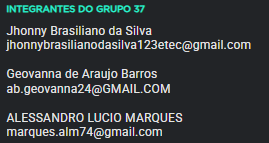
**ESTRUTURA DO TECH CHALLENGED 04**

**Alessandro Marques, Geovanna Barros, Jhonny Silva**

****

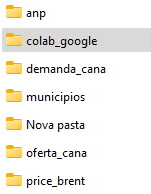
**Objetivo: Gerar modelo de série temporal para o Brent – Petróleo (db ipeadata)**

1. **Coleta de dados (planilha excel)**

Planilha excel da ANP (preços combustível e consumo de combustível)

Planilha excel da Conab (oferta de Cana-de-Açúcar e Etanol MG)

Planilha excel do Ipeadata (série histórica do preço do brent do petróleo bruto)



A pasta *colab\_google* subimos a base *IPEADATA* para trabalhar os modelos antes de incluí-los no *Vscode*. A municípios não utilizamos nesse projeto, pois as bases ANP (preços e consumo) e Conab (oferta) já continham as features municípios e estados.

1. **Transformação dos dados acima no KNIME**

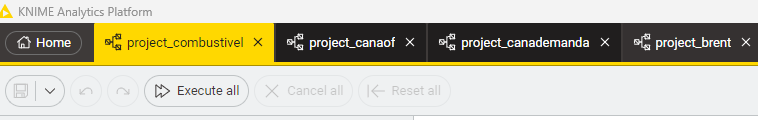
As planilhas fora base para 04 projetos no Knime

Oferta (cana – dados Conab)

Demanda (Anp – Preços Combustíveis)

Consumo (Anp - Consumo de Combustíveis)

Brent (IPEADATA – preços do petróleo)

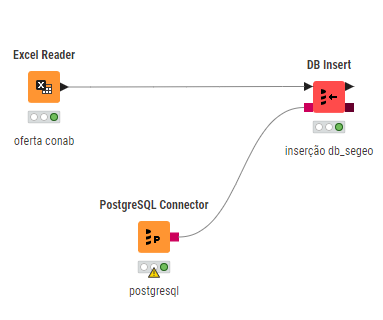


Subimos as planilhas respectivas a cada projeto. Fizemos a transformação de variáveis (features) com a alteração de nomes e tipo de variável. Após, incluímos os NODES de planilhas ***Insert*** e do bando de dados ***Postegresql***.

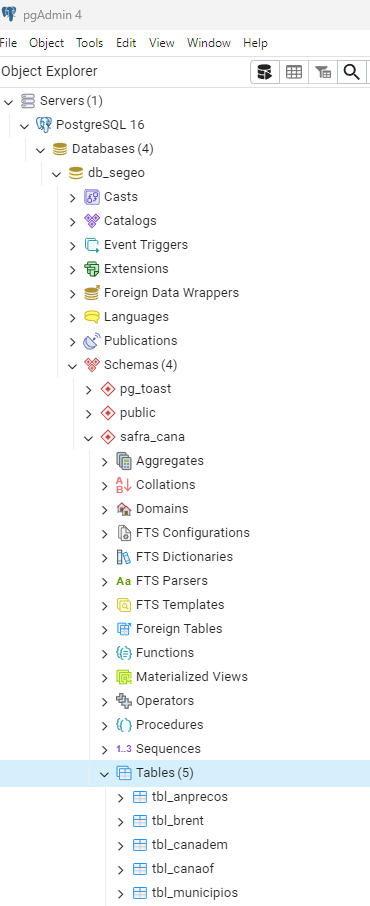
Fizemos o processo para todas as planilhas: Oferta; Demanda; Consumo e Brent

Estas bases de dados foram conectadas ao bando de dados Postgresql e posteriormente, os dados foram inseridos na base de dados.

