e uma *entry box*. Dessa forma, é utilizado o método *columnconfigure* pra configurar a *Frame* em colunas, que permite que sejam colocados dois elementos gráficos num mesmo espaço da moldura. As colunas são numeradas

de 0 até 9 e todas elas têm o mesmo “peso”, o que significa que o espaço da moldura reservado a elas é igual.

O Quadro 13 contém uma parte do código referente à construção desses itens. Como as definições a seguir foram feitas para todos os itens, enumerados de 0 até 9, o Quadro 13 apenas contém o exemplo de 2 elementos de forma a diminuir a redundância da explicação. Note que isso se repete, porém as informações de texto mudam conforme as 10 variáveis de fabricação da bobina. Para uma visão mais detalhada do código, consulte o Apêndice A.

Quadro 13 – Espaço de verificação dos dados

```
#Exibição (cima) e inserção de dados (baixo)
d1_label=Label(dados_frame, text=String1)
d1_label.grid(row=0, column=0, padx=15, pady=15)
d1_entry=Entry(dados_frame)
d1_entry.grid(row=1, column=0, padx=15, pady=5)

d2_label=Label(dados_frame, text=String2)
d2_label.grid(row=0, column=1, padx=15, pady=15)
d2_entry=Entry(dados_frame)
d2_entry.grid(row=1, column=1, padx=15, pady=5)
```

Fonte: Autoria própria (2022).

É declarada a classe *d1_label* que utiliza do método *Label* para entrar na *Frame* e com o texto de acordo com a variável *String1*, que contém o texto “ot”. O método *grid* é chamado para definir que esse rótulo com o texto será colocado na linha 0 (mais acima) e na coluna 0 (mais à esquerda) e com distância de 15 pixels para os dois lados.

A classe *d1_entry* é uma classe de caixa de entrada de texto, muito utilizada para adquirir dados de um usuário do programa e essa classe é definida através do método *Entry*, definida com a propriedade da moldura. De forma similar à *label*, a *entry* também pode ser chamada pelo método *grid* e ela é configurada como localizada abaixo ao rótulo, portanto linha 1 (abaixo da linha 0) e na mesma coluna.

Note que esse padrão é igual para as informações e caixa de texto do segundo dado, chamados como *d2_label* e *d2_entry*. A única diferença é a posição deles na coluna, agora definidos como na coluna 1. Isso se repete para todos os outros dados, até o décimo dado que recebe o nome de “Resultado Modelo”.

Por fim, é definido um botão, com o texto “Editar” ao lado das caixas de texto. O Quadro 14 mostra o código de definição do botão.

Quadro 14 – Definição do botão Editar

```
#Adição de botões
button_edit=Button(dados_frame, text="Editar", command=edit_record)
button_edit.grid(row=1, column=10, padx=15, pady=15, sticky=EW)
```

Fonte: Autoria própria (2022).

O botão é declarado como a classe *button_edit* através do método *Button*, colocado à moldura e com o comando *edit_record* que será explicado na seção das funções do programa.

Por fim, a Figura 11 contém um exemplo da janela com todos os elementos gráficos.

Figura 11 – Janela com todos os elementos gráficos

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Verificação dos dados

ot	Espessura	Largura	Comprimento Fin	Descarte Espessur	Descarte Temperati	Descarte Final	Resultado Final	espessura do Descart	Resultado Model
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Editar

Fonte: Autoria própria (2022).

3.5 FUNÇÕES DO PROGRAMA

3.5.1 Seleção dos dados

Com os elementos gráficos na Janela, era necessário definir as funções do programa. Começou-se por implantar a função de seleção de linha, que deveria mudar a linha da cor

selecionada pelo clique do *mouse* e colocar as informações nas caixas de texto para edição. Assim, foi desenvolvido um código conforme o Quadro 15.

Quadro 15 – Função de seleção de dados

```
#Clicar e selecionar dados
#Mudar Cor da seleção
style.map('Treeview', background=[('selected', "#313BB4")])

#Variável global de seleção da linha
selected=""
def select_data(e):
    #Limpar as caixas de entrada
    d1_entry.delete(0, END)
    d2_entry.delete(0, END)
    d3_entry.delete(0, END)
    d4_entry.delete(0, END)
    d5_entry.delete(0, END)
    d6_entry.delete(0, END)
    d7_entry.delete(0, END)
    d8_entry.delete(0, END)
    d9_entry.delete(0, END)
    d10_entry.delete(0, END)

    global selected
    selected=my_tree.focus()
    values=my_tree.item(selected, 'values')

    #Jogar informação na entry box
    d1_entry.insert(0, values[0])
    d2_entry.insert(0, values[1])
    d3_entry.insert(0, values[2])
    d4_entry.insert(0, values[3])
    d5_entry.insert(0, values[4])
    d6_entry.insert(0, values[5])
    d7_entry.insert(0, values[6])
    d8_entry.insert(0, values[7])
    d9_entry.insert(0, values[8])
    d10_entry.insert(0, values[9])

#Acoplar Treeview na função de seleção
my_tree.bind("<ButtonRelease-1>", select_data)
```

Fonte: Autoria própria (2022).

Para que a linha selecionada fique evidente, foi utilizado o método *map* da classe padrão *style* para configurar o plano de fundo da linha da *Treeview* selecionada através do

“endereço”, *index*, que foi definido como a variável *selected* e com o código de cor hexadecimal 313BB4, que é uma cor azul escuro.

Essa variável *selected* é definida como uma variável global pois carregará a informação do endereço da linha selecionada e isso será muito importante quando a janela for atualizada. Como os elementos da *Treeview* são deletados e inseridos novamente, a posição da informação na linha seria perdida se não fosse salva em uma variável global. Ao utilizar o programa, a linha se mantém selecionada mesmo quando há uma atualização da janela e isso é porque o método *focus* é chamado utilizando a informação dessa variável.

Declara-se a função *select_data* com uma variável *e* que não é declarada, isso é porque por padrão, o interpretador *Python* entende *e* como uma variável de evento. O evento em questão é acoplado à *Treeview* usando a propriedade *ButtonRelease-1* que, na biblioteca do *Tkinter* é definido como o evento de soltar do botão esquerdo do *mouse*.

