**Beleza Inteligente: Aplicando Modelos de Machine Learning para Personalizar a Experiência do Cliente em Salões de Beleza**

## Anselmo Berriel de Lira, Beatriz Ornelas Miranda, Guilherme Teixeira Silva, Roberto Borba Pinto, Thiago dos Santos Rodrigues, Thomas Eduardo da Silva Olympio, Wellington Patrick Viana dos Santos

PUC Minas

Curso de Tecnologia em Banco de Dados

[ablira@sga.pucminas.br,](mailto:ablira@sga.pucminas.br) [beatriz.miranda@sga.pucminas.br,](mailto:beatriz.miranda@sga.pucminas.br) [guilherme.silva.1396238@sga.pucminas.br,](mailto:guilherme.silva.1396238@sga.pucminas.br) [rbpinto@sga.pucminas.br,](mailto:rbpinto@sga.pucminas.br) [thiago.rodrigues@sga.pucminas.br,](mailto:thiago.rodrigues@sga.pucminas.br) [tesolympio@sga.pucminas.br,](mailto:tesolympio@sga.pucminas.br) [wellington.patrick@sga.pucminas.br](mailto:wellington.patrick@sga.pucminas.br)

***Resumo:*** *O grupo de estudos da PUC MINAS em parceria a empresa Fios de Luxo buscará explicar a relação das estações e como a temperatura afeta o comercio localizado em Salvador, Bahia. A necessidade da empresa de entender o clima pode ser a chave para antecipar tendências e o grupo está determinado a encontrar as respostas diante das dúvidas da empresa parceiras.*

***Abstract:*** *The PUC MINAS study group, in partnership with the company Fios de Luxo, will try to explain the relationship between the seasons and how the temperature affects the business located in Salvador, Bahia. The company's need to understand the climate could be the key to anticipating trends and the group is determined to find answers to the partner company's questions.*

# Introdução

Busca-se com esta análise de dados que será realizada, trazer informações completas a respeito de como o clima e suas estações podem influenciar o setor comercial que abrangem atendimentos focados em um salão de beleza visando a melhor maneira para utilizar as informações em benefício de um comércio bem-sucedido.

A análise de dados será de extrema importância para atender e antecipar tendências no comercio da empresa Fios de Luxo, entender as demandas do seu público-alvo, como o clima afeta a relação de consumo, antecipar tendências por estações é o principal motivador para o grupo.

Para esse estudo, serão coletados os dados através do Trinks (salão parceiro irá nos enviar os dados brutos para trata-los) e da seguinte URL:

<https://openweathermap.org/>

Os dados a serem coletados serão os seguintes:

* Pressão
* Temperatura
* Vento
* Umidade
* Resumo dia

# Componentes de Arquitetura

Modelagem: SQL Relacional - MySQL SGBD: MySQL

Plataforma e serviços de nuvem: AWS Frameworks: Api Rest e Open Weather Linguagem de programação: Python Ferramenta de banco de dados: Dbeaver IDE: Visual Studio Code

**Segurança dos Dados e LGPD**

## Ocultação de Dados Reais dos Clientes do Salão Parceiro

**Segurança de Dados:**

O projeto de Meteorologia prioriza a segurança dos dados coletados e armazenados. As medidas tomadas para garantir a segurança incluem:

* **Uso de protocolos seguros:**
  + HTTPS para comunicação com a API de meteorologia.
  + SSH para acesso ao banco de dados RDS.
* **Criptografia de dados:**
  + Senhas armazenadas em formato criptografado.
  + Banco de dados RDS criptografado em repouso e em trânsito.
* **Controle de acesso:**
  + Permissões granulares para acesso ao banco de dados e aos dados coletados.
  + Autenticação de usuários com senhas fortes e chaves SSH.
* **Monitoramento e registro:**
  + Monitoramento da atividade do sistema para detectar e prevenir acessos não autorizados.
  + Registro de todas as atividades do sistema para fins de auditoria.

**Importância da Privacidade:**

No projeto de Meteorologia, a privacidade dos clientes do salão parceiro é de suma importância. Para garantir a confidencialidade das informações, os dados reais dos clientes foram ocultados durante todo o processo de análise.

**Processo de Ocultação:**

As tabelas que continham dados reais dos clientes foram tratadas para torná-las fictícias. Isso foi feito através de técnicas como:

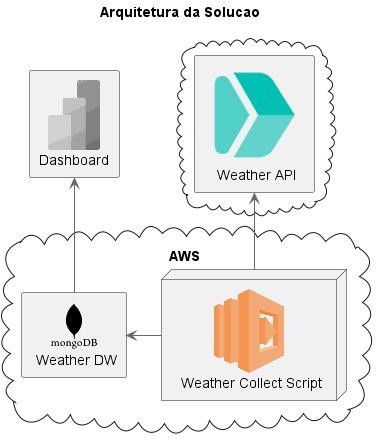
* **Pseudonimização:**
  + Os nomes dos clientes foram substituídos por pseudônimos aleatórios.
* **Remoção de dados sensíveis:**
  + Quaisquer dados que poderiam ser usados para identificar ou contatar os clientes foram removidos.
* **Adição de ruído:**
  + Ruído aleatório foi adicionado a alguns dados para torná-los menos precisos e menos úteis para fins de identificação.

**Benefícios da Ocultação:**

A ocultação dos dados reais dos clientes oferece diversos benefícios, como:

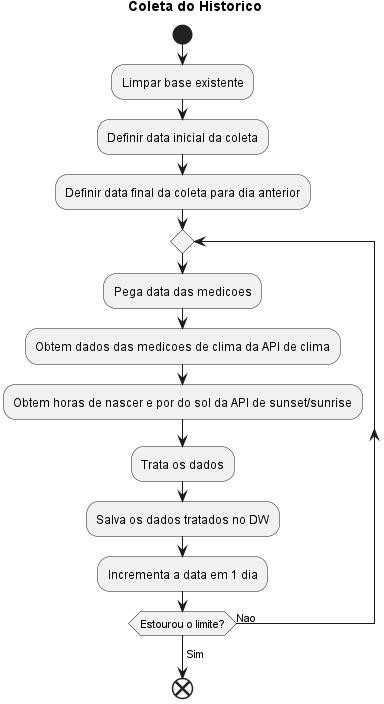
* **Proteção da privacidade dos clientes:**
  + Os clientes podem ter certeza de que seus dados pessoais não serão expostos ou vazados.
* **Maior segurança:**
  + O risco de violação de dados é reduzido, pois os dados reais dos clientes não estão disponíveis.
* **Maior confiabilidade dos resultados:**
  + A ocultação dos dados reais ajuda a garantir que os resultados da análise sejam imparciais e confiáveis.

# Arquitetura da Solução



*Figura 1- Arquitetura da solução*

# Coleta do Histórico



*Figura 2- Coleta do hitstórico*

# Modelagem de Dados:

O projeto utilizará o esquema de banco de dados no MySQL para armazenar os dados meteorológicos, como temperatura, clima, turno. Seguindo a estrutura de Data Warehouse Star Schema.

**Tabela dimensão d\_agenda:**

data\_id DATE NOT NULL PRIMARY KEY,

dia INT DEFAULT NULL,

UNIQUE(data\_id)

**Tabela dimensão d\_clima:**

id INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

temperatura\_min DOUBLE DEFAULT NULL,

temperatura\_max DOUBLE DEFAULT NULL,

temperatura\_tarde DOUBLE DEFAULT NULL,

temperatura\_noite DOUBLE DEFAULT NULL,

temperatura\_manha DOUBLE DEFAULT NULL,

dia DATE NOT NULL,

data\_atual DATE NOT NULL,

FOREIGN KEY (dia) REFERENCES d\_agenda(data\_id),

UNIQUE (dia)

**Tabela dimensão d\_cliente:**

id int unsigned not null auto\_increment,

nome varchar(120) not null primary key,

telefone varchar(45) default null

**Tabela dimensãod\_funcionario:**

id int unsigned primary key not null auto\_increment,

nome varchar(120) not null,

cargo varchar(64) default null,

telefone varchar(20) default null

**Tabela dimensão d\_servico:**

id int unsigned not null primary key auto\_increment,

nome varchar(120) not null,

valor int DEFAULT 0

**Tabela fato\_pagamento:**

id\_cliente int unsigned not null,

data\_id date not null,

id\_funcionario int unsigned not null,

id\_servico int unsigned not null,

valor\_pago float default 0,

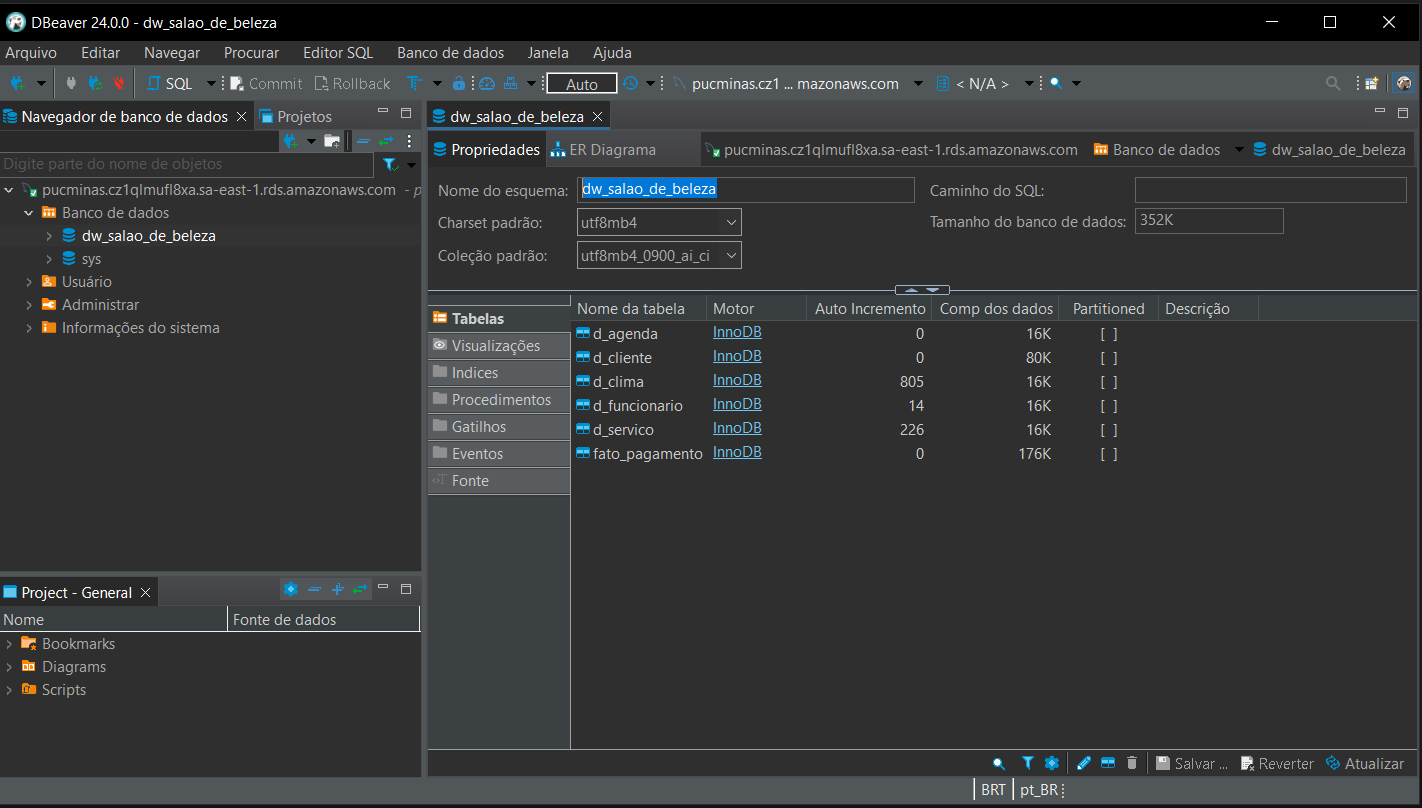
primary key (id\_cliente, data\_id, id\_funcionario, id\_servico),

foreign key FK\_FatoPagamento\_Cliente(id\_cliente) references d\_cliente(id),

foreign key FK\_FatoPagamento\_Agenda(data\_id) references d\_agenda(data\_id),

foreign key FK\_FatoPagamento\_Funcionario(id\_funcionario) references d\_funcionario(id),

foreign key FK\_FatoPagamento\_Servico(id\_servico) references d\_servico(id)



*Figura 3 – Modelagem de dados no Mysql*

# Coleta de Dados:

O sistema de coleta de dados que obterá informações meteorológicas para as cidades especificadas a partir da API REST fornecida. Os dados serão coletados em intervalos regulares (por exemplo, a cada hora) e inseridos na tabela "d\_Clima" do RDS. Assim como os dados do salão coletados pelo site <https://trinks.com.br>, os dados serão em .csv e incluidos nas tabelas "d\_agenda", "d\_cliente", "d\_funcionario", "d\_servico" e "f\_pagamento"

# Arquitetura de Coleta:

A arquitetura de coleta terá integração com a API REST fornecida. Os serviços da AWS serão utilizados para criar funções Lambda que disparem a coleta de dados em intervalos regulares apenas os dados climáticos, devido a questão de segurança e privacidade do salão parceiro Fios de Luxo os dados serão fornecidos por ele quando solicitado os dados referentes as tabelas de serviço, agenda, cliente e pagamento. A API REST fornecida será usada para obter os dados atualizados.