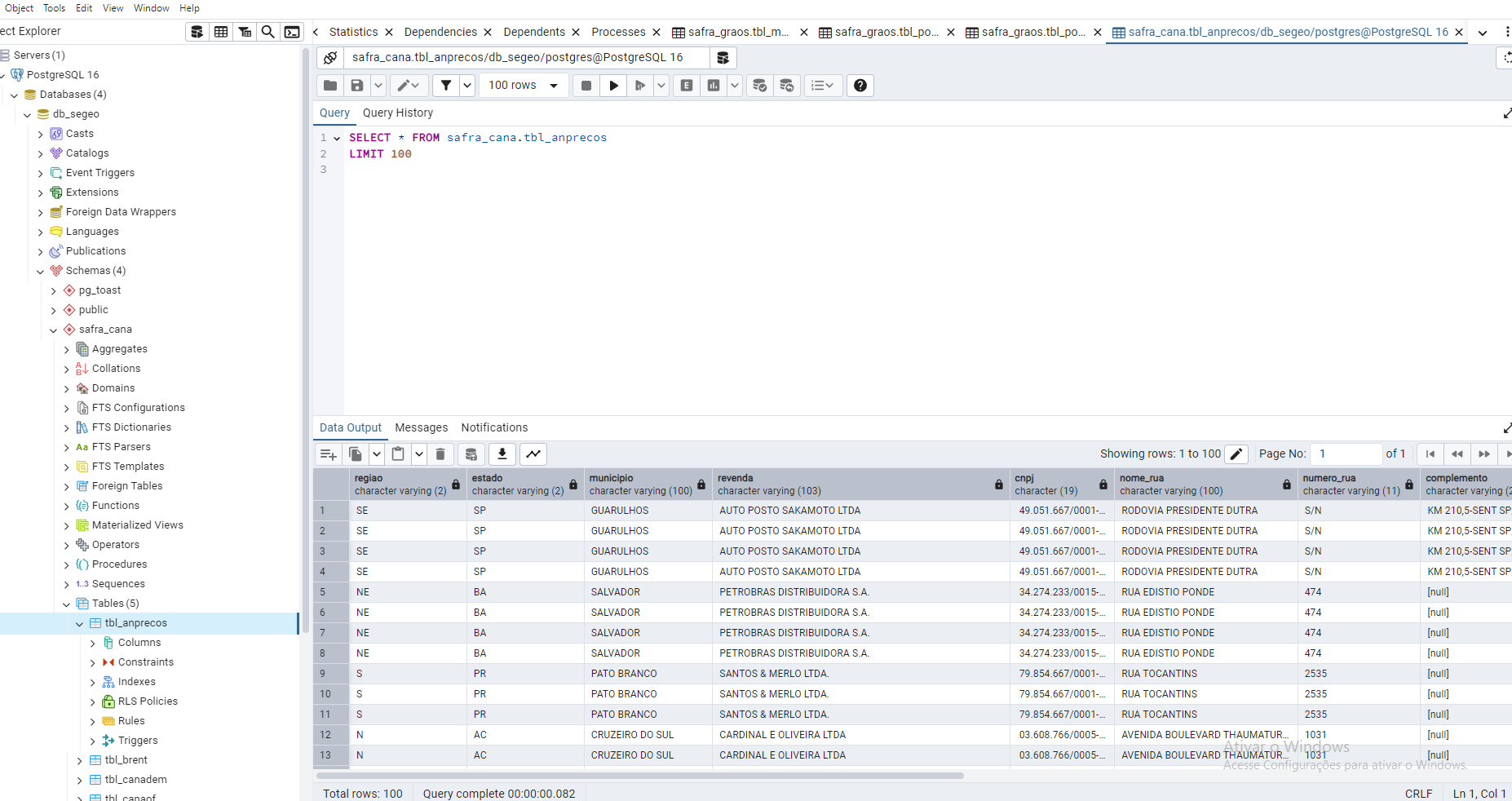
Abaixo segue um exemplo da modelagem que foi executada para todas as planilhas/tabelas.



Exemplo: dados oferta Conab

1. Criamos um banco de dados - db\_segeo, este tem um schema denominado safra\_cana, para colocar os dados de cana e preços de combustíveis.