Ao se clicar na linha, as informações de todas as caixas de entrada são limpas, o endereço da nova linha selecionada é salvo e os valores dos dados na nova linha são inseridos um a um em cada caixa de entrada através dos valores armazenados no vetor *values*, que os recebe através do método *item* que retorna os valores de cada coluna de uma respectiva linha, que nesse caso são os dados de uma bobina.

3.5.2 Edição dos dados

Com os dados selecionados e mostrados em tela, agora é necessário que o usuário seja capaz de editá-los. Para isso, foi desenvolvida a função *edit_record* conforme mostrado no Quadro 16.

Como no tópico anterior, o método *focus* retorna o valor do *index* da linha e o armazena em *selected*, em seguida o método *item* insere nos dados referentes à essa linha de endereço os valores digitados em cada uma das caixas de entrada. Então, é realizada a conexão com o banco de dados, que será descrita mais profundamente adiante no trabalho e então as informações das caixas são apagadas e a função *update* é chamada.

Quadro 16 – Função de edição de dados

```

def edit_record():
    #Edição dos dados mostrados na Treeview
    selected=my_tree.focus()
    my_tree.item(selected, text="", values=(d1_entry.get(),
                                           d2_entry.get(),
                                           d3_entry.get(),
                                           d4_entry.get(),
                                           d5_entry.get(),
                                           d6_entry.get(),
                                           d7_entry.get(),
                                           d8_entry.get(),
                                           d9_entry.get(),
                                           d10_entry.get(),
                                           ))

    #Realiza conexão

    #Limpar as caixas de entrada
    d1_entry.delete(0, END)
    d2_entry.delete(0, END)
    d3_entry.delete(0, END)
    d4_entry.delete(0, END)
    d5_entry.delete(0, END)
    d6_entry.delete(0, END)
    d7_entry.delete(0, END)
    d8_entry.delete(0, END)
    d9_entry.delete(0, END)
    d10_entry.delete(0, END)

    update()

```

Fonte: Autoria própria (2022).

3.5.3 Funções de atualização da Janela

Para garantir que os dados mostrados na janela sejam sempre os mais atuais, foram definidas algumas funções de atualização da janela. Para essa aplicação, os dois elementos que contém dados são a *Treeview* e a moldura que mostra a quantidade de bobinas produzidas, sendo assim, como explicado anteriormente, essas foram as duas seções da janela que receberam funções de criação e destruição de item gráfico. Aproveitando essas funções, é declarada a função *update* conforme o Quadro 17.

Quadro 17 – Função *update*

```
#Atualizar tudo
def update():
    clear_treeview()
    ins_dados()
    totalb_delete()
    totalb_update()

#Manter linha selecionada
my_tree.selection_set(selected)

update_loop()
```

Fonte: Autoria própria (2022).

Como pode se perceber, a função primeiramente deleta todos os elementos da *Treeview* e coloca novamente, caso haja algum novo item, na nova inserção de dados na lista, ele será considerado pelo laço de *for* explicado no item 3.2.3. Após isso, toda a moldura dos dados de produção é destruída e construída novamente utilizando os dados atualizados de produção conforme o mesmo laço. Mantém-se a linha selecionada ao salvar o endereço da linha e inseri-la novamente através do método *selection_set* e então o programa é redirecionado à função *update_loop*, que é explicitada no Quadro 18.

Quadro 18 – Função *update_loop*

```
#Loop
def update_loop():
    window.after(5000, update)
```

Fonte: Autoria própria (2022).

Essa função consiste de uma única linha, que chama a função *update* a cada 5000 milissegundos através do método *after* para a classe *window*, que é a janela. Dessa maneira, o programa se mantém sempre em um ciclo entre a função *update* que direciona o fluxo para *update_loop*, que por sua vez, redireciona à *update* após 5 segundos. Dessa maneira, a cada 5 segundos, a janela toda é atualizada automaticamente, de forma a detectar mudanças no banco de dados. Caso não haja mudança, mas haja edição pelo usuário, ocorrerá o evento explicado no item anterior e assim, o a janela será sempre atualizada.

3.6 CONEXÃO COM O BANCO DE DADOS

Para essa parte do trabalho, foi necessário instalar o programa pgAdmin 4, localizado no site www.postgresql.org. Após a instalação, foi necessário configurar o servidor que seria utilizado para realizar a conexão.

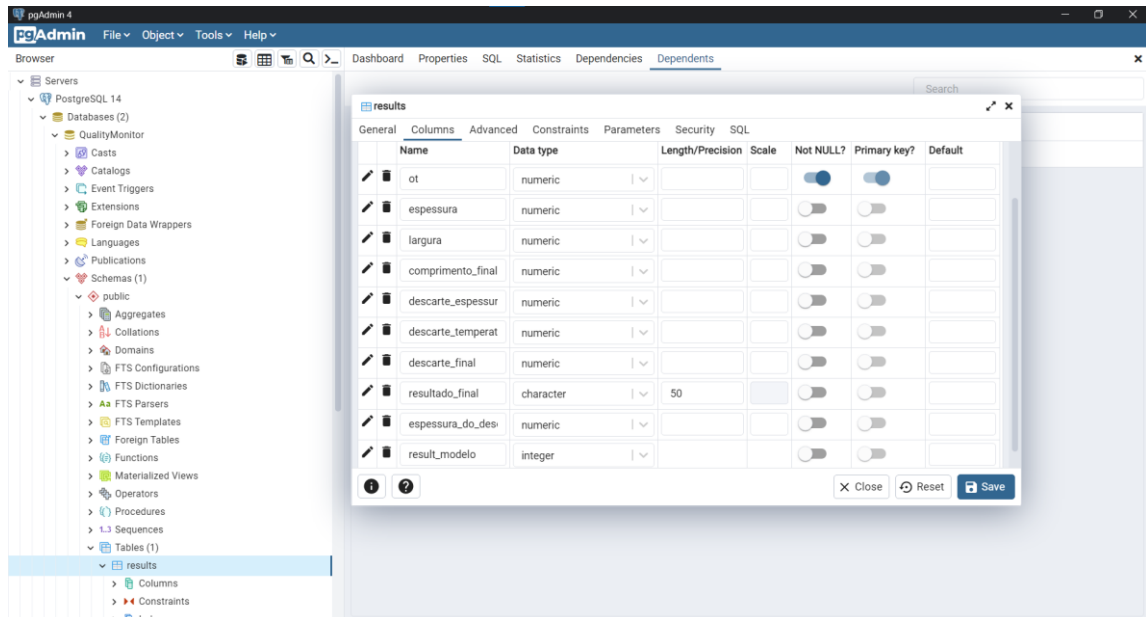
De início, o programa dá uma configuração padrão de nome de usuário e é necessária configuração de uma senha. Para esse programa, foi utilizado o nome padrão, que é admin e a senha foi definida como a senha 123. Não há problemas de segurança por duas razões: A primeira é que os dados apresentados nesse trabalho não são reais, pois não se obteve autorização para expor dados de produção e a segunda razão é que o servidor para essa aplicação será um servidor local, armazenada na própria máquina e sem conexão com a rede mundial de computadores.