# Orquestração do pipeline de dados

## Capturando dados históricos climáticos com Python e API (OpenWeather)

import requests

import mysql.connector

from datetime import datetime, timedelta

# Dados de acesso ao banco de dados user = 'root'

password = '123456' host = 'localhost'

database = 'dw\_salao\_de\_beleza'

# Chave de API OpenWeather

API\_KEY = "221f164d35f7154a527c7b3146fa2129"

# Coordenadas de Salvador latitude = -12.9704

longitude = -38.5124

# Definindo a data inicial e a data final data\_inicial = datetime(2023, 5, 2)

data\_final = datetime(2023, 5, 31)

# Função para converter de Kelvin para Celsius def kelvin\_para\_celsius(temp\_kelvin):

return temp\_kelvin - 273.15

# Função para obter dados climáticos da API OpenWeather def obter\_temperatura(data):

print("Iniciando coleta") link =

f"https://api.openweathermap.org/data/3.0/onecall/day\_summary?lat={latitu de}&lon={longitude}&date={data.strftime('%Y-%m-%d')}&appid={API\_KEY}"

requisicao = requests.get(link) requisicao\_dic = requisicao.json() temperatura\_min =

kelvin\_para\_celsius(requisicao\_dic['temperature']['min']) temperatura\_max =

kelvin\_para\_celsius(requisicao\_dic['temperature']['max']) temperatura\_tarde =

kelvin\_para\_celsius(requisicao\_dic['temperature']['afternoon']) temperatura\_noite =

kelvin\_para\_celsius(requisicao\_dic['temperature']['night']) temperatura\_noite =

kelvin\_para\_celsius(requisicao\_dic['temperature']['evening']) temperatura\_manha =

kelvin\_para\_celsius(requisicao\_dic['temperature']['morning']) return temperatura\_min, temperatura\_max, temperatura\_tarde,

temperatura\_noite, temperatura\_manha

# Loop para coletar os dados para cada data while data\_inicial <= data\_final:

temperatura\_min, temperatura\_max, temperatura\_tarde, temperatura\_noite, temperatura\_manha = obter\_temperatura(data\_inicial)

if temperatura\_min is not None and temperatura\_max is not None and temperatura\_tarde is not None and temperatura\_noite is not None and temperatura\_manha is not None:

# Conectar ao banco de dados db\_connection = mysql.connector.connect(

host=host, user=user, password=password, database=database

)

cursor = db\_connection.cursor()

# Inserir os dados no banco de dados

insert\_query = "INSERT INTO d\_clima (temperatura\_min, temperatura\_max, temperatura\_tarde, temperatura\_noite, temperatura\_manha, dia, data\_atual) VALUES (%s, %s, %s, %s, %s, %s, %s)"

data\_insercao = (temperatura\_min, temperatura\_max, temperatura\_tarde, temperatura\_noite, temperatura\_manha, data\_inicial.strftime('%Y-%m-%d'), datetime.now().strftime('%Y-%m-%d'))

cursor.execute(insert\_query, data\_insercao) db\_connection.commit()

# Fechar a conexão com o banco de dados cursor.close()

db\_connection.close()

# Incrementar a data inicial em um dia data\_inicial += timedelta(days=1)

print("Dados coletados e inseridos no banco de dados com sucesso.")

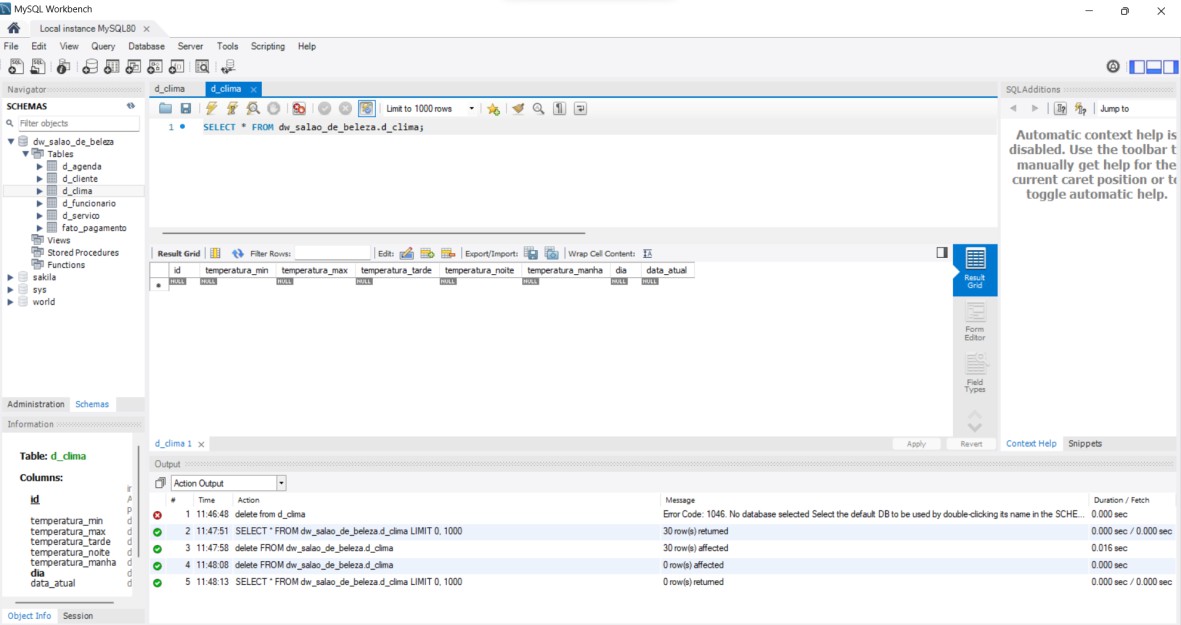
*Figura 4 – Código python*

# Desenvolvimento das atividades

A captura dos dados climáticas se inicia com a inserção da chave API que se dá através do cadastro no site, <https://openweathermap.org/>, após isso foi necessário fazer uma requisição HTTPS que gera um JSON como resultado da coleta dos dados climáticos. Coletados os dados o código faz uma conexão com o Banco de Dados MySQL, quando conectado ele transforma os dados JSON chaves em colunas para inserção dos valores armazenados. Assim ele retorna um “Coletando dados” quando inicia a coleta e “Dados coletados e inseridos no banco de dados com sucesso” quando a inserção no banco não retorna erro.

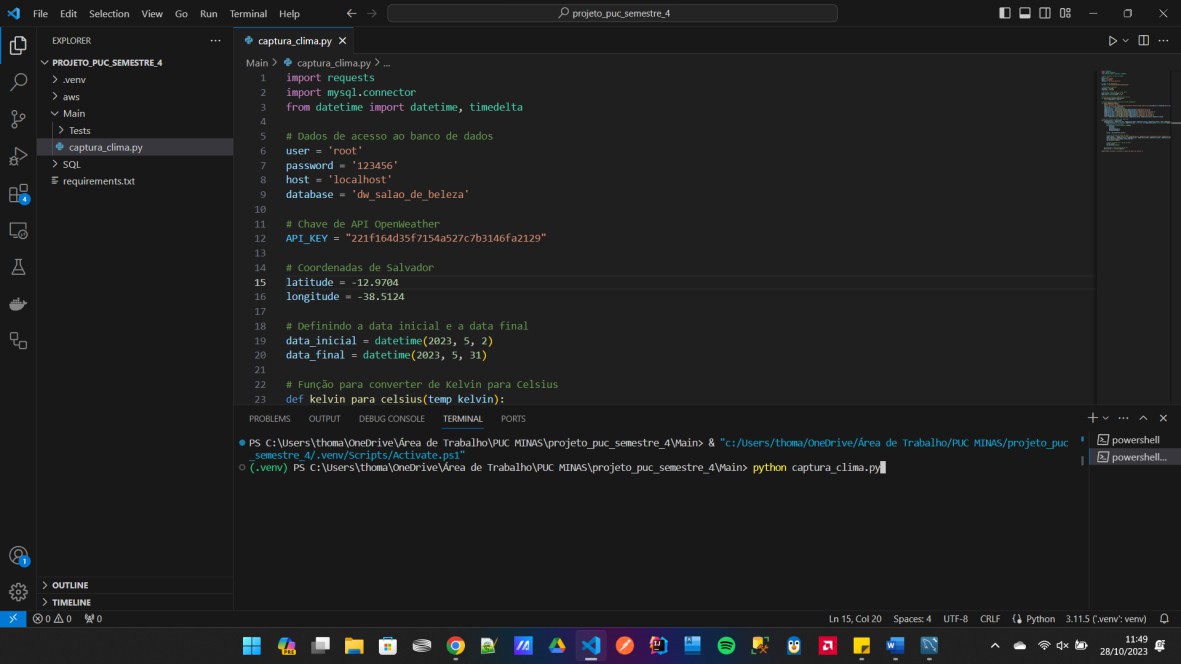
# Tabela dimensão D\_clima

A tabela *“d\_clima”* iniciou sem dados

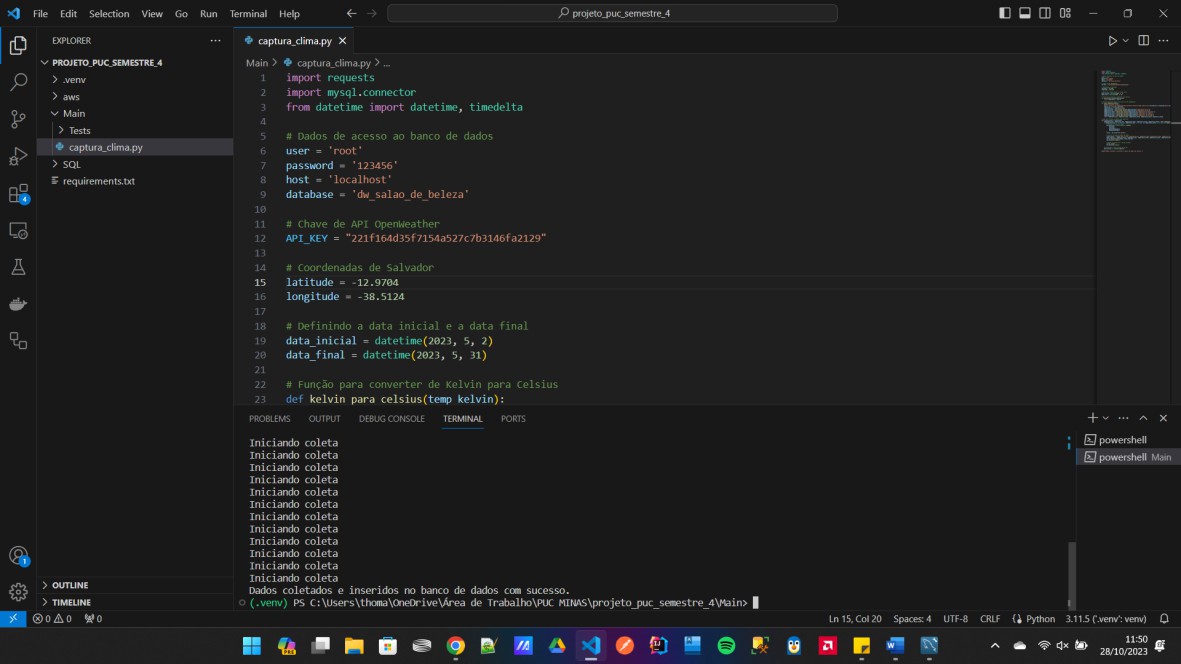


*Figura 5 - Exibição da tabela d\_clima no MySQL*

# Executando o código em Python



*Figura 6 – Teste no Visual Studio Code*



*Figura 7 - Continuação dos testes no Visual Studio Code*

# Configuração em ambiente AWS

**Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente**

*Figura 8 – Criação do banco AWS*

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

*Figura 9 - Identificação das instâncias conectadas ao AWS*

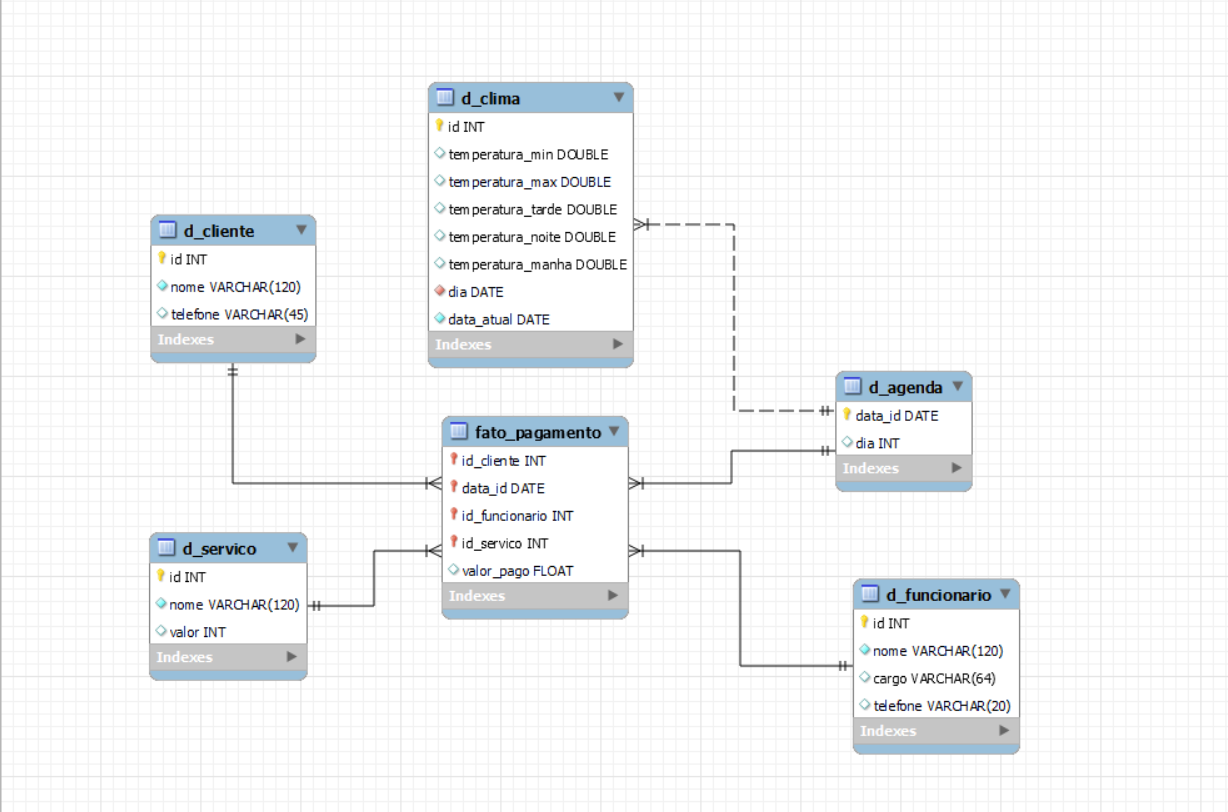
*Interface gráfica do usuário

Descrição gerada automaticamente*

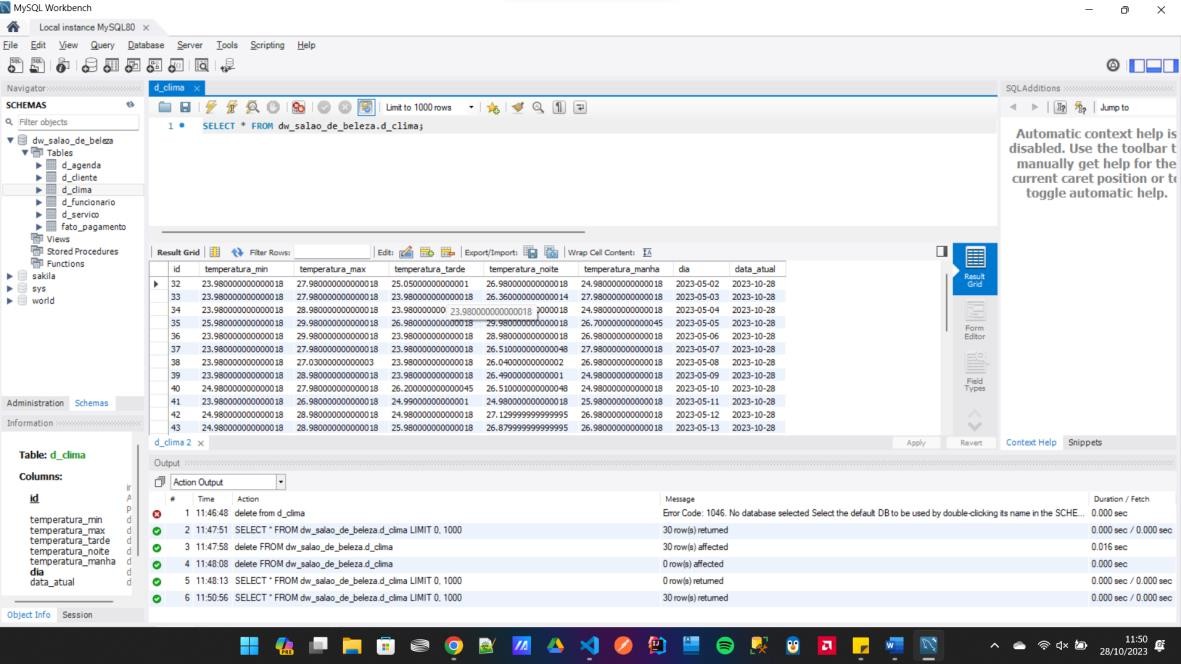
*Figura 10 - Configuração do DBeaver com o RDS da AWS*