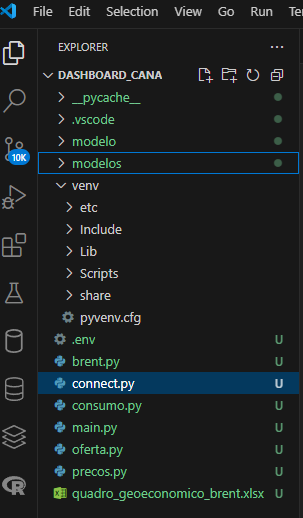
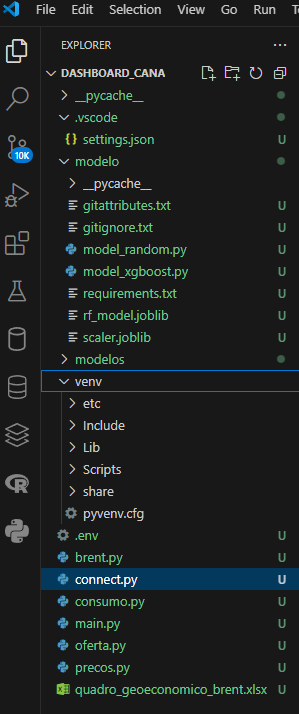
As tabelas respectivas, após a criação, foram preenchidas via ***Knime*** com os dados de cada planilha já preparada. O processo foi com via ***insert*** no db\_segeo\safra\_cana\tbl\_...



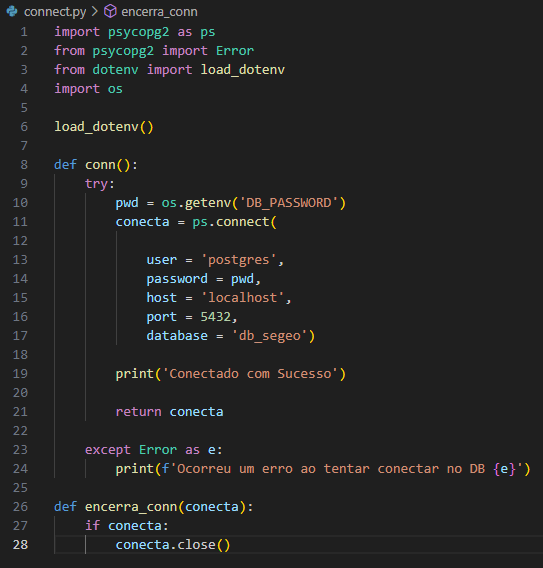
Segue o exemplo de uma tabela preenchida: tbl\_anprecos. As demais estão com a mesma estrutura, todas inseridas no db\_segeo\safra\_cana

1. Depois utilizamos o VSCODE para inserir todos os arquivos importantes para gerar os insights de preços, consumo de combustível, oferta de cana-de-açúcar e etanol e modelo de previsão de preço do petróleo (brent).

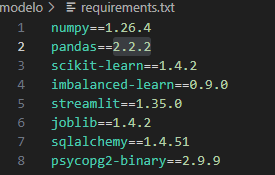
Foi criada um ambiente virtual **VENV**. A estrutura do Vscode segue abaixo:

* 1. Primeiramente, fizemos a conecção do Vscode com o banco de dados.

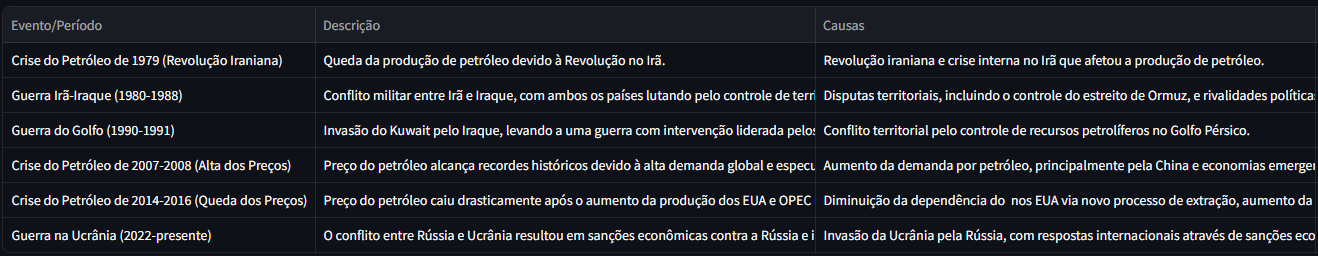


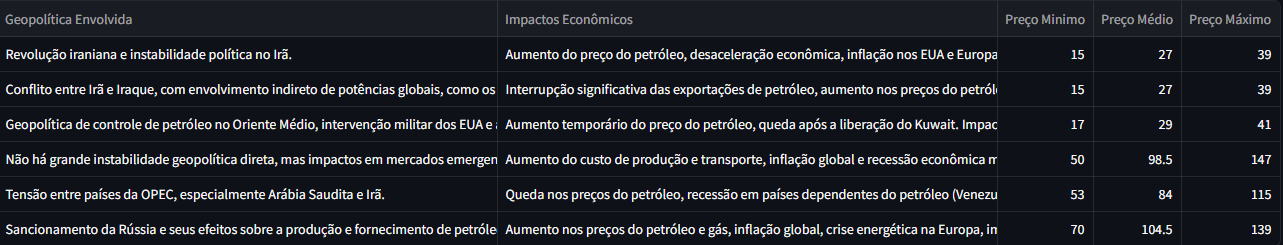
* 1. Criamos cinco arquivo oferta.py, consumo.py, precos.py, brent.py e main.py (gera a aplicação streamlit).
  2. Criamos o arquivo requirements.txt:



* 1. Em modelos, fizemos dois modelos já trabalhados no google\_colab, que são os arquivos model\_random.py (Random\_Forest) e model\_xgboost.py (Xgboost). Segue cada um abaixo:
  2. Criamos um arquivo para referenciar os acontecimentos mais importantes sobre o brent (petróleo bruto), quadro\_geoconomico\_brent.xlsx

O quadro foi inserido para o operador/analista ao visualizar no formato st.dataframe, observará os acontecimentos para correlaciona-los, as colunas do arquivo são: ***Evento/Período; Descrição; Causas; Geopolítica Envolvida; Impactos Econômicos; Preço Min, Med, Máx.***

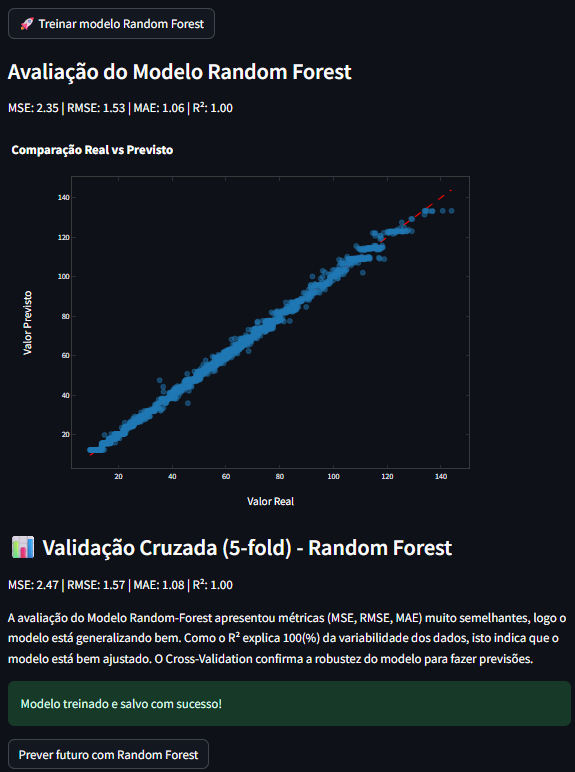




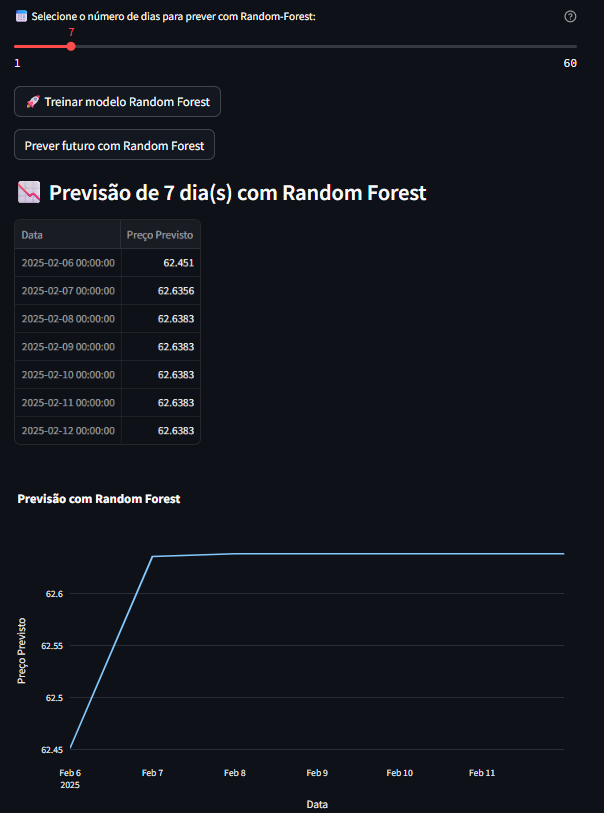
1. Modelos e Aplicação (Random\_Forest e XGBoost)
2. Validação dos modelos: Métricas escolhidas