Logo após as configurações iniciais, deparou-se com uma interface com várias opções. No canto esquerdo da tela havia um símbolo da cabeça de um elefante com o nome PostgreSQL 14 e uma seta apontando para baixo. Clicou-se nessa seta e procurou-se a seção *Databases*. Se criou um banco de dados com o botão direito nessa seção e em *Database*, após *Create*, colocou o nome *QualityMonitor*. Com o botão esquerdo no banco de dados, procurou-se por *Schemas* e em *Schemas*, clicou-se com o botão direito em *Tables* e, para essa aplicação, criou-se a tabela chamada *results*.

Essa tabela foi criada como uma tabela pública e suas colunas foram criadas de forma a seguir a posição, o nome e o tipo de informação das colunas que foram definidas logo na criação da *Treeview*. A tela de configuração da tabela *results* pode ser vista na Figura 12.

A variável *ot* foi definida como a chave primeira (*primary key*), ou seja, ela será o principal indicador de ID nessa tabela. Todas as operações de edição e criação de dados, tais como também a ordenação em ordem crescente ou decrescente seguirá apenas o valor de *ot*.

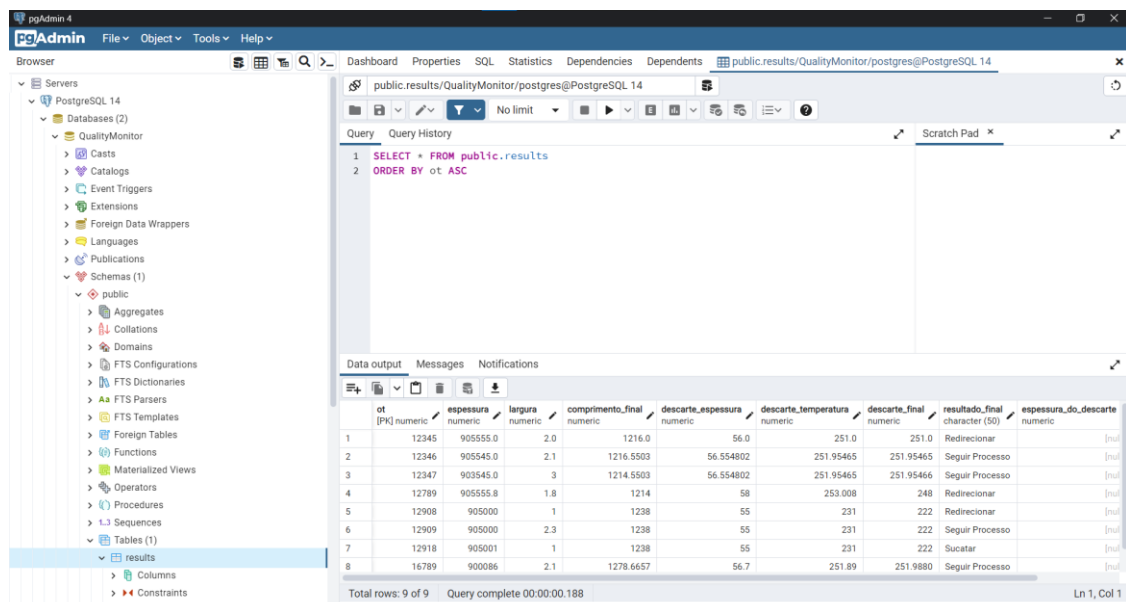
Figura 12 – Tela de configuração da tabela pública de dados *results*



Fonte: Autoria própria (2022).

Foram inseridos os mesmos dados fictícios colocados na *Treeview* no banco de dados PostgreSQL. Para verificar os dados, procura-se o ícone de uma tabela no canto superior direito e clica-se nele. A tabela *results* com os dados das bobinas é vista na Figura 13.

Figura 13 – Tabela *results* com os dados



Fonte: Autoria própria (2022).

Com o banco de dados criado, agora era necessário adequar o código e suas funções para realizar conexão ao banco, buscar e editar os dados presentes nele.

O Quadro 19 mostra o código de configuração da base de dados para o programa e a importação da biblioteca de manipulação de dados do *PostgreSQL* em *Python*.

Quadro 19 – Configuração de conexão ao banco de dados PostgreSQL

```
#Definição das variáveis de conexão ao banco de dados
DB_HOST="localhost"
DB_NAME="QualityMonitor"
DB_USER="postgres"
DB_PASSWORD="123"

#Importando as bibliotecas
import psycopg2
```

Fonte: Autoria própria (2022).

Após a definição das variáveis globais de conexão, foi necessário adaptar as outras duas funções do programa que necessitavam de conexão ao banco para funcionarem corretamente, a começar pela *Treeview*. O Quadro 20 mostra a adaptação do código da construção da *Treeview*.