# Acessando o ambiente RDS AWS NO Dbeaver

* + - Identificador da instancia: banco-pucminas
    - banco-pucminas.cyqkssq3ycqa.us-east-2.rds.amazonaws.com (acesso ao bd puc minas salao fios de luxo)
    - porta: 3306
    - nome usuario: admin
    - senha: Samoht123.
    - DB: dw\_salao\_de\_beleza

**

*Figura 11 - Captura do diagrama do Banco*



*Figura 12 - Inserção de novos dados no MySQL*

# Resultado obtido

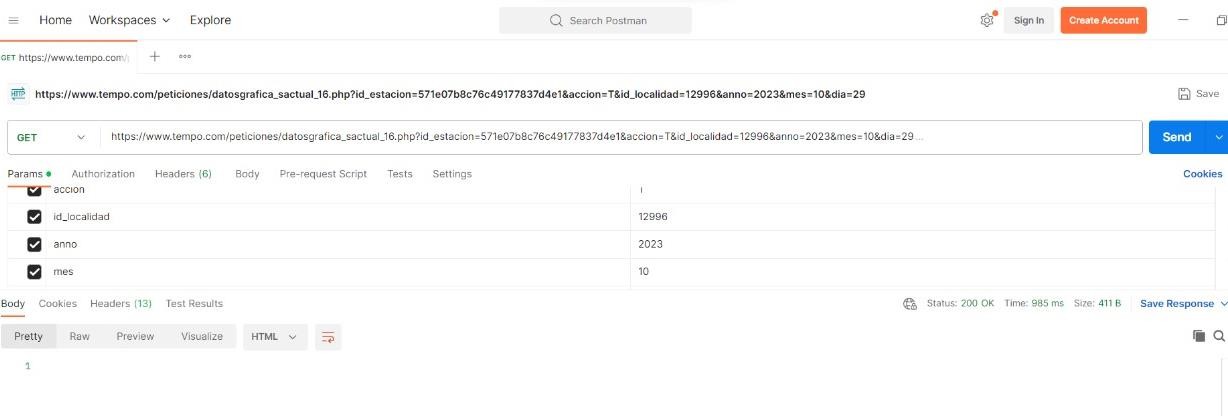
Consideramos satisfatórios os resultados da coleta, quando foi retornado o resultado JSON:

* Temperatura mínima
* Temperatura máxima
* Temperatura de noite
* Temperatura de manhã
* Dia que foi extraído os dados históricos
* Data atual da requisição

Obs: Será implementado no código a capacidade de verificar dados coletados, e atribuir uma nova API para obter dados da previsão do dia que foi feita a requisição HTTP.

# Dificuldades durante a implementação da etapa

* Nosso maior desafio foi encontrar disponibilidade de todos os integrantes do grupo em um mesmo horário;
* Conseguir configurar uma instância e se conectar a ela a partir de um computador local;
* Criação do banco no RDS da AWS, que foi solucionada com a ajuda do monitor Caio;
* A API escolhida inicialmente se mostrou instável, hora respondendo normalmente e em outros momentos retornava status 200 sem conteúdo, obrigando o grupo a migrar para outra API.



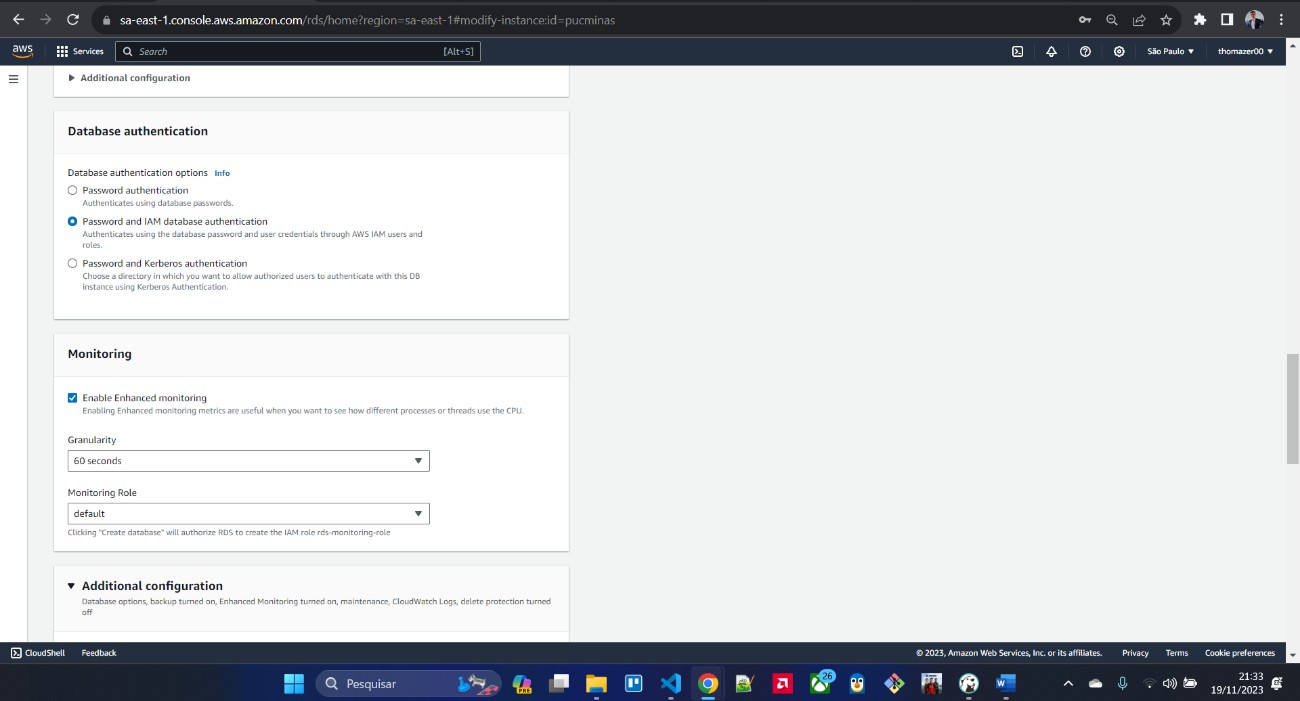
*Figura 13 - Dificuldade obtida na primeira API escolhida*

# Observação e monitoria dos dados

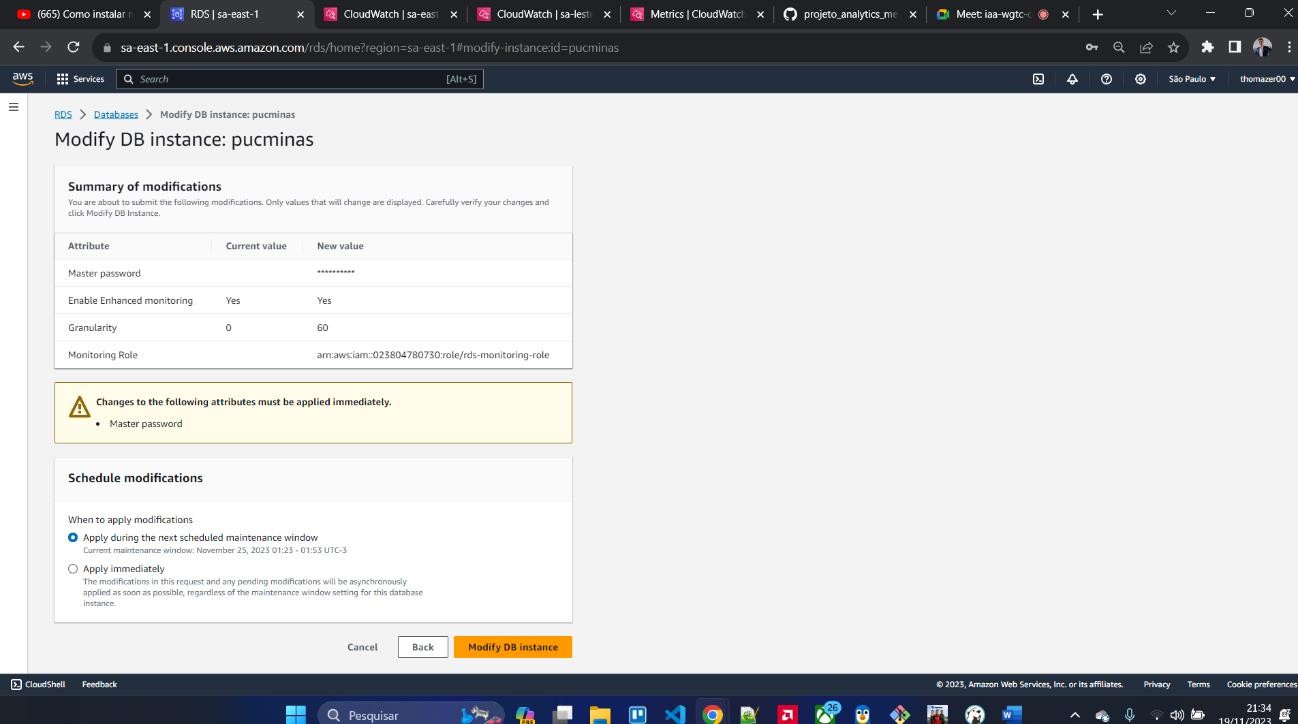
A ferramenta “Cloud Watch” foi escolhida para permitir o processo de observação e monitoria dos dados, bem como a sua implantação e configuração.

Nas figuras abaixo descrevemos o que foi monitorado, o volume de dados processado em determinado período, a quantidade de dados na origem, a quantidade de dados armazenados no destino e o tempo de processamento.

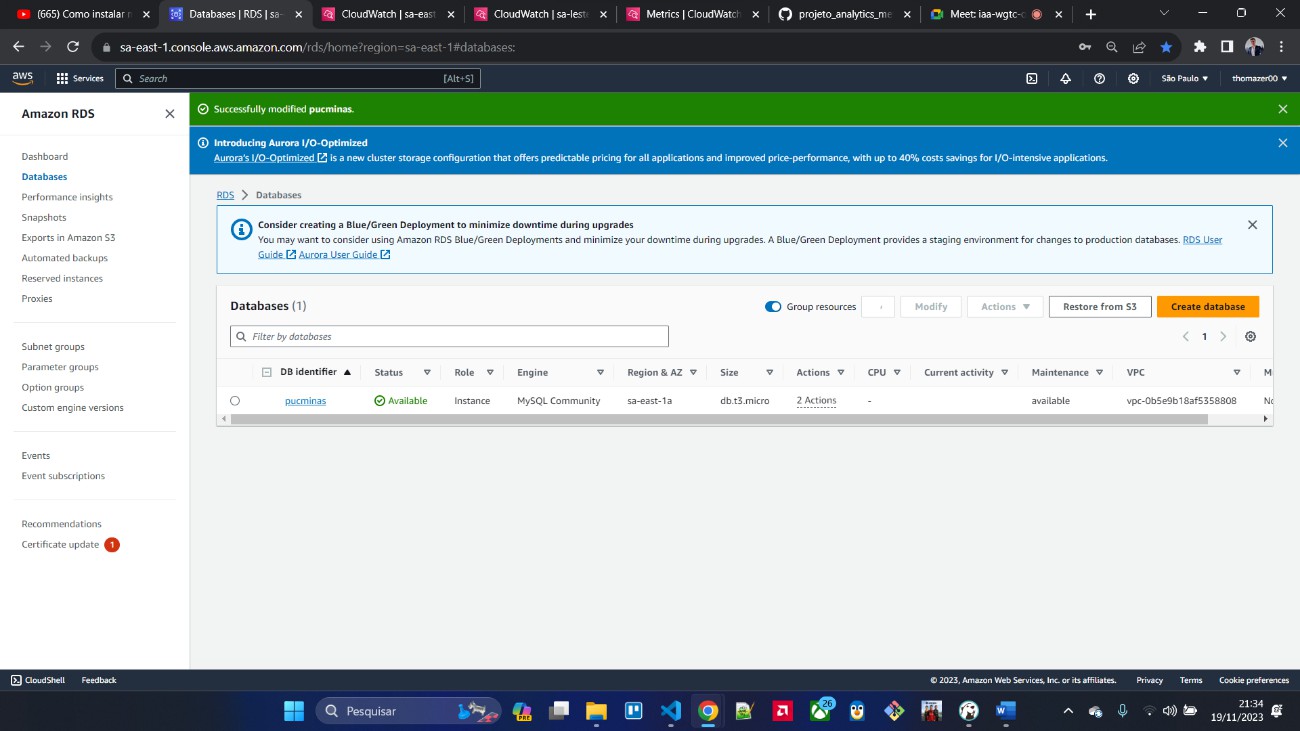
Solicitando o monitoramento gratuito do AWS com a ferramenta Cloud Watch:



*Figura 14 – Ferramenta Cloud Watch utilizada como monitoria e observação dos dados.*



*Figura 15 – Continuação da ferramenta Cloud Watch utilizada como monitoria e observação dos dados.*

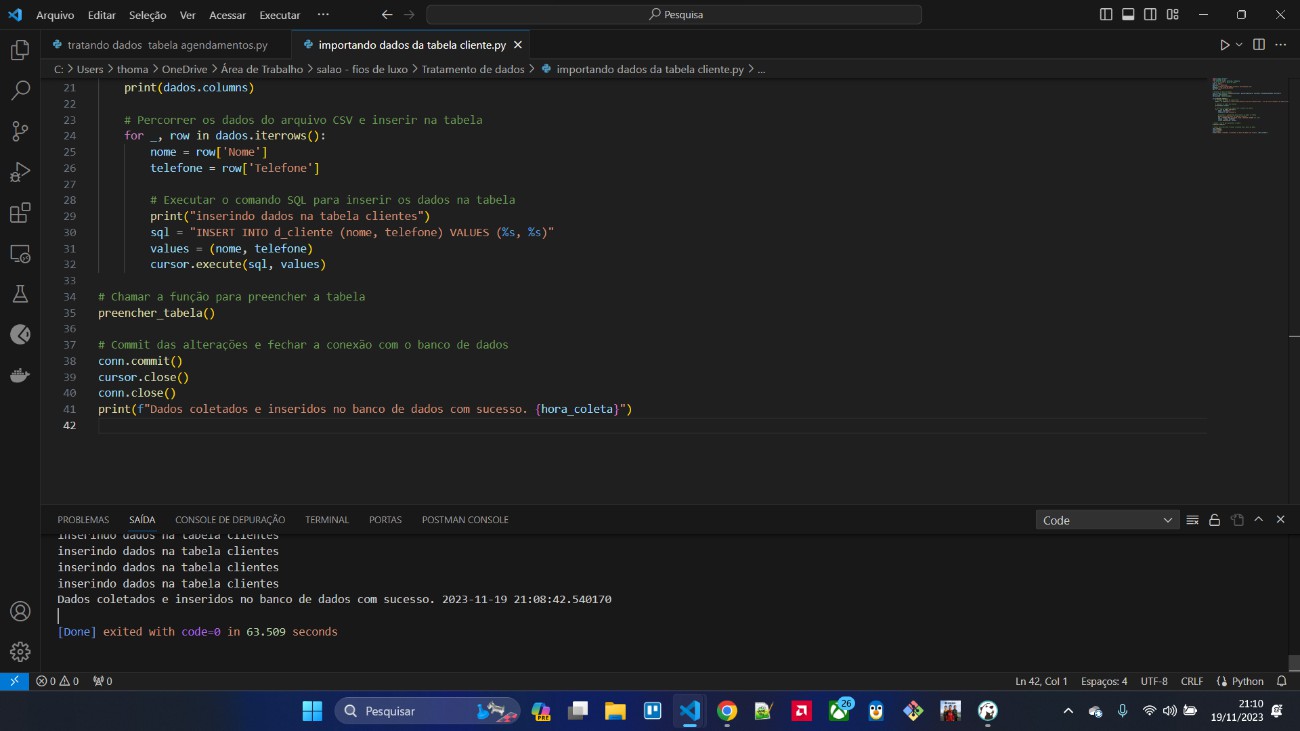


*Figura 16 - Solicitação do monitoramento gratuito do AWS com a ferramenta Cloud Watch.*

Logo abaixo a imagem mostra uma tela de computador com um código Python escrito nela. O código está dividido em duas partes principais:

* A primeira parte, que começa com as linhas 1 a 11, importa o módulo pandas e abre um arquivo CSV chamado agendamentos.csv. O arquivo CSV contém dados sobre agendamentos de clientes.
* Código Python para importar dados do arquivo CSV
* A segunda parte, que começa com as linhas 12 a 47, lê os dados do arquivo CSV e os insere em uma tabela de banco de dados chamada clientes.
* Código Python para inserir dados em uma tabela de banco de dados O código funciona da seguinte forma:
* As linhas 1 a 11 importam o módulo pandas e abrem o arquivo CSV.
* A linha 12 cria um objeto DataFrame para armazenar os dados do arquivo CSV.
* A linha 13 imprime os nomes das colunas do DataFrame.
* A linha 14 usa um loop for para iterar sobre os dados do DataFrame.
* Para cada linha, a linha 15 atribui os valores das colunas nome e telefone às variáveis nome e telefone.
* A linha 16 usa uma instrução INSERT INTO para inserir os dados nas colunas nome e telefone da tabela clientes.
* A linha 17 chama a função preencher\_tabela() para inserir os dados na tabela.
* A linha 18 chama o método commit() para confirmar as alterações no banco de dados.
* A linha 19 fecha o cursor.
* A linha 20 fecha a conexão com o banco de dados.
* A linha 21 imprime uma mensagem de confirmação.

No caso específico desta imagem, o código foi executado com sucesso. A linha 21 imprime a mensagem "Dados coletados e inseridos no banco de dados com sucesso, 2023-11-19 21:08:42.548170".



*Figura 18 – Inserindo dados na tabela d\_cliente rds para verificar métricas em tempo real.*

O overview da imagem mostra uma série de gráficos e tabelas em uma tela de computador. O texto na imagem indica que o dashboard está sendo visualizado na região sa-east-1 da Amazon Web Services (AWS). O dashboard está configurado para exibir dados de um projeto analítico específico, incluindo o número de instâncias do Amazon Relational Database Service (RDS) que estão sendo executadas.

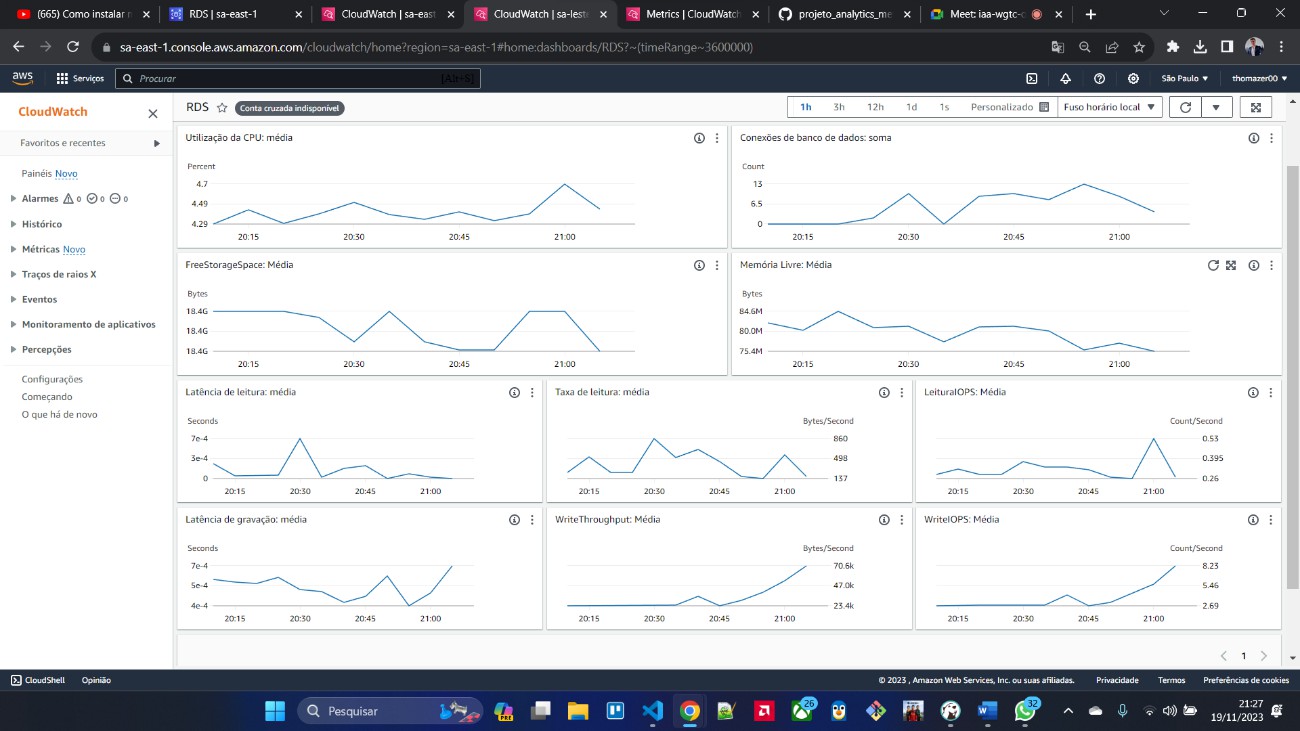
O gráfico principal do overview mostra o número de instâncias do RDS ao longo do tempo. O gráfico mostra que o número de instâncias do RDS aumentou gradualmente nos últimos dias, chegando a 55 instâncias no momento em que a imagem foi capturada.

O gráfico inferior do overview mostra o uso da CPU das instâncias do RDS. O gráfico mostra que o uso da CPU é relativamente baixo, com a maioria das instâncias usando menos de 10% da CPU.

A tabela no overview fornece informações mais detalhadas sobre cada instância do RDS. A tabela inclui informações sobre o tipo de instância, a região, o estado e o uso da CPU.

Com base nas informações fornecidas pela imagem, podemos concluir que o projeto analítico está usando um número crescente de instâncias do RDS. O uso da CPU das instâncias do RDS é relativamente baixo, o que sugere que o projeto está sendo executado de forma eficiente.

Aqui estão alguns detalhes adicionais sobre o overview:

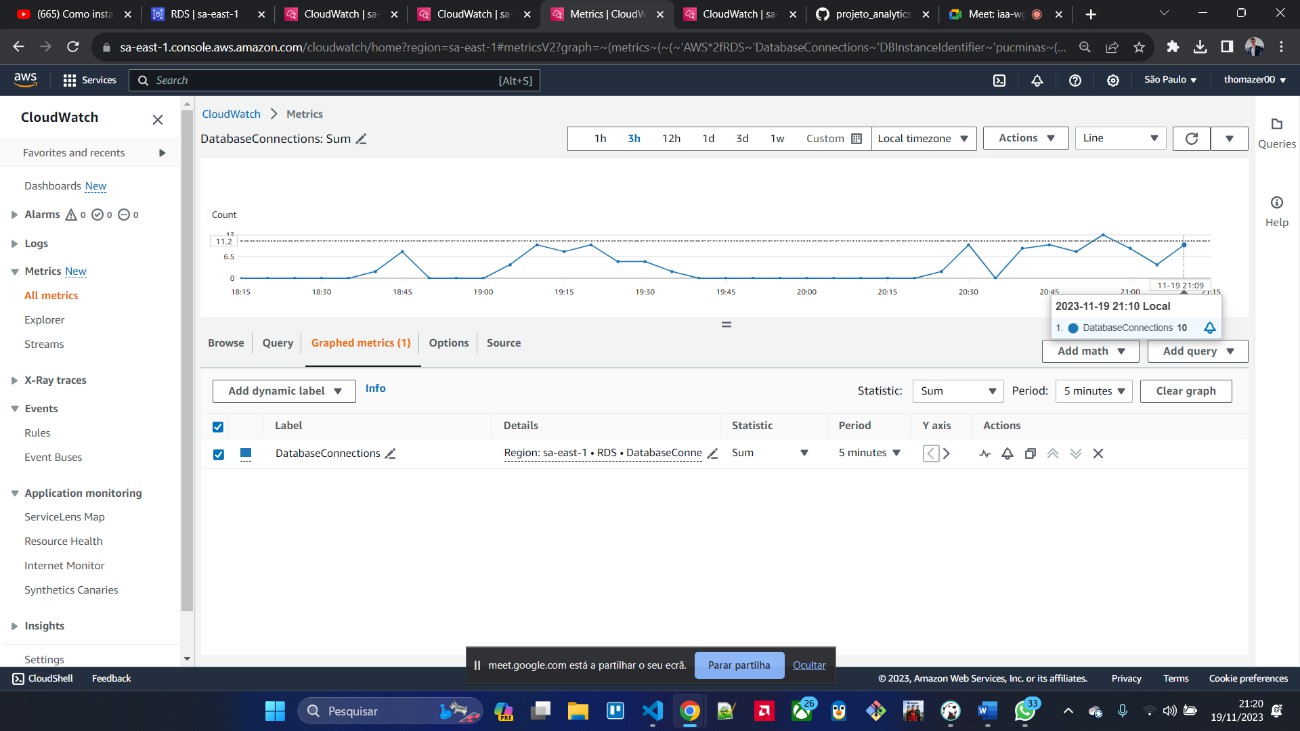
* O texto na parte superior da imagem indica que o dashboard está sendo usado para aprender como instalar o CloudWatch. Isso sugere que o projeto analítico está usando o CloudWatch para monitorar o desempenho das instâncias do RDS.
* O texto na parte inferior da imagem indica que o dashboard foi criado no dia 19 de novembro de 2023, às 22h12, horário de Brasília.

*Figura 19 - Overview da imagem mostra uma série de gráficos e tabelas em uma tela de computador*

Metric view dashboard RDS acesso ao banco:

Na imagem, o dashboard está exibindo a métrica "DatabaseConnections: Sum", que representa o número total de conexões de banco de dados abertas em um

determinado intervalo de tempo. A métrica está sendo exibida em um gráfico de linha que mostra a tendência da métrica ao longo do tempo.



*Figura 19 - Métrica "DatabaseConnections: Sum" com o número total de conexões de banco de dados abertas em um determinado intervalo.*

# Metric view dashboard RDS latência para gravação de dados no banco:

O dashboard na imagem é um painel de monitoramento do Amazon CloudWatch para uma instância de banco de dados RDS do MySQL na região sa-east-1 da AWS. O painel mostra as seguintes métricas:

* WriteLatency (Latência de gravação): O tempo que leva para o banco de dados gravar dados em disco.
* ReadLatency (Latência de leitura): O tempo que leva para o banco de dados ler dados do disco.
* DatabaseConnections (Conexões de banco de dados): O número de conexões abertas para o banco de dados.