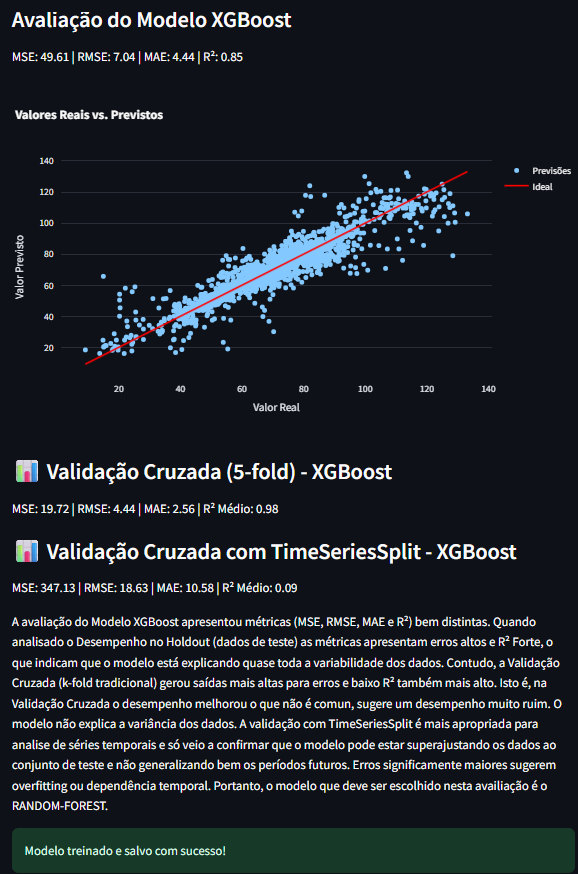


* Avaliação do Modelo Randon Forest/XGBoost Hout
* Avaliação do Modelo Randon Forest/XGBoost Cross-Validation (5 fold)
* Avaliação do Modelo Randon Forest/XGBoost Cross-Validation TimeSeriesSplit



***Insight*** – os preços do petróleo estão com comportamento similar à Crise do Petróleo de 2014-2016 (Queda dos Preços) com efeitos geopolíticos similares à uma recessão global. O barril com preços abaixo de $ 60 dólares e as tensões nos países árabes e a guerra entre Ucrânia e Rússia, intensificam a tendência de recessão com queda de preços do barril, com efeito multiplicador para outros setores. O Brasil com sua matriz energética diversificada e com a Produção de Etanol tende a amenizar o impacto desde que observado a curva descendente dos preços do Brent e com maior atuação de políticas públicas em favor da produção alcooleira. Detalhamento na oferta é que a safras estão açucareiras puxadas pelos preços da commodities açúcar no mercado internacional a mais de 5 safras.





1. Abaixo segue o arquivo com o código python dos modelos.

Random Forest:

import streamlit as st

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import plotly.express as px

from sklearn.pipeline import make\_pipeline

from sklearn.model\_selection import cross\_validate, KFold

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.preprocessing import StandardScaler

import plotly.graph\_objects as go

import joblib

from connect import conn, encerra\_conn

import warnings

warnings.filterwarnings('ignore')

# ============================

# Funções auxiliares

# ============================

def criar\_caracteristicas(brent):

    for lag in [1, 2, 3, 7, 14, 30]:

        brent[f'lag\_{lag}'] = brent['brent\_price'].shift(lag)

    for janela in [7, 14, 30]:

        brent[f'media\_movel\_{janela}'] = brent['brent\_price'].rolling(window=janela).mean()

    brent['dia\_da\_semana'] = brent['data'].dt.dayofweek

    brent['mes'] = brent['data'].dt.month

    brent['ano'] = brent['data'].dt.year

    return brent.dropna()

def treinar\_modelo\_ml(brent):

    caracteristicas = [col for col in brent.columns if col.startswith('lag\_') or col.startswith('media\_movel\_') or

                       col in ['dia\_da\_semana', 'mes', 'ano']]

    X = brent[caracteristicas]

    y = brent['brent\_price']

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

    escalador = StandardScaler()

    X\_treino\_escalado = escalador.fit\_transform(X\_treino)