Na função *ins_dados* são adicionadas as linhas de conexão ao banco. Como explicado na seção de introdução, *cur* é o cursor extraído através de *conn*, uma variável que recebe os dados de conexão através da função *psycopg2.connect*. Para extrair os dados da tabela que foi criada, é usado o método *execute* da classe do cursor *cur* e nele é escrito um texto. Esse texto, na verdade, é executado como um código de *query* do SQL, portanto, deve ser escrito conforme o interpretador do PostgreSQL, por isso é escrito “SELECT * FROM public.results” o que significa que a tabela pública *results* está selecionada para os métodos de extração ou edição de dados da biblioteca.

Quadro 20 – *Treeview* conectada ao banco de dados

```

#Conexão com o banco de dados
def ins_dados():
    conn = psycopg2.connect(dbname=DB_NAME,
user=DB_USER, password=DB_PASSWORD,
host=DB_HOST)
    cur = conn.cursor()

    global cont, contse, contre, contsu

    cont=0
    contse=0
    contre=0
    contsu=0

    #Puxar o banco de dados
    cur.execute("SELECT * FROM public.results")
    dados=cur.fetchall()

    #Adicionar dados na Treeview
    #Adicionando Etiquetas (para mudar a cor da linha)

    #Commit
    conn.commit()

    #Fechando a conexão
    cur.close()
    conn.close()

```

Fonte: Autoria própria (2022).

A variável *dados* que tinha seus elementos escritos em código é reutilizada agora para receber todos os itens da tabela através do método *fetchall*. Após isso, o código é escrito de forma exatamente igual à descrita no item 3.2.3.

Por fim, é fechada a conexão através dos métodos *commit*, para garantir que a conexão foi estabelecida e houve comunicação entre o programa e o servidor e o método *close* para interromper a comunicação.

A função *edit_record* também foi modificada para poder realizar alterações no banco de dados. As alterações podem ser vistas no Quadro 21.

Quadro 21 – Função *edit_record* com conexão ao banco de dados

```

#Editar os valores do item
def edit_record():
    #Edição dos dados mostrados na Treeview

    #Realiza conexão

    cur.execute("""UPDATE public.results SET
        espessura='{0}',
        largura='{1}',
        comprimento_final='{2}',
        descarte_espessura='{3}',
        descarte_temperatura='{4}',
        descarte_final='{5}',
        resultado_final='{6}'
        WHERE ot='{7}'
        """).format(d2_entry.get(),d3_entry.get(),d4_entry.get(),d5_entry.get(),d6_entr
        y.get(),d7_entry.get(),d8_entry.get(),d1_entry.get())

    #Commit

    #Fechando a conexão

    #Limpar as caixas de entrada

    update()

```

Fonte: Autoria própria (2022).

Nota-se que a diferença para essa função é o uso do método *execute* que permite que um código em *query* SQL seja escrito dentro dele pra realizar edições dentro do banco de dados do PostgreSQL através do *Python*.

São utilizadas três aspas porque assim o texto pode ser quebrado usando a tecla *enter* de modo a facilitar a interpretação do uso das variáveis no código. É usada a função em *query UPDATE* na tabela pública *results* e seleciona através da função *SET* todas as variáveis a serem alteradas, utilizando *ot* como a variável de ID.

Para que o código *query* pudesse entender qual dado fazer a alteração na tabela, foi necessário “indexar”, ou seja, fazer uma espécie de quebra em uma *String* de texto para que uma variável do programa possa ser utilizada no meio dele. Isso foi feito na seção 3.3 de contagem de bobinas e aqui nesse caso foi necessário numerar dentro das chaves para que o valor presente em cada caixa fosse respectivamente alocado à coluna correta na tabela.

4 RESULTADOS

Com todas as funcionalidades aplicadas, pode-se comprovar o funcionamento do programa ao observar a base de dados em conjunto com o sistema de supervisão desenvolvido.

Percebe-se que a interface é clara e com seções bem definidas, assim como também oferece *feedbacks* visuais intuitivos para o usuário.

As Figuras 14 e 15 mostram o programa em modo janela e em tela cheia com a seleção de linha ativada.

Figura 14 – Programa com linha selecionada em modo janela

Interface de Produção

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Verificação dos dados

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None

Editar

Fonte: Autoria própria (2022).

Note que o texto fica muito mais claro de entender no modo tela cheia por conta do espaço de tela maior que é ocupado pela janela de interface. Infelizmente, devido à aplicação, decidiu-se por não alterar o espaço reservado das caixas para nenhuma das variáveis, já que os valores reais, por serem muito precisos, usam muitos algarismos e, logo, bastante espaço nas caixas.

Figura 15 – Programa com linha selecionada em tela cheia

Interface de Produção

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12916	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Verificação dos dados

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12345	905555.0	2.0	1216.0	56.0	251.0	251.0	Redirecionar	None	None

Editar

Fonte: Autoria própria (2022).

Sendo assim, manteve-se essa proporção mesmo sabendo que parte do texto poderia ficar “cortado” se a aplicação se mantivesse em modo janela. Mesmo assim, percebeu-se que com monitores com maiores resoluções, esse problema aparece de forma menos significativa, já que há mais espaço para trabalhar o redimensionamento da janela.

Além disso, o botão de Editar foi testado e, conforme pode-se perceber pelas Figuras 16 até 19, além de editar os dados diretamente na *Treeview*, os valores também são refletidos ao banco de dados.

Note que o valor de “Descarte Espessura” alterou de 58.0 nas Figuras 16 e 17 para 57.2 nas Figuras 18 e 19.

Figura 16 – Exemplo de mudança de Descarte Espessura no programa anterior

Interface de Produção

ot	Espeçura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espeçura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.8880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	58.0	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Verificação dos dados

ot	Espeçura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espeçura do Descarte	Resultado Modelo
12345	905555.0	2.0	1216.0	58.0	251.0	251.0	Redirecionar	None	None

Fonte: Autoria própria (2022).