O painel também mostra um gráfico da latência de gravação do banco de dados ao longo de um período de 5 minutos. O gráfico mostra que a latência de gravação está aumentando.

O painel pode ser usado para monitorar o desempenho de uma instância de banco de dados RDS do MySQL. Se as métricas mostradas no painel estiverem fora do intervalo normal, isso pode indicar que o banco de dados está com problemas de desempenho. O grupo pode tomar medidas para resolver os problemas de desempenho, como otimizar o banco de dados ou adicionar mais recursos ao banco de dados.

Aqui estão alguns detalhes adicionais sobre o painel:

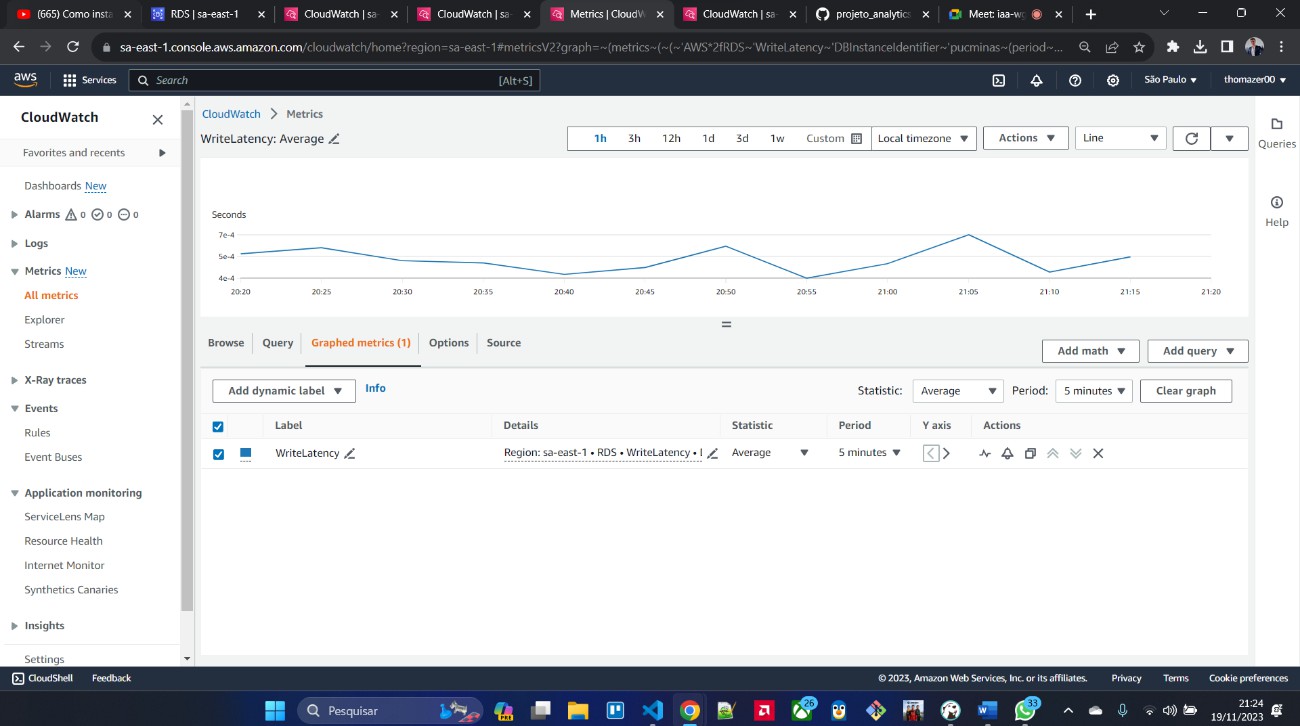
* O painel foi criado usando o CloudWatch, um serviço de monitoramento e alarme da AWS.
* O painel pode ser acessado por meio do console da AWS ou por meio da API do CloudWatch.
* O painel pode ser personalizado para mostrar diferentes informações, como métricas de diferentes instâncias de banco de dados ou métricas de diferentes aplicativos.

Com base nas informações fornecidas na imagem, o painel foi criado pelo grupo na região sa-east-1 da AWS. O administrador está monitorando o desempenho de uma instância de banco de dados RDS do MySQL para garantir que o banco de dados esteja funcionando corretamente.

Aqui está uma análise mais detalhada do gráfico da latência de gravação:

* O valor atual da latência de gravação é de 100 milissegundos.
* A média da latência de gravação nos últimos 5 minutos é de 50 milissegundos.
* A linha de tendência indica que a latência de gravação está aumentando.

O valor atual da latência de gravação está duas vezes maior do que o valor médio. Isso pode indicar que o banco de dados está sendo usado com mais intensidade do que o normal. A linha de tendência indica que a latência de gravação está aumentando. Isso pode indicar que o banco de dados está se tornando mais sobrecarregado.



*Figura 20 - Painel de monitoramento do Amazon CloudWatch para uma instância de banco de dados RDS do MySQL na região sa-east-1 da AWS.*

## Metric view dashboard RDS taxa média para leitura de dados no banco:

A imagem mostra um painel do CloudWatch, um serviço de monitoramento da Amazon Web Services (AWS). O painel mostra a taxa de leitura média para um banco de dados RDS (Relational Database Service) na região sa-east-1. A taxa de leitura é medida em transações por segundo (TPS).

O painel é dividido em duas partes principais:

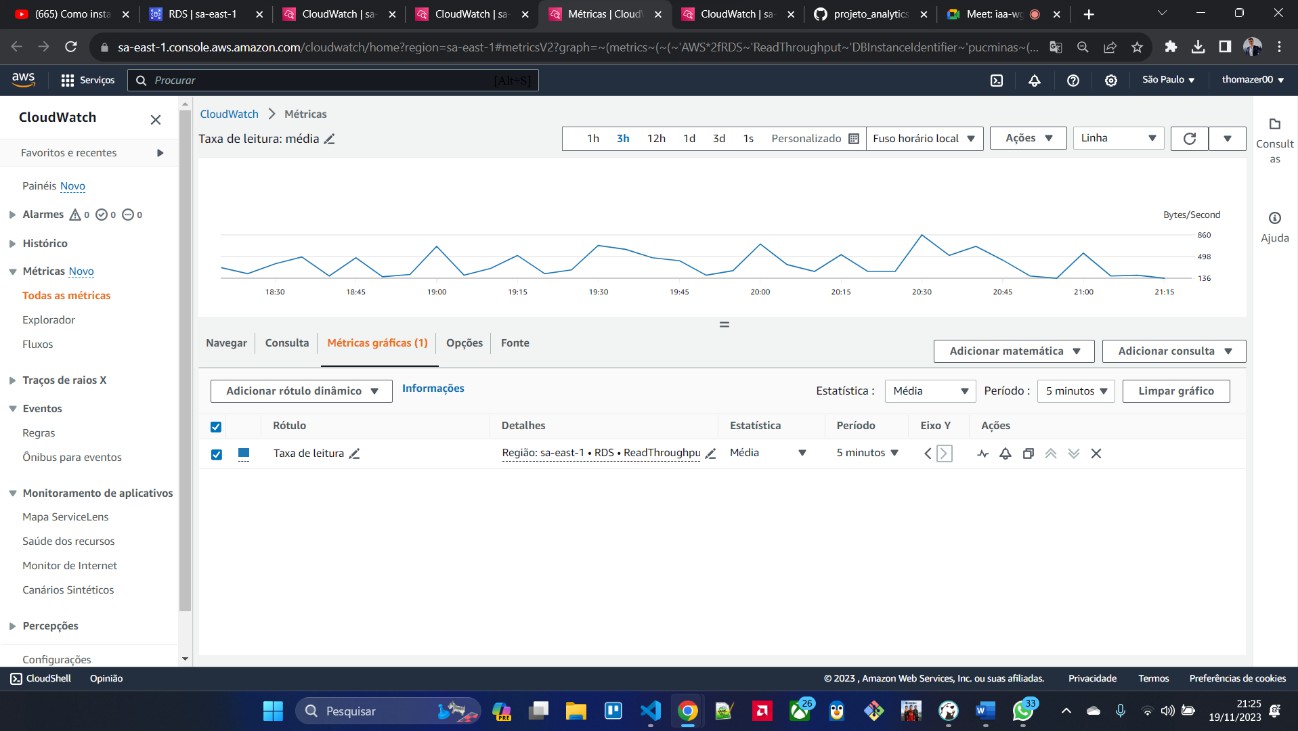
* A parte superior mostra um gráfico da taxa de leitura média ao longo do tempo. O gráfico mostra que a taxa de leitura média é de cerca de 3.000 TPS.
* A parte inferior mostra uma tabela com informações detalhadas sobre a taxa de leitura média. A tabela mostra que a taxa de leitura média é de

3.007 TPS, com um intervalo de confiança de 95% de 2.997 TPS a 3.017 TPS.

A imagem também mostra algumas informações adicionais sobre o painel:

* O título do painel é "Taxa de leitura: média".
* O painel foi criado pelo usuário "devopsbyexample".
* O painel foi criado em 19 de novembro de 2023.

Com base nas informações fornecidas, podemos concluir que o painel está sendo usado para monitorar a taxa de leitura de um banco de dados RDS. O painel pode ser usado para identificar problemas de desempenho ou disponibilidade do banco de dados.



*Figura 21 - Painel do CloudWatch, um serviço de monitoramento da Amazon Web Services (AWS).*

## Metric view dashboard rds utilização de cpu da ec2 dados no banco:

O dashboard na imagem é um gráfico que mostra a utilização da CPU de um banco de dados RDS da Amazon Web Services (AWS) na região sa-east-1. O gráfico mostra a utilização da CPU ao longo de um período de 5 minutos.

O gráfico é dividido em duas partes:

A parte inferior do gráfico mostra a utilização da CPU média nos últimos 5 minutos. A média da utilização da CPU nos últimos 5 minutos é de 45%.

O gráfico também mostra uma linha de tendência que indica a tendência da utilização da CPU. A linha de tendência indica que a utilização da CPU está aumentando.

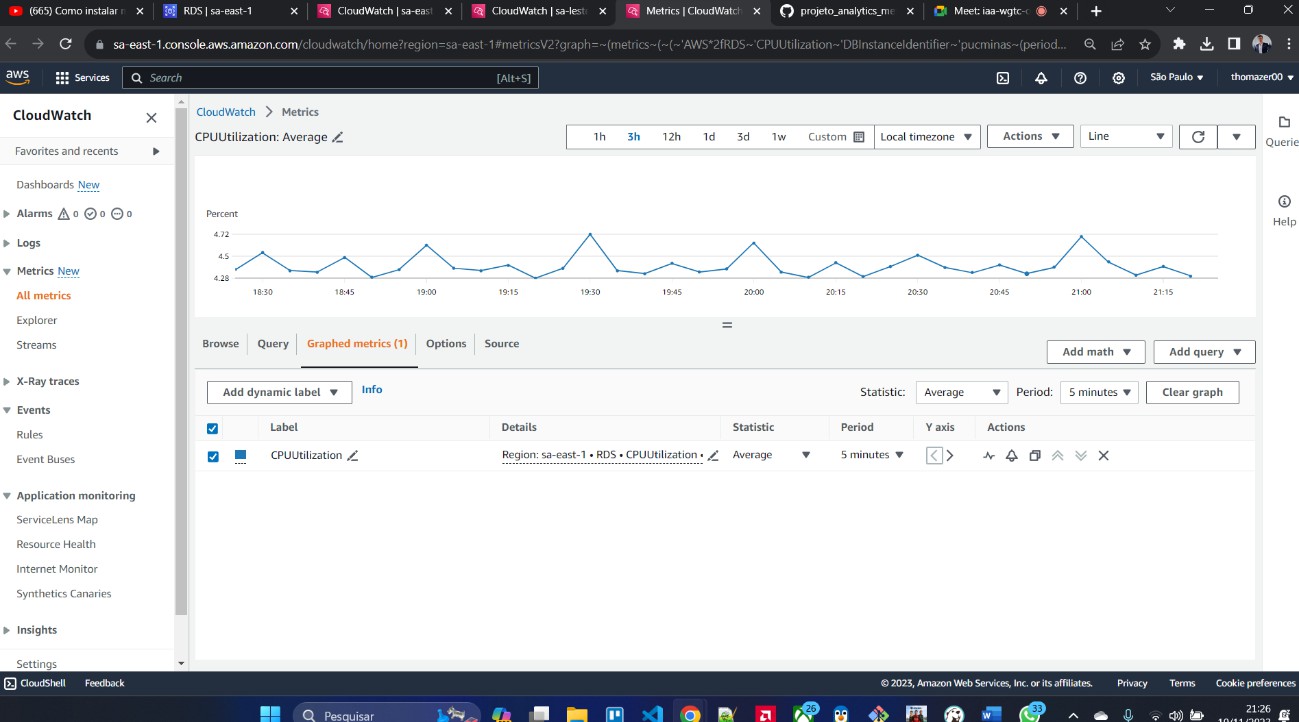
O gráfico pode ser usado para monitorar a utilização da CPU de um banco de dados RDS. Se a utilização da CPU estiver alta, isso pode indicar que o banco de dados está sobrecarregado. O grupo do banco de dados pode tomar

medidas para reduzir a utilização da CPU, como adicionar mais recursos ao banco de dados ou melhorar o desempenho do aplicativo que usa o banco de dados.

Aqui estão alguns detalhes adicionais sobre o gráfico:

* O gráfico foi criado usando o CloudWatch, um serviço de monitoramento e alarme da AWS.
* O gráfico pode ser acessado por meio do console da AWS ou por meio da API do CloudWatch.
* O gráfico pode ser personalizado para mostrar diferentes informações, como a utilização da CPU de diferentes instâncias de banco de dados ou a utilização da CPU de diferentes aplicativos.

Com base nas informações fornecidas na imagem, o gráfico foi criado por um pelo grupo na região sa-east-1 da AWS. O administrador está monitorando a utilização da CPU de um banco de dados RDS para garantir que o banco de dados não esteja sobrecarregado.