    X\_teste\_escalado = escalador.transform(X\_teste)

    modelo\_rf = RandomForestRegressor(n\_estimators=100, max\_depth=5, min\_samples\_leaf=5,random\_state=42)

    modelo\_rf.fit(X\_treino\_escalado, y\_treino)

    y\_pred = modelo\_rf.predict(X\_teste\_escalado)

    #joblib.dump(modelo\_rf, 'rf\_model.joblib')

    #joblib.dump(escalador, 'scaler.joblib')

    joblib.dump(modelo\_rf, 'modelos/rf\_model.joblib')  # SALVA O MODELO TREINADO

    joblib.dump(escalador, 'modelos/scaler.joblib')

    return y\_teste, y\_pred

def avaliar\_modelo(y\_teste, y\_pred):

    mse = mean\_squared\_error(y\_teste, y\_pred)

    rmse = np.sqrt(mse)

    mae = mean\_absolute\_error(y\_teste, y\_pred)

    r2 = r2\_score(y\_teste, y\_pred)

    st.subheader('Avaliação do Modelo Random Forest')

    st.write(f'MSE: {mse:.2f} | RMSE: {rmse:.2f} | MAE: {mae:.2f} | R²: {r2:.2f}')

    fig, ax = plt.subplots()

    ax.scatter(y\_teste, y\_pred, alpha=0.5)

    ax.plot([y\_teste.min(), y\_teste.max()], [y\_teste.min(), y\_teste.max()], 'r--')

    ax.set\_xlabel('Valor Real')

    ax.set\_ylabel('Valor Previsto')

    ax.set\_title('Comparação Real vs Previsto')

    #st.pyplot(fig)

    #st.plotly\_chart(fig)

    fig, ax = plt.subplots()

    ax.scatter(y\_teste, y\_pred, alpha=0.5)

    ax.plot([y\_teste.min(), y\_teste.max()], [y\_teste.min(), y\_teste.max()], 'r--')

    # Labels e título com cor branca e tamanho maior

    ax.set\_xlabel('Valor Real', color='white', fontsize=14)

    ax.set\_ylabel('Valor Previsto', color='white', fontsize=14)

    ax.set\_title('Comparação Real vs Previsto', color='white', fontsize=16)

    # Define cor branca para os valores nos eixos

    ax.tick\_params(axis='x', colors='white')

    ax.tick\_params(axis='y', colors='white')

    # Fundo escuro opcional (fica mais elegante com texto branco)

    fig.patch.set\_facecolor('black')      # fundo fora do gráfico

    ax.set\_facecolor('#222222')           # fundo dentro do gráfico

    st.plotly\_chart(fig)

###############################################################################################################

def validar\_modelo\_com\_crossval(df):

    caracteristicas = [col for col in df.columns if col.startswith('lag\_') or col.startswith('media\_movel\_') or

                       col in ['dia\_da\_semana', 'mes', 'ano']]

    X = df[caracteristicas]

    y = df['brent\_price']

    modelo\_rf = RandomForestRegressor(

        n\_estimators=100,

        max\_depth=5,

        min\_samples\_leaf=5,

        random\_state=42

    )

    # Pipeline com escalonamento + modelo

    pipeline = make\_pipeline(StandardScaler(), modelo\_rf)

    # K-Fold com 5 divisões e shuffle para embaralhar os dados

    kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

    # Avaliação com múltiplas métricas

    resultados = cross\_validate(

        pipeline, X, y,

        cv=kf,

        scoring=['neg\_mean\_squared\_error', 'neg\_mean\_absolute\_error', 'r2'],

        return\_train\_score=False

    )

    # Resultados médios

    mse = -resultados['test\_neg\_mean\_squared\_error'].mean()

    mae = -resultados['test\_neg\_mean\_absolute\_error'].mean()

    r2 = resultados['test\_r2'].mean()

    rmse = mse \*\* 0.5

    st.subheader('📊 Validação Cruzada (5-fold) - Random Forest')

    st.write(f'MSE: {mse:.2f} | RMSE: {rmse:.2f} | MAE: {mae:.2f} | R²: {r2:.2f}')