Figura 17 – Exemplo de mudança de Descarte Espessura no banco de dados anterior

pgAdmin 4

public.results/QualityMonitor/postgres@PostgreSQL 14

```

1 SELECT * FROM public.results
2 ORDER BY ot ASC
    
```

ot	espeçura	largura	comprimento_final	descarte_espeçura	descarte_temperatura	descarte_final	resultado_final	espes
1	12345	905555.0	2.0	1216.0	58.0	251.0	251.0	Redirecionar
2	12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo
3	12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo
4	12789	905555.8	1.8	1214	58	253.008	248	Redirecionar
5	12908	905000	1	1238	55	231	222	Redirecionar
6	12909	905000	2.3	1238	55	231	222	Seguir Processo
7	12918	905001	1	1238	55	231	222	Sucatar
8	16789	900086	2.1	1278.6657	56.7	251.89	251.8880	Seguir Processo

Total rows: 9 of 9 Query complete 00:00:00.085 Rows selected: 1 Ln 1, Col 1

Fonte: Autoria própria (2022).

Figura 18 – Exemplo de mudança de Descarte Espessura no programa após

Interface de Produção

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
12918	905001	1	1238	55	231	222	Sucatar	None	None
12908	905000	1	1238	55	231	222	Redirecionar	None	None
16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo	None	None
12345	905555.0	2.0	1216.0	57.2	251.0	251.0	Redirecionar	None	None
12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo	None	None
129012	905000	1	1238	55	231	222	Seguir Processo	None	None
12789	905555.8	1.8	1214	58	253.008	248	Redirecionar	None	None
12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo	None	None
12909	905000	2.3	1238	55	231	222	Seguir Processo	None	None

Totais da Produção

Total Produzido: 9 Total em Cliente: 5 Total Redirecionado: 3 Total Sucatado: 1

Verificação dos dados

ot	Espessura	Largura	Comprimento Final	Descarte Espessura	Descarte Temperatura	Descarte Final	Resultado Final	Espessura do Descarte	Resultado Modelo
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Editar

Fonte: Autoria própria (2022).

Figura 19 – Exemplo de mudança de Descarte Espessura no banco de dados após

pgAdmin 4

public.results/QualityMonitor/postgres@PostgreSQL 14

```

1 SELECT * FROM public.results
2 ORDER BY ot ASC

```

ot	espesura	largura	comprimento_final	descarte_espesura	descarte_temperatura	descarte_final	resultado_final	espes
1	12345	905555.0	2.0	1216.0	57.2	251.0	251.0	Redirecionar
2	12346	905545.0	2.1	1216.5503	56.554802	251.95465	251.95465	Seguir Processo
3	12347	903545.0	3	1214.5503	56.554802	251.95465	251.95466	Seguir Processo
4	12789	905555.8	1.8	1214	58	253.008	248	Redirecionar
5	12908	905000	1	1238	55	231	222	Redirecionar
6	12909	905000	2.3	1238	55	231	222	Seguir Processo
7	12918	905001	1	1238	55	231	222	Sucatar
8	16789	900086	2.1	1278.6657	56.7	251.89	251.9880	Seguir Processo

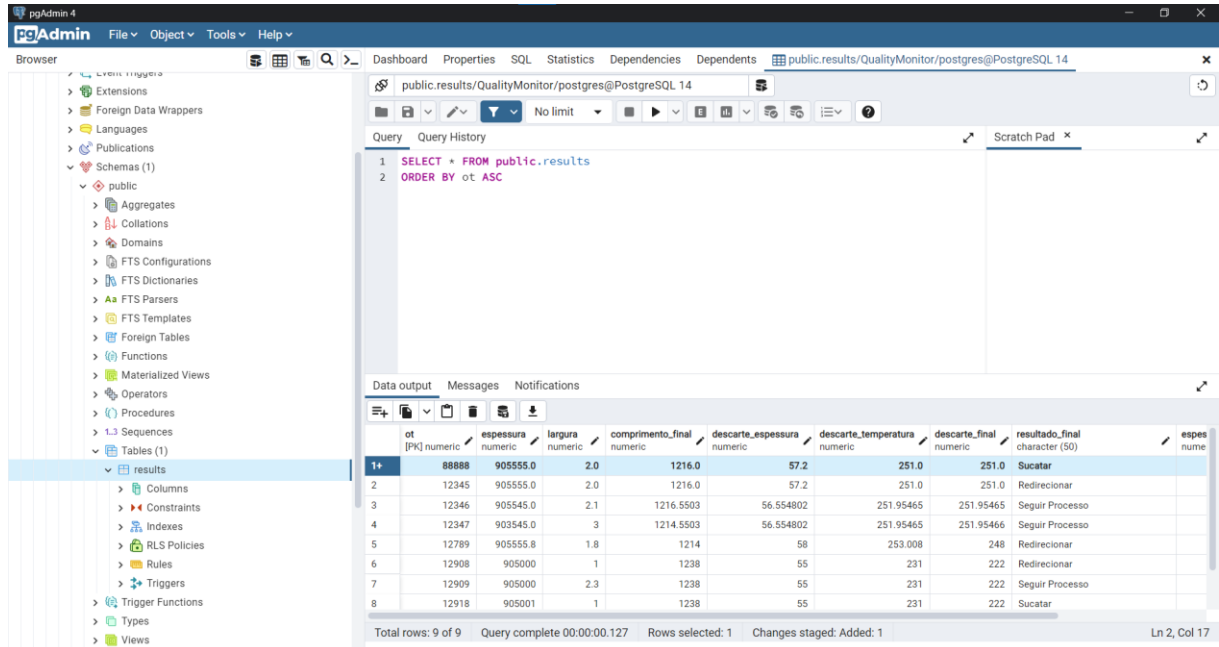
Total rows: 9 of 9 Query complete 00:00:00.127 Rows selected: 1 Ln 1, Col 1

Fonte: Autoria própria (2022).

Por fim, a função de atualização automática pode ser vista através das Figuras 20 e 21, aonde uma nova informação de bobina é adicionada pelo banco de dados e é vista na interface após a atualização de 5 segundos do sistema de supervisão.