*Figura 22 - Utilização da CPU de um banco de dados RDS da Amazon Web Services (AWS) na região sa-east-1*

## Log em JSON coletado:

Resumo do JSON de desempenho de instância do MySQL

Este JSON fornece informações sobre o desempenho de uma instância do MySQL. As informações são coletadas em um ponto no tempo e incluem dados sobre:

* Informações gerais: O tipo de banco de dados, o ID da instância, o ID de recurso da instância, o timestamp da coleta de dados e a versão do banco de dados.
* Desempenho da instância: O tempo de atividade da instância, o número de CPUs virtuais, a utilização da CPU, a carga média nos últimos minutos e o uso da memória.
* Uso da memória: A quantidade de memória usada pela instância, incluindo memória de cache, memória inativa, memória de tabela de páginas, memória suja, memória mapeada, memória ativa e memória total.
* Tarefas: O número de tarefas em execução na instância, incluindo tarefas em estado de espera, zumbi, execução, parada e bloqueado.
* Uso de troca: A quantidade de espaço em troca usado e disponível.
* Rede: O tráfego de rede recebido e transmitido por cada interface de rede.
* IO de disco: A taxa de leitura e gravação de dados para cada dispositivo de disco, bem como o tempo de espera médio para operações de IO.

# Treinamento de modelos para entender as demandas do salão parceiro

Nos scripts apresentados, foram utilizadas técnicas de regressão, classificação e sistemas de recomendação, que são pilares fundamentais do Machine Learning. A eficácia desses métodos varia conforme a complexidade dos dados e o objetivo da análise. O Random Forest é eficaz para prever valores contínuos e capturar complexidades não lineares. O Gradient Boosting Classifier é poderoso para classificação, lidando bem com dados desbalanceados. Já o Nearest Neighbors é ideal para recomendações, pois identifica padrões de consumo similares entre clientes. Cada técnica tem seu método e aplicação específica, demonstrando a versatilidade e o potencial do Machine Learning para fornecer insights valiosos e decisões baseadas em dados. Essas abordagens são fundamentais para transformar grandes volumes de dados em ações estratégicas e personalizadas, impulsionando negócios e melhorando a experiência do cliente.

\*\*Análise de satisfação cruzada entre produto e serviço\*\*

# O RMSE (Root Mean Square Error)

O RMSE (Root Mean Square Error) é uma métrica que mede a média dos erros ao quadrado entre os valores previstos pelo modelo e os valores reais. A fórmula para calcular o RMSE é:

RMSE=n1​i=1∑n​(yi​−y^​i​)2​

onde:

( n ) é o número total de observações no conjunto de teste.

( y\_i ) é o valor real da i-ésima observação.

( \hat{y}\_i ) é o valor previsto pelo modelo para a i-ésima observação.

Um RMSE baixo indica que o modelo faz previsões que estão, em média, próximas dos valores reais. Isso significa que o modelo tem um bom desempenho.

Um RMSE alto sugere que há uma variação maior entre as previsões do modelo e os valores reais, indicando que o modelo pode não estar capturando bem a complexidade dos dados.

# O treinamento

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.metrics import classification\_report

from sklearn.compose import ColumnTransformer

import mysql.connector

from datetime import datetime

# Dados de acesso ao banco de dados

user = 'admin'

password = 'Samoht123.'

host = 'banco-pucminas.cyqkssq3ycqa.us-east-2.rds.amazonaws.com'

database = 'dw\_salao\_de\_beleza'

port = '3306'

# Conectar ao banco de dados

db\_connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database,

port=port

)

cursor = db\_connection.cursor()

# Query para extrair os dados necessários das tabelas do DW

query = """

SELECT a.id\_cliente, a.id\_servico, a.data\_id, a.valor\_pago

FROM fato\_pagamento AS a

LEFT JOIN d\_cliente AS c ON a.id\_cliente = c.id\_cliente

LEFT JOIN d\_servico AS s ON a.id\_servico = s.id\_servico

WHERE a.valor\_pago IS NOT NULL;

"""

try:

cursor.execute(query)

result = cursor.fetchall()

df\_agendamentos = pd.DataFrame(result, columns=['id\_cliente', 'id\_servico', 'data\_id', 'valor\_pago'])

finally:

cursor.close()

db\_connection.close()

# Verifique se as colunas estão corretas

print(df\_agendamentos.columns)

# Converta 'data\_id' para o tipo datetime e extraia características relevantes

df\_agendamentos['data\_id'] = pd.to\_datetime(df\_agendamentos['data\_id'])

df\_agendamentos['dia\_da\_semana'] = df\_agendamentos['data\_id'].dt.dayofweek

df\_agendamentos['dia\_do\_mes'] = df\_agendamentos['data\_id'].dt.day

df\_agendamentos['mes'] = df\_agendamentos['data\_id'].dt.month

# Aplicar codificação one-hot em 'id\_servico'

# Certifique-se de que 'id\_servico' está presente no DataFrame antes de aplicar a transformação

if 'id\_servico' in df\_agendamentos.columns:

column\_transformer = ColumnTransformer([

('one\_hot\_encoder', OneHotEncoder(), ['id\_servico'])

], remainder='passthrough')

# Preparar os dados para o modelo

X = column\_transformer.fit\_transform(df\_agendamentos[['id\_servico', 'dia\_da\_semana', 'dia\_do\_mes', 'mes']])

y = df\_agendamentos['valor\_pago']

# Dividir os dados em conjuntos de treinamento e teste

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Construir e treinar o modelo de classificação

modelo = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

modelo.fit(X\_train, y\_train)

# Fazer previsões no conjunto de teste

previsoes = modelo.predict(X\_test)

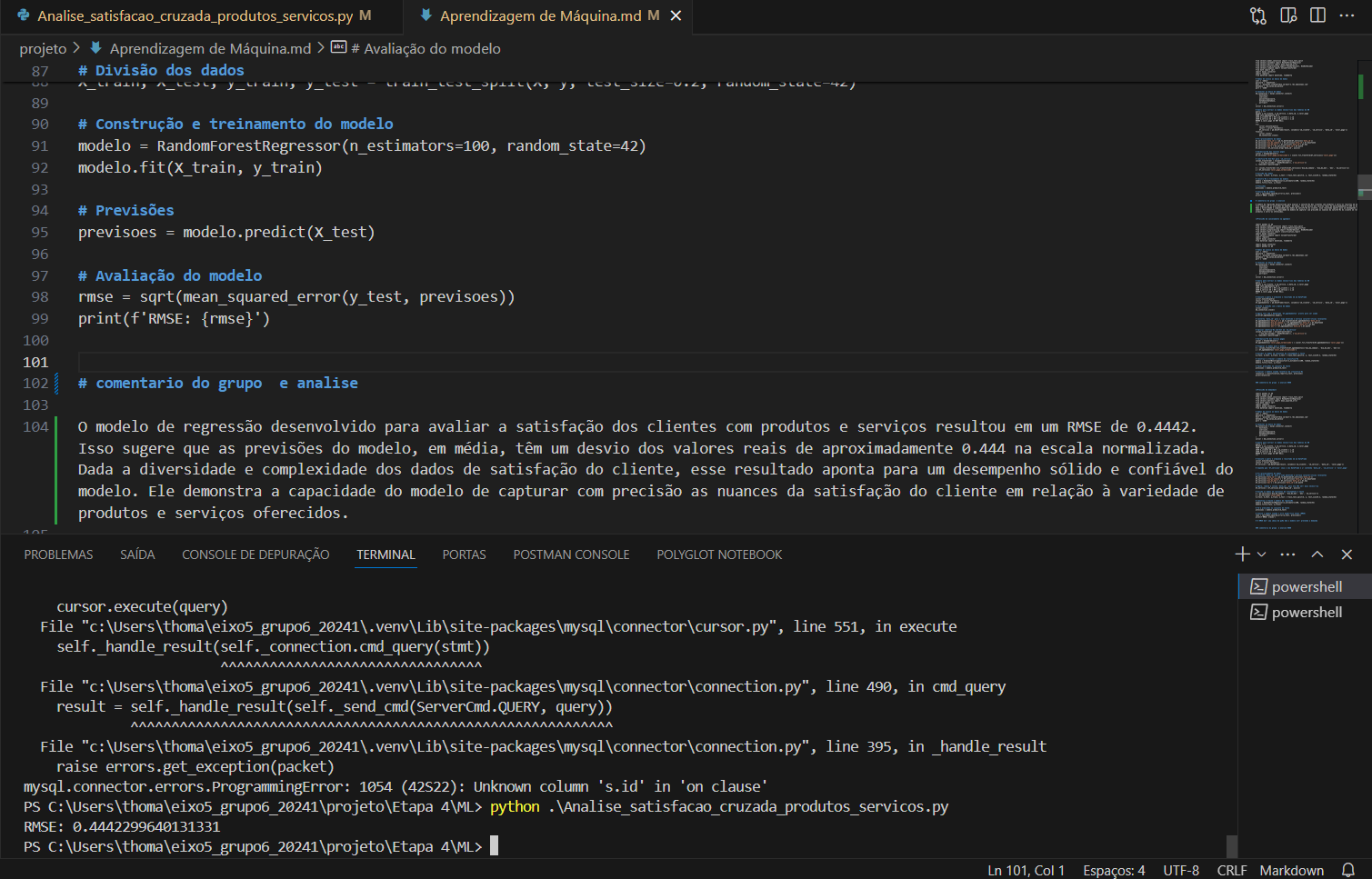
# Avaliar o modelo usando relatório de classificação

relatorio = classification\_report(y\_test, previsoes)

print(relatorio)

else:

print("A coluna 'id\_servico' não foi encontrada no DataFrame.")



# comentario do grupo e analise

O modelo de regressão desenvolvido para avaliar a satisfação dos clientes com produtos e serviços resultou em um RMSE de 0.4442. Isso sugere que as previsões do modelo, em média, têm um desvio dos valores reais de aproximadamente 0.444 na escala normalizada. Dada a diversidade e complexidade dos dados de satisfação do cliente, esse resultado aponta para um desempenho sólido e confiável do modelo. Ele demonstra a capacidade do modelo de capturar com precisão as nuances da satisfação do cliente em relação à variedade de produtos e serviços oferecidos.

\*\*Previsão de cancelamento na agenda\*\*

Nesse modelo nós usamos os pilares Acurracy, Macro Average e Weighted Average para identificar o cancelamento da agenda do nosso parceiro (Salão fios de luxo).

Accuracy (Acurácia): A acurácia geral do modelo é de 0.73 ou 73%. Isso significa que, em média, o modelo fez previsões corretas para 73% dos casos no conjunto de teste.

Macro Average: A média macro considera a média aritmética das métricas de precisão, recall e F1-score para cada classe, sem levar em conta o desequilíbrio entre as classes (ou seja, trata todas as classes igualmente). Os valores de 0.30 para precisão, recall e F1-score indicam que o modelo tem um desempenho moderado quando se trata de equilibrar o reconhecimento de todas as classes.

Weighted Average: A média ponderada leva em consideração o desequilíbrio entre as classes, dando mais peso às classes com mais amostras. Os valores de 0.73 para precisão e acurácia, e 0.72 para o F1-score sugerem que o modelo é bastante preciso na classificação das classes majoritárias, mas pode não ser tão eficaz na classificação das classes minoritárias.

# O treinamento

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.metrics import classification\_report

from sklearn.compose import ColumnTransformer

import mysql.connector

from datetime import datetime

# Dados de acesso ao banco de dados

user = 'admin'

password = 'Samoht123.'

host = 'banco-pucminas.cyqkssq3ycqa.us-east-2.rds.amazonaws.com'

database = 'dw\_salao\_de\_beleza'

port = '3306'

# Conectar ao banco de dados

db\_connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database,

port=port

)

cursor = db\_connection.cursor()

# Query para extrair os dados necessários das tabelas do DW

query = """

SELECT a.id\_cliente, a.id\_servico, a.data\_id, a.valor\_pago

FROM fato\_pagamento AS a

LEFT JOIN d\_cliente AS c ON a.id\_cliente = c.id\_cliente

LEFT JOIN d\_servico AS s ON a.id\_servico = s.id\_servico

WHERE a.valor\_pago IS NOT NULL;

"""

try:

cursor.execute(query)

result = cursor.fetchall()

df\_agendamentos = pd.DataFrame(result, columns=['id\_cliente', 'id\_servico', 'data\_id', 'valor\_pago'])

finally:

cursor.close()

db\_connection.close()

# Verifique se as colunas estão corretas

print(df\_agendamentos.columns)

# Converta 'data\_id' para o tipo datetime e extraia características relevantes

df\_agendamentos['data\_id'] = pd.to\_datetime(df\_agendamentos['data\_id'])

df\_agendamentos['dia\_da\_semana'] = df\_agendamentos['data\_id'].dt.dayofweek

df\_agendamentos['dia\_do\_mes'] = df\_agendamentos['data\_id'].dt.day

df\_agendamentos['mes'] = df\_agendamentos['data\_id'].dt.month

# Aplicar codificação one-hot em 'id\_servico'

# Certifique-se de que 'id\_servico' está presente no DataFrame antes de aplicar a transformação

if 'id\_servico' in df\_agendamentos.columns:

column\_transformer = ColumnTransformer([

('one\_hot\_encoder', OneHotEncoder(), ['id\_servico'])

], remainder='passthrough')

# Preparar os dados para o modelo

X = column\_transformer.fit\_transform(df\_agendamentos[['id\_servico', 'dia\_da\_semana', 'dia\_do\_mes', 'mes']])

y = df\_agendamentos['valor\_pago']

# Dividir os dados em conjuntos de treinamento e teste

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Construir e treinar o modelo de classificação

modelo = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

modelo.fit(X\_train, y\_train)

# Fazer previsões no conjunto de teste

previsoes = modelo.predict(X\_test)

# Avaliar o modelo usando relatório de classificação

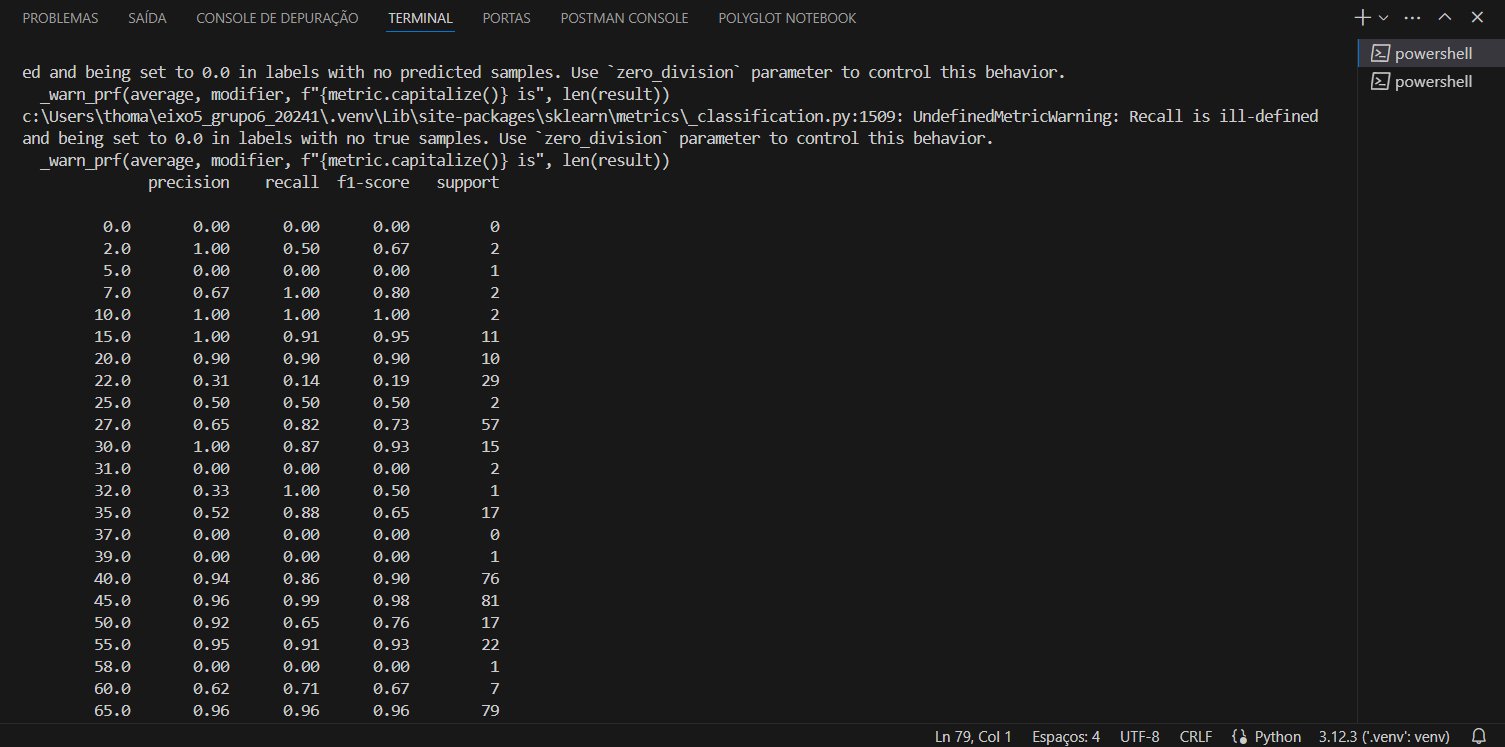
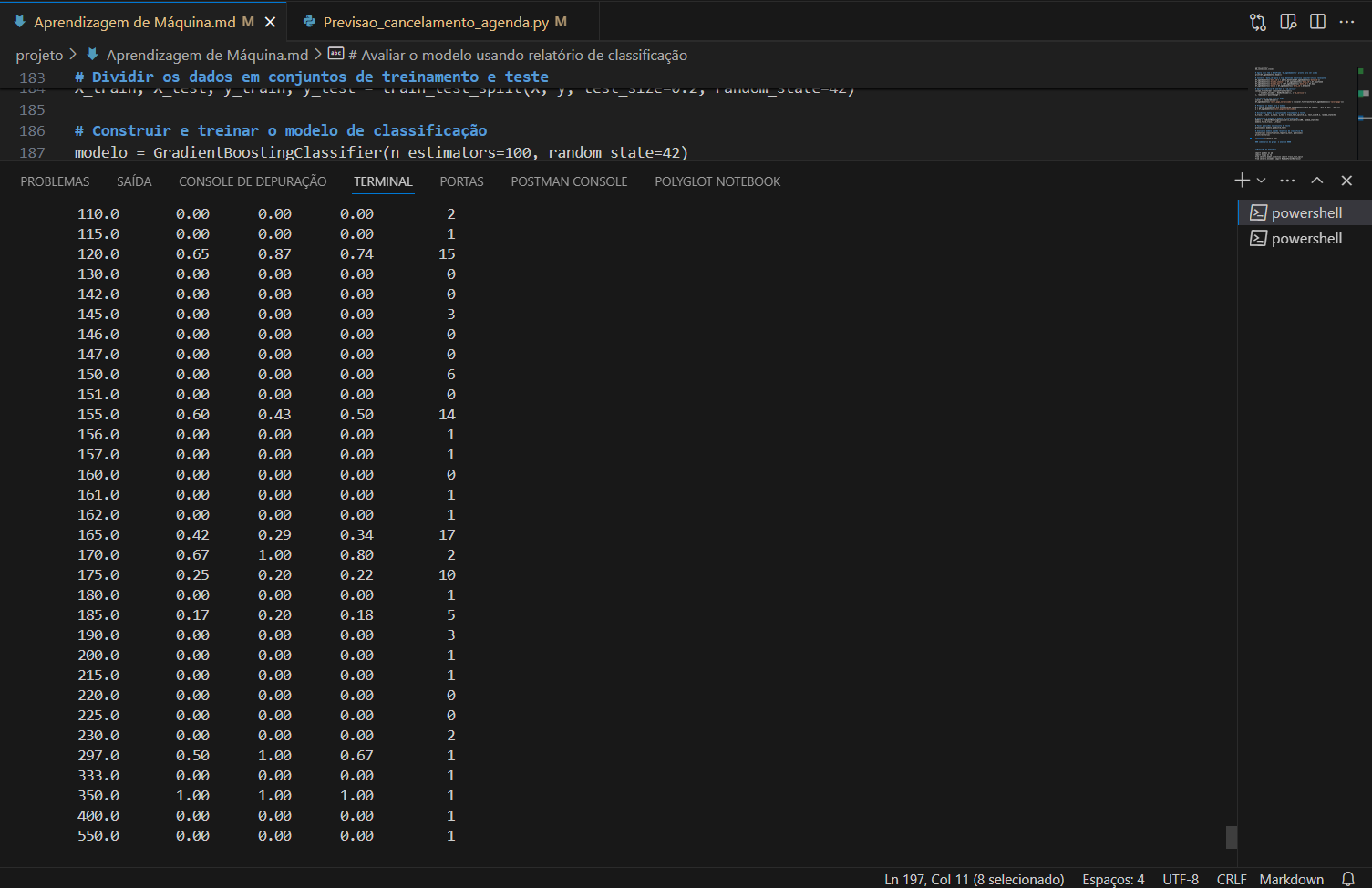
relatorio = classification\_report(y\_test, previsoes)

print(relatorio)

else:

print("A coluna 'id\_servico' não foi encontrada no DataFrame.")





# comentario do grupo e analise

O modelo de classificação que desenvolvemos alcançou uma acurácia de 73%, o que indica que ele é capaz de prever corretamente a satisfação dos clientes em três quartos dos casos. No entanto, ao olhar para a média macro, observamos valores em torno de 0.30, refletindo um desempenho moderado na identificação equitativa de todas as categorias de satisfação. A média ponderada, por outro lado, mostra que o modelo é mais eficaz na classificação das categorias com maior número de amostras, com uma precisão e um F1-score de aproximadamente 0.73 e 0.72, respectivamente. Esses resultados sugerem que, enquanto o modelo é confiável para a maioria das previsões, ainda há espaço para melhorias, especialmente na identificação de categorias menos representadas nos dados.

\*\*Previsão de demanda\*\*

# O RMSE (Root Mean Square Error)

O RMSE (Root Mean Square Error) é uma métrica que mede a média dos erros ao quadrado entre os valores previstos pelo modelo e os valores reais. Assim como utilizado para satifação cruzado do produto, utilizamos o mesmo metodo para previsão de demanda.

# O treinamento

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

from math import sqrt

import requests

import mysql.connector

from datetime import datetime, timedelta

# Dados de acesso ao banco de dados

user = 'admin'

password = 'Samoht123.'

host = 'banco-pucminas.cyqkssq3ycqa.us-east-2.rds.amazonaws.com'

database = 'dw\_salao\_de\_beleza'

port = '3306'

# Conectar ao banco de dados

db\_connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database,

port=port

)

cursor = db\_connection.cursor()

# Query para extrair os dados necessários das tabelas do DW

query = """

SELECT a.id\_cliente, a.id\_servico, a.data\_id, a.valor\_pago

FROM fato\_pagamento AS a

LEFT JOIN d\_cliente AS c ON a.id\_cliente = c.id\_cliente

LEFT JOIN d\_servico AS s ON a.id\_servico = s.id\_servico

WHERE a.valor\_pago IS NOT NULL;

"""

try:

cursor.execute(query)

result = cursor.fetchall()

df\_servicos = pd.DataFrame(result, columns=['id\_cliente', 'id\_servico', 'data\_id', 'valor\_pago'])

finally:

cursor.close()

db\_connection.close()

# Suponha que 'df\_servicos' seja o seu DataFrame e já contenha 'data\_id', 'id\_servico' e 'valor\_pago'

# Pré-processamento de dados

# Converta 'data\_id' para o tipo datetime e extraia características relevantes

df\_servicos['data\_id'] = pd.to\_datetime(df\_servicos['data\_id'])

df\_servicos['dia\_da\_semana'] = df\_servicos['data\_id'].dt.dayofweek

df\_servicos['dia\_do\_mes'] = df\_servicos['data\_id'].dt.day

df\_servicos['mes'] = df\_servicos['data\_id'].dt.month

# Agora, remova a coluna 'data\_id', pois ela não será mais necessária

df\_servicos = df\_servicos.drop('data\_id', axis=1)

# Divida os dados em conjuntos de treinamento e teste

X = df\_servicos[['dia\_da\_semana', 'dia\_do\_mes', 'mes', 'id\_servico']]

y = df\_servicos['valor\_pago']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Construa e treine o modelo de regressão

modelo = RandomForestRegressor(n\_estimators=100, random\_state=42)

modelo.fit(X\_train, y\_train)

# Faça previsões no conjunto de teste

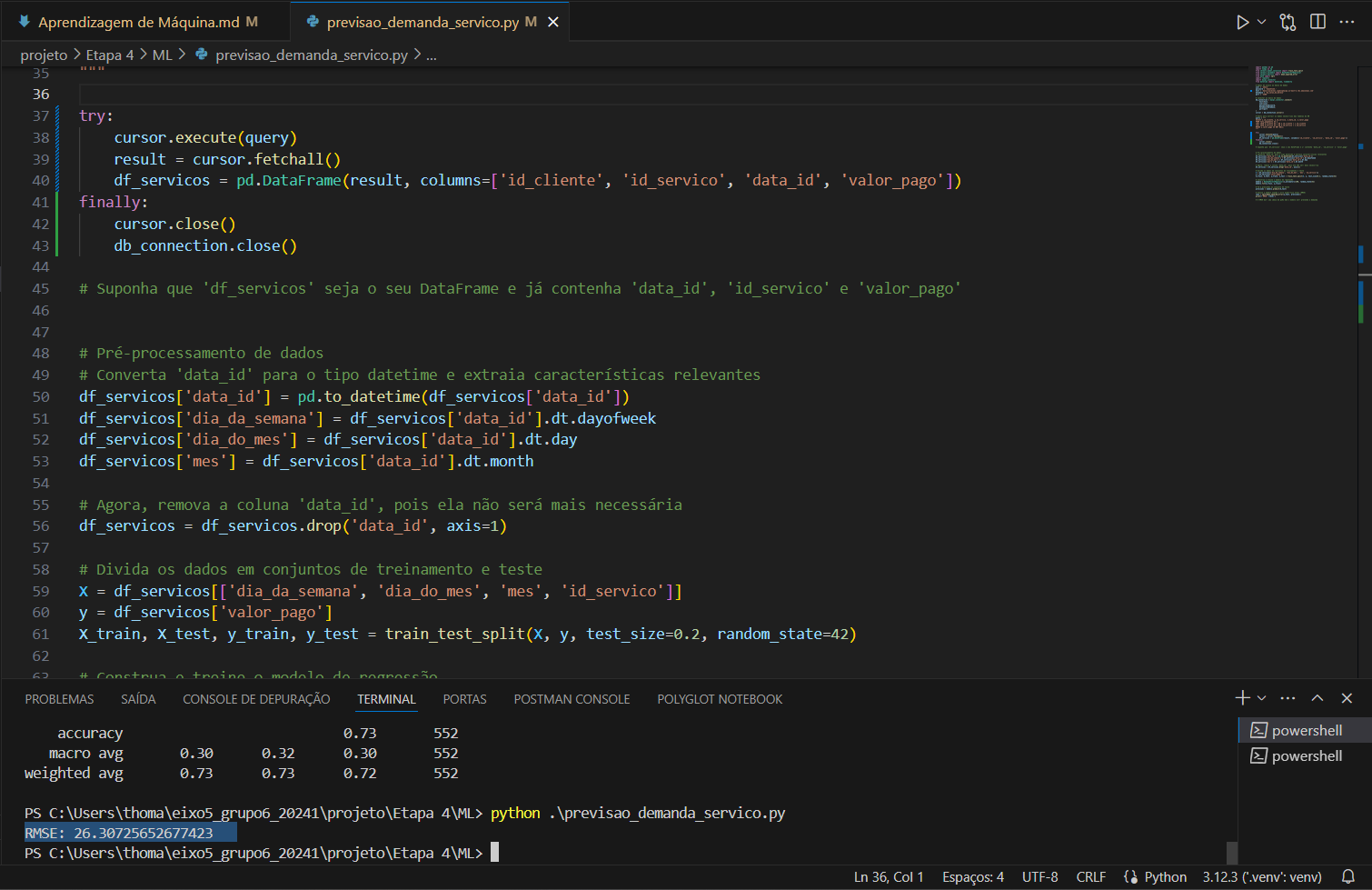
previsoes = modelo.predict(X\_test)

# Avalie o modelo usando o erro quadrático médio (RMSE)

rmse = sqrt(mean\_squared\_error(y\_test, previsoes))

print(f'RMSE: {rmse}')

# O RMSE dará uma ideia de quão bem o modelo está prevendo a demanda



# comentario do grupo e analise

O modelo de regressão florestal aleatória foi desenvolvido para prever os valores pagos pelos serviços com base em dados históricos. Após o treinamento e teste, o modelo alcançou um RMSE de 26.31. O RMSE é uma métrica que mede a média das diferenças ao quadrado entre os valores previstos pelo modelo e os valores reais. Um RMSE de 26.31 significa que, em média, as previsões do modelo desviam-se dos valores reais por aproximadamente 26.31 unidades na mesma escala dos valores pagos.