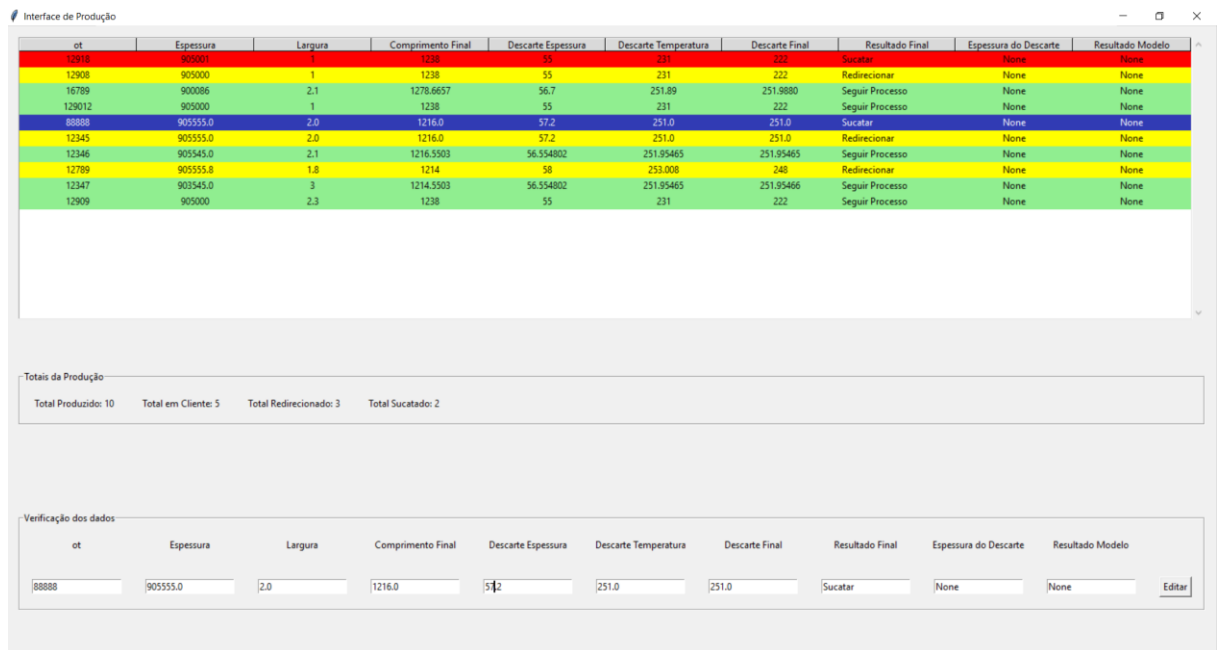
É adicionada uma bobina com todas as características da bobina de *ot* igual a 12345, porém com o *ot* igual a 88888 e com o resultado “Sucatado”.

Figura 20 – Exemplo de nova linha no banco de dados



Fonte: Autoria própria (2022).

Figura 21 – Exemplo de nova linha no programa



Fonte: Autoria própria (2022).

5 CONCLUSÃO

Neste projeto foi desenvolvido um sistema de supervisão utilizando a linguagem de programação em *Python* para aplicação na indústria de laminação de metais. O sistema se comportou como o esperado ao realizar os testes de alteração de dados hipotéticos inseridos no programa.

A interface gráfica atualizou-se corretamente a cada 5 segundos, como configurado e além disso, mostrou uma nova linha na *Treeview* referente a um novo elemento no banco de dados PostgreSQL. Além disso, foi possível editar o banco com as novas informações inseridas nas caixas de texto da interface após acionar o botão “editar”.

Com o funcionamento esperado do sistema de supervisão desenvolvido, pôde-se perceber na prática o quão eficiente é utilizar *Python* para aplicação em sistemas que envolvem banco de dados. Não só existem muitas bibliotecas que facilitam o desenvolvimento do código garantindo um ambiente de programação *front end*, ou seja, de mais alto nível, como também as documentações presentes para essas bibliotecas são bem detalhadas e informativas.

PostgreSQL também se mostrou bem cabível nesse caso. Sua interface é bastante intuitiva e o jeito de manipular dados através do *query* do SQL também é bem fácil de aprender. Pela finalidade do trabalho, ambas bibliotecas do *Python* e o programa PostgreSQL se mostraram mais do que o suficiente, provendo funções cruciais para o funcionamento do sistema e oferecendo, algumas vezes, mais de uma opção para certas aplicações.

Conclui-se, então, que através desse sistema, o supervisor será capaz de acompanhar os dados de produção de laminação de metal com bastante clareza, além de ser capaz de editar os dados de forma muito prática através dos campos de texto e do botão, alcançando o objetivo proposto nesse trabalho.

REFERÊNCIAS

DUBOIS, P. F. **Python: batteries included**. v. 9. IEEE/AIP, 2007.

GIL, A. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.

GRAYSON, J. E. **Python and tkinter programming**. Greenwich: Manning Publications Co., 2000.

MILMANN, K.; AVAIZIS, M. **Scientific python**. v. 11. IEEE/AIP, 2011.

SACOMANO, J. B. *et al.* **Indústria 4.0**. São Paulo: Blucher, 2018.

SHIPMAN, J. W. **Tkinter 8.5 reference: a GUI for python**. New Mexico: Tech Computer Center, v. 54, 2013.

STONES, R.; MATTHEW, N. **Beginning databases with PostgreSQL: from novice to professional**. Apress, 2006.

TAMMINEN, S. *et al.* From measurements to knowledge: online quality monitoring and smart manufacturing. *In*: PERNER, P. (ed.). **Advances in data mining**. Applications and theoretical aspects. Springer, Cham, 2018. p. 17-28.

YACOB, F. *et al.* Anomaly detection in skin model shapes using machine learning classifiers. **The International Journal of Advanced Manufacturing Technology**, v. 105, n. 9, p. 3677-3689, 2019.

APÊNDICE A – Programa do Sistema de Supervisão em *Python*

```
#Sistema de Supervisão em python

#Definição das variáveis de conexão ao banco de dados
DB_HOST="localhost"
DB_NAME="QualityMonitor"
DB_USER="postgres"
DB_PASSWORD="123"

#Importando as bibliotecas
import psycopg2
from tkinter import *
from tkinter import ttk

#Definição da Janela
window=Tk()
window.title("Interface de Produção")
window.geometry("1280x720")

#Configuração do estilo da treeview
style = ttk.Style()
style.theme_use('default')
style.configure("Treeview", background="white", foreground="black",
fieldbackground="white")

#Frame no qual entrará o Treeview
tree_frame = Frame(window)
tree_frame.pack(fill="both", expand=1, padx=15, pady=15)

#Scrollbar da Treeview
tree_scroll = Scrollbar(tree_frame)
tree_scroll.pack(side="right", fill="y")
```

```
#Treeview
my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set, selectmode="extended",
height=12)
my_tree.pack(fill="both", expand=1)

#Configuração da Scrollbar
tree_scroll.config(command=my_tree.yview) #Deve ser definido após a criação da classe
my_tree

#Definição das colunas
String1="ot"
String2="Espessura"
String3="Largura"
String4="Comprimento Final"
String5="Descarte Espessura"
String6="Descarte Temperatura"
String7="Descarte Final"
String8="Resultado Final"
String9="Espessura do Descarte"
String10="Resultado Modelo"
my_tree['columns'] = (String1, String2, String3, String4, String5, String6, String7, String8,
String9, String10)

#Formatação das Colunas
my_tree.column("#0", width=0, stretch=NO)
my_tree.column(String1, anchor=CENTER, width=100)
my_tree.column(String2, anchor=CENTER, width=100)
my_tree.column(String3, anchor=CENTER, width=100)
my_tree.column(String4, anchor=CENTER, width=100)
my_tree.column(String5, anchor=CENTER, width=100)
my_tree.column(String6, anchor=CENTER, width=100)
my_tree.column(String7, anchor=CENTER, width=100)
my_tree.column(String8, anchor=CENTER, width=100)
my_tree.column(String9, anchor=CENTER, width=100)
```

```
my_tree.column(String10, anchor=CENTER, width=100)
```

```
#Criação dos Headings
```

```
my_tree.heading("#0", text="", anchor=CENTER)
```

```
my_tree.heading(String1, text=String1, anchor=CENTER)
```

```
my_tree.heading(String2, text=String2, anchor=CENTER)
```

```
my_tree.heading(String3, text=String3, anchor=CENTER)
```

```
my_tree.heading(String4, text=String4, anchor=CENTER)
```

```
my_tree.heading(String5, text=String5, anchor=CENTER)
```

```
my_tree.heading(String6, text=String6, anchor=CENTER)
```

```
my_tree.heading(String7, text=String7, anchor=CENTER)
```

```
my_tree.heading(String8, text=String8, anchor=CENTER)
```

```
my_tree.heading(String9, text=String9, anchor=CENTER)
```

```
my_tree.heading(String10, text=String10, anchor=CENTER)
```

```
cont=0
```

```
contse=0
```

```
contre=0
```

```
contsu=0
```

```
#Conexão com o banco de dados
```

```
def ins_dados():
```

```
    conn = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST)
```

```
    cur = conn.cursor()
```

```
    global cont, contse, contre, contsu
```

```
    cont=0
```

```
    contse=0
```

```
    contre=0
```

```
    contsu=0
```

```
#Puxar o banco de dados
```



```

cur.execute("SELECT * FROM public.results")
dados=cur.fetchall()

#Adicionar dados na Treeview
seguir="Seguir Processo"
redirecionar="Redirecionar"

for record in dados:
    if record[7].find(seguir) != -1:
        my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0], record[1],
record[2], record[3], record[4], record[5], record[6], record[7], record[8], record[9]),
tags=('seguir',))
        cont += 1
        contse += 1

    elif record[7].find(redirecionar) != -1:
        my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0], record[1],
record[2], record[3], record[4], record[5], record[6], record[7], record[8], record[9]),
tags=('redirecionado',))
        cont += 1
        contre += 1

    else:
        my_tree.insert(parent="", index="0", iid=cont, text="", values = (record[0], record[1],
record[2], record[3], record[4], record[5], record[6], record[7], record[8], record[9]),
tags=('sucitado',))
        cont += 1
        contsu += 1

#Adicionando Etiquetas (para mudar a cor da linha)
my_tree.tag_configure('seguir', background="lightgreen")
my_tree.tag_configure('redirecionado', background="yellow")
my_tree.tag_configure('sucitado', background="red")

#Commit

```

```

conn.commit()

#Fechando a conexão
cur.close()
conn.close()

#Frame total de bobinas
totalb_frame = LabelFrame(window, text="Totais da Produção")
totalb_frame.pack(fill="x", expand="yes", padx=15)

#Atualizar o total de bobinas no Frame
def totalb_update():
    totp_label=Label(totalb_frame, text=("Total Produzido: {}".format(cont)))
    totp_label.grid(row=0, column=0, padx=15, pady=15, sticky = "nsew")
    totc_label=Label(totalb_frame, text=("Total em Cliente: {}".format(contse)))
    totc_label.grid(row=0, column=1, padx=15, pady=15, sticky = "nsew")
    totred_label=Label(totalb_frame, text=("Total Redirecionado: {}".format(contre)))
    totred_label.grid(row=0, column=2, padx=15, pady=15, sticky = "nsew")
    totsuc_label=Label(totalb_frame, text=("Total Sucatado: {}".format(consu)))
    totsuc_label.grid(row=0, column=3, padx=15, pady=15, sticky = "nsew")

#Deletar o total de bobinas no frame
def totalb_delete():
    for i in totalb_frame.winfo_children():
        i.destroy()

#Definição do Frame e etiqueta para a área de verificação dos dados
dados_frame = LabelFrame(window, text="Verificação dos dados")
dados_frame.pack(fill="x", side="bottom", expand="yes", padx=15, pady=10)
dados_frame.columnconfigure(0, weight=1)
dados_frame.columnconfigure(1, weight=1)
dados_frame.columnconfigure(2, weight=1)
dados_frame.columnconfigure(3, weight=1)

```

```
dados_frame.columnconfigure(4, weight=1)
dados_frame.columnconfigure(5, weight=1)
dados_frame.columnconfigure(6, weight=1)
dados_frame.columnconfigure(7, weight=1)
dados_frame.columnconfigure(8, weight=1)
dados_frame.columnconfigure(9, weight=1)
```