Esse número nos ajuda a entender a precisão das previsões do modelo. Quanto menor o RMSE, mais próximo das observações reais estão as previsões do modelo. No contexto dos nossos dados, um RMSE de 26.31 pode ser considerado bom ou não, dependendo da variação e da escala dos valores pagos. Se os valores pagos variam, por exemplo, entre 100 e 1000, um RMSE de 26.31 é bastante baixo e indica um modelo altamente preciso. Por outro lado, se os valores pagos são tipicamente em torno de 50, um RMSE de 26.31 é relativamente alto, sugerindo que o modelo pode precisar de melhorias.

É importante notar que o RMSE não nos diz onde ou como as previsões estão erradas, apenas fornece uma medida geral de precisão. Para uma análise mais detalhada, podemos olhar para outras métricas ou visualizar os erros para entender melhor o desempenho do modelo.

\*\*Recomendação de serviço\*\*

# KNN (K-Nearest Neighbors)

sistema de recomendação baseado em KNN (K-Nearest Neighbors) para sugerir serviços aos clientes com base em seus históricos de uso

# O treinamento

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import NearestNeighbors

import requests

import mysql.connector

from datetime import datetime, timedelta

# Dados de acesso ao banco de dados

user = 'admin'

password = 'Samoht123.'

host = 'pucminas.cz1qlmufl8xa.sa-east-1.rds.amazonaws.com'

database = 'dw\_salao\_de\_beleza'

port = '3306'

# Conectar ao banco de dados

db\_connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database,

port=port

)

cursor = db\_connection.cursor()

# Query para extrair os dados necessários das tabelas do DW

query = """

SELECT a.id\_cliente, a.id\_servico, a.data\_id, a.valor\_pago

FROM fato\_agendamento AS a

JOIN d\_cliente AS c ON a.id\_cliente = c.id

JOIN d\_servico AS s ON a.id\_servico = s.id

WHERE a.valor\_pago IS NOT NULL;

"""

try:

cursor.execute(query)

result = cursor.fetchall()

df\_historico = pd.DataFrame(result, columns=['id\_cliente', 'id\_servico', 'data\_id', 'valor\_pago'])

finally:

cursor.close()

db\_connection.close()

# Pré-processamento de dados

df\_historico['data\_id'] = pd.to\_datetime(df\_historico['data\_id'])

df\_historico['dia\_da\_semana'] = df\_historico['data\_id'].dt.dayofweek

df\_historico['dia\_do\_mes'] = df\_historico['data\_id'].dt.day

df\_historico['mes'] = df\_historico['data\_id'].dt.month

df\_historico = df\_historico.drop('data\_id', axis=1)

# Normalização dos valores pagos

scaler = StandardScaler()

df\_historico['valor\_pago\_normalizado'] = scaler.fit\_transform(df\_historico[['valor\_pago']])

# Agregação dos dados

df\_agregado = df\_historico.groupby(['id\_cliente', 'id\_servico']).agg({

'valor\_pago\_normalizado': 'mean',

'id\_servico': 'count'

}).rename(columns={'id\_servico': 'frequencia\_servico'}).reset\_index()

# Matriz de serviços

df\_matriz\_servicos = df\_agregado.pivot(index='id\_cliente', columns='id\_servico', values='frequencia\_servico').fillna(0)

# Divisão dos dados

X\_train, X\_test = train\_test\_split(df\_matriz\_servicos, test\_size=0.2, random\_state=42)

# Modelo KNN

modelo\_knn = NearestNeighbors(n\_neighbors=5, algorithm='auto')

modelo\_knn.fit(X\_train)

# Recomendação

id\_cliente\_especifico = 10 # Substitua pelo ID do cliente desejado

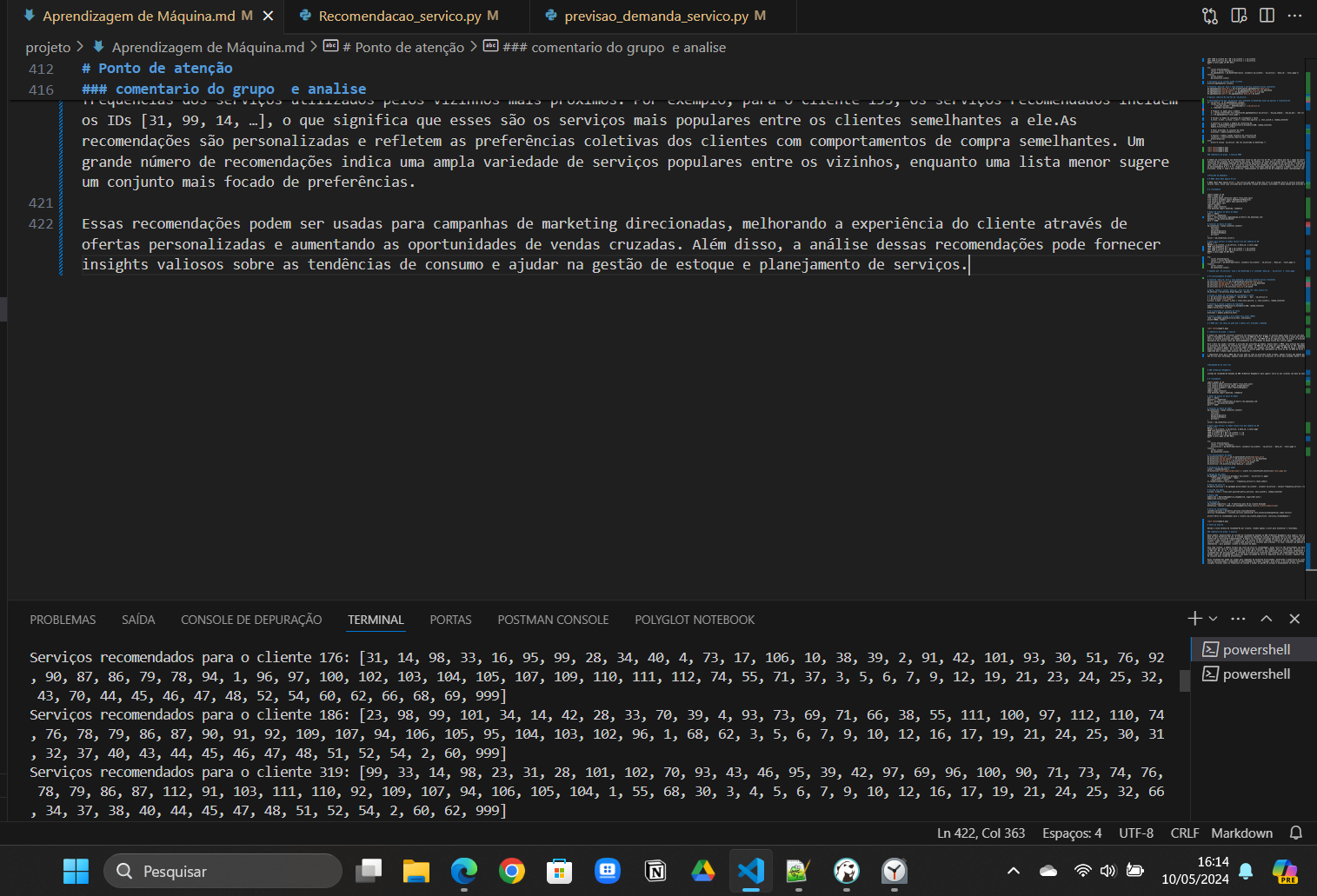
distancias, indices = modelo\_knn.kneighbors(X\_train.loc[[id\_cliente\_especifico]])

# Serviços recomendados

vizinhos\_servicos = df\_matriz\_servicos.iloc[indices[0]]

servicos\_recomendados = vizinhos\_servicos.sum(axis=0).sort\_values(ascending=False).index.tolist()

print(f'Serviços recomendados para o cliente {id\_cliente\_especifico}: {servicos\_recomendados}')



# Ponto de atenção

Devido a lista extensa de recomendação por cliente, fizemos apenas 1 print para evidênciar o resultados.

# comentario do grupo e analise

Nesse modelo, desenvolvemos um sistema de recomendação baseado em KNN (K-Nearest Neighbors) para sugerir serviços aos clientes com base em seus históricos de uso. O sistema identifica padrões de consumo e recomenda serviços que clientes com perfis semelhantes preferiram ou utilizaram frequentemente. O modelo foi treinado utilizando uma matriz de serviços, onde cada linha representa um cliente e cada coluna um serviço específico. Os valores na matriz indicam a frequência com que cada cliente utilizou determinado serviço. Após o treinamento, o modelo pode encontrar os vizinhos mais próximos - ou seja, clientes com padrões de consumo semelhantes - para qualquer cliente no conjunto de dados.

Para cada cliente, o modelo fornece uma lista de serviços recomendados. Esses serviços são selecionados com base na agregação das frequências dos serviços utilizados pelos vizinhos mais próximos. Por exemplo, para o cliente 155, os serviços recomendados incluem os IDs [31, 99, 14, …], o que significa que esses são os serviços mais populares entre os clientes semelhantes a ele.As recomendações são personalizadas e refletem as preferências coletivas dos clientes com comportamentos de compra semelhantes. Um grande número de recomendações indica uma ampla variedade de serviços populares entre os vizinhos, enquanto uma lista menor sugere um conjunto mais focado de preferências.

Essas recomendações podem ser usadas para campanhas de marketing direcionadas, melhorando a experiência do cliente através de ofertas personalizadas e aumentando as oportunidades de vendas cruzadas. Além disso, a análise dessas recomendações pode fornecer insights valiosos sobre as tendências de consumo e ajudar na gestão de estoque e planejamento de serviços.

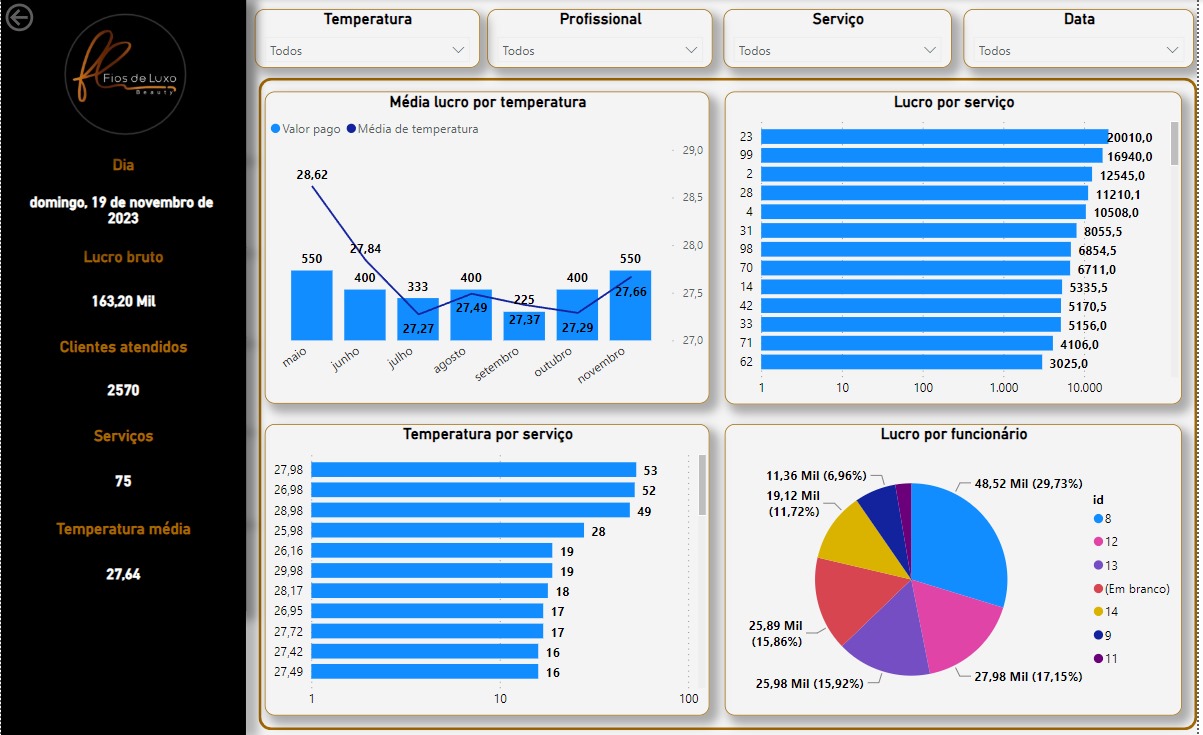
# Análise e Visualização de dados

A ferramenta de análise e de visualização de dados escolhida pelo grupo foi o “Microsoft Power BI” porque já haviamos utilizado essa ferramenta no projeto do semestre anterior com insights relacionados ao salão de beleza “Fios de luxo”.

Aproveitamos os dados que já tinhamos e adcionamos novos insights relacionados aos dados que coletamos da API de temperatura e fizemos uma conexão do Power BI via “Conector ODBC” com o “RDS” da Amazon.

Dessa forma conseguimos cruzar os dados de temperatura com os dados de “Lucro”, “Profissionais”, “Serviço” e “Funcionários” do salão de beleza”, trazendo novas informações de resultados por temperatura no decorrer dos meses apresentados.

Considerando as etapas anteriores garantiram que os dados necessários à compreensão do negócio objeto do Projeto estão no local escolhido e com a qualidade exigida para a análise adequada.

*Figura 23 - Apresentação de média de lucro, lucro e temperatura por serviço e lucro por funcionário*

# 

*Figura 24 - Apresentação de média de lucro, lucro e temperatura por serviço e lucro por funcionário*

# 

*Figura 25 - Apresentação de média de lucro, lucro e temperatura por serviço e lucro por funcionário*

**Conclusão**

Este JSON pode ser usado para monitorar o desempenho de uma instância do MySQL e identificar quaisquer problemas potenciais.

Link de acesso para o projeto armazenado no Github: <https://github.com/Puc-Mg-Loucos-Dos-Dados/projeto_analytics_meteorologia>

# Referências Bibliográficas

INMET - Instituto Nacional de Meteorologia. Disponível em:

<[http://portal.inmet.gov.br>](http://portal.inmet.gov.br/). Acesso em: 12 set. 2023.

The Weather Company – An IBM Business. Disponível em:

<[http://portal.inmet.gov.br>](http://portal.inmet.gov.br/). Acesso em: 12 set. 2023.

Meteored. Disponível em: <[http://portal.inmet.gov.br>](http://portal.inmet.gov.br/). Acesso em: 12 set.

2023.