#Exibição (cima) e inserção de dados (baixo)

```
d1_label=Label(dados_frame, text=String1)
d1_label.grid(row=0, column=0, padx=15, pady=15)
d1_entry=Entry(dados_frame)
d1_entry.grid(row=1, column=0, padx=15, pady=5)
```

```
d2_label=Label(dados_frame, text=String2)
d2_label.grid(row=0, column=1, padx=15, pady=15)
d2_entry=Entry(dados_frame)
d2_entry.grid(row=1, column=1, padx=15, pady=5)
```

```
d3_label=Label(dados_frame, text=String3)
d3_label.grid(row=0, column=2, padx=15, pady=15)
d3_entry=Entry(dados_frame)
d3_entry.grid(row=1, column=2, padx=15, pady=5)
```

```
d4_label=Label(dados_frame, text=String4)
d4_label.grid(row=0, column=3, padx=15, pady=15)
d4_entry=Entry(dados_frame)
d4_entry.grid(row=1, column=3, padx=15, pady=5)
```

```
d5_label=Label(dados_frame, text=String5)
d5_label.grid(row=0, column=4, padx=15, pady=15)
d5_entry=Entry(dados_frame)
d5_entry.grid(row=1, column=4, padx=15, pady=5)
```

```
d6_label=Label(dados_frame, text=String6)
```

```
d6_label.grid(row=0, column=5, padx=15, pady=15)
d6_entry=Entry(dados_frame)
d6_entry.grid(row=1, column=5, padx=15, pady=5)
```

```
d7_label=Label(dados_frame, text=String7)
d7_label.grid(row=0, column=6, padx=15, pady=15)
d7_entry=Entry(dados_frame)
d7_entry.grid(row=1, column=6, padx=15, pady=5)
```

```
d8_label=Label(dados_frame, text=String8)
d8_label.grid(row=0, column=7, padx=15, pady=15)
d8_entry=Entry(dados_frame)
d8_entry.grid(row=1, column=7, padx=15, pady=5)
```

```
d9_label=Label(dados_frame, text=String9)
d9_label.grid(row=0, column=8, padx=9, pady=15)
d9_entry=Entry(dados_frame)
d9_entry.grid(row=1, column=8, padx=15, pady=5)
```

```
d10_label=Label(dados_frame, text=String10)
d10_label.grid(row=0, column=9, padx=15, pady=15)
d10_entry=Entry(dados_frame)
d10_entry.grid(row=1, column=9, padx=15, pady=5)
```

```
#Limpar a Treeview
```

```
def clear_treeview():
    for item in my_tree.get_children():
        my_tree.delete(item)
```

```
#Clicar e selecionar dados
```

```
#Mudar Cor da seleção
```

```
style.map("Treeview", background=[('selected', "#313BB4")])
```

```
#Variável global de seleção da linha
```

```
selected=""  
  
def select_data(e):  
    #Limpar as caixas de entrada  
    d1_entry.delete(0, END)  
    d2_entry.delete(0, END)  
    d3_entry.delete(0, END)  
    d4_entry.delete(0, END)  
    d5_entry.delete(0, END)  
    d6_entry.delete(0, END)  
    d7_entry.delete(0, END)  
    d8_entry.delete(0, END)  
    d9_entry.delete(0, END)  
    d10_entry.delete(0, END)  
  
    global selected  
    selected=my_tree.focus()  
    values=my_tree.item(selected, 'values')  
  
    #Jogar informação na entry box  
    d1_entry.insert(0, values[0])  
    d2_entry.insert(0, values[1])  
    d3_entry.insert(0, values[2])  
    d4_entry.insert(0, values[3])  
    d5_entry.insert(0, values[4])  
    d6_entry.insert(0, values[5])  
    d7_entry.insert(0, values[6])  
    d8_entry.insert(0, values[7])  
    d9_entry.insert(0, values[8])  
    d10_entry.insert(0, values[9])  
  
#Atualizar tudo  
def update():  
    clear_treeview()  
    ins_dados()
```

```

totalb_delete()
totalb_update()
#Manter linha selecionada
my_tree.selection_set(selected)
update_loop()

```

```
#Editar os valores do item
```

```

def edit_record():
    #Edição dos dados mostrados na Treeview
    selected=my_tree.focus()
    my_tree.item(selected, text="", values=(d1_entry.get(),
                                           d2_entry.get(),
                                           d3_entry.get(),
                                           d4_entry.get(),
                                           d5_entry.get(),
                                           d6_entry.get(),
                                           d7_entry.get(),
                                           d8_entry.get(),
                                           d9_entry.get(),
                                           d10_entry.get(),
                                           ))

```

```
#Realiza conexão
```

```

conn = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST)
cur = conn.cursor()

```

```

cur.execute("""UPDATE public.results SET
    espessura='{0}',
    largura='{1}',
    comprimento_final='{2}',
    descarte_espessura='{3}',
    descarte_temperatura='{4}',

```

```
        descarte_final='{5}',
        resultado_final='{6}'
        WHERE ot='{7}'
""" .format(d2_entry.get(),d3_entry.get(),d4_entry.get(),d5_entry.get(),d6_entry.get(),d7_entry.
get(),d8_entry.get(),d1_entry.get()))
```

```
#Commit
```

```
conn.commit()
```

```
#Fechando a conexão
```

```
cur.close()
```

```
conn.close()
```

```
#Limpar as caixas de entrada
```

```
d1_entry.delete(0, END)
```

```
d2_entry.delete(0, END)
```

```
d3_entry.delete(0, END)
```

```
d4_entry.delete(0, END)
```

```
d5_entry.delete(0, END)
```

```
d6_entry.delete(0, END)
```

```
d7_entry.delete(0, END)
```

```
d8_entry.delete(0, END)
```

```
d9_entry.delete(0, END)
```

```
d10_entry.delete(0, END)
```

```
update()
```

```
#Adição de botões
```

```
button_edit=Button(dados_frame, text="Editar", command=edit_record)
```

```
button_edit.grid(row=1, column=10, padx=15, pady=15, sticky=EW)
```

```
#Acoplar Treeview na função de seleção
```

```
my_tree.bind("<ButtonRelease-1>", select_data)
```

```
ins_dados()
```

```
totalb_update()
```

```
#Loop
```

```
def update_loop():
```

```
    window.after(5000, update)
```

```
update_loop()
```

```
mainloop()
```