#################################################################################################################

def prever\_futuro\_rf(df, dias\_previstos):

    modelo\_rf = joblib.load('modelo/rf\_model.joblib')

    escalador = joblib.load('modelo/scaler.joblib')

    ultimos\_dados = df.copy()

    previsoes = []

    for i in range(dias\_previstos):

        ultima\_linha = ultimos\_dados.iloc[-1].copy()

        nova\_data = ultima\_linha['data'] + pd.Timedelta(days=1)

        nova\_linha = {'data': nova\_data}

        for lag in [1, 2, 3, 7, 14, 30]:

            if len(ultimos\_dados) >= lag:

                nova\_linha[f'lag\_{lag}'] = ultimos\_dados.iloc[-lag]['brent\_price']

            else:

                nova\_linha[f'lag\_{lag}'] = np.nan

        for janela in [7, 14, 30]:

            ultimos\_dados['brent\_price'] = ultimos\_dados['brent\_price'].astype(float) # adicionada

            nova\_linha[f'media\_movel\_{janela}'] = ultimos\_dados['brent\_price'].tail(janela).mean()

        nova\_linha['dia\_da\_semana'] = nova\_data.dayofweek

        nova\_linha['mes'] = nova\_data.month

        nova\_linha['ano'] = nova\_data.year

        nova\_linha\_df = pd.DataFrame([nova\_linha]).dropna()

        if nova\_linha\_df.empty:

            break

        X\_novo = nova\_linha\_df.drop(columns=['data'])

        X\_novo\_scaled = escalador.transform(X\_novo)

        y\_pred = modelo\_rf.predict(X\_novo\_scaled)[0]

        nova\_linha\_df['brent\_price'] = y\_pred

        ultimos\_dados = pd.concat([ultimos\_dados, nova\_linha\_df[ultimos\_dados.columns]], ignore\_index=True)

        previsoes.append((nova\_data, y\_pred))

    # Mostrar resultado

    previsoes\_df = pd.DataFrame(previsoes, columns=['Data', 'Preço Previsto'])

    st.subheader(f'📉 Previsão de {dias\_previstos} dia(s) com Random Forest')

    st.dataframe(previsoes\_df.set\_index('Data'))

    fig = px.line(previsoes\_df, x='Data', y='Preço Previsto', title='Previsão com Random Forest')

    st.plotly\_chart(fig, use\_container\_width=True)

# ============================

# Aplicativo principal

# ============================

def execute\_random():

    st.title('📈 Previsão de Preço do Brent - Série Temporal')

    st.write("Este app utiliza Random Forest e XGBoost para prever preços futuros com base na base de dados do Petróleo Bruto - Brent.")

    try:

        # Conectando ao banco de dados

        connection = conn()

        cursor = connection.cursor()

        cursor.execute('SELECT \* FROM safra\_cana.tbl\_brent')

        rows = cursor.fetchall()

        col\_names = [desc[0] for desc in cursor.description]

        df = pd.DataFrame(rows, columns=col\_names)

        # Fechando conexão com o banco

        cursor.close()

        encerra\_conn(connection)

    except Exception as e:

        st.error(f"Erro ao conectar ou consultar o banco de dados: {e}")

        return

    if 'data' not in df.columns or 'brent\_price' not in df.columns:

        st.error('❌ A tabela deve conter as colunas "data" e "brent\_price"')

        return

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

    st.subheader('🔎 Visualização dos dados')

    col1, col2 = st.columns(2)

    with col1:

        st.write("\*\*Início do DataFrame\*\*")

        st.write(df.head())

    with col2:

        st.write("\*\*Fim do DataFrame\*\*")

        st.write(df.tail())

    df = criar\_caracteristicas(df)

    st.markdown("---")

    dias\_previstos = st.slider(

        '📅 Selecione o número de dias para prever com Random-Forest:',

        min\_value=1,

        max\_value=60,

        value=7,

        help='Arraste para ver a previsão')

    if st.button('🚀 Treinar modelo Random Forest'):

        y\_teste, y\_pred = treinar\_modelo\_ml(df)

        avaliar\_modelo(y\_teste, y\_pred)

        validar\_modelo\_com\_crossval(df)  # nova funcionalidade

        st.write('A avaliação do Modelo Random-Forest apresentou métricas (MSE, RMSE, MAE) muito semelhantes,\

                 logo o modelo está generalizando bem. Como o R² explica 100(%) da variabilidade dos dados, isto indica\

                     que o modelo está bem ajustado. O Cross-Validation confirma a robustez do modelo para fazer previsões. ')

        st.success("Modelo treinado e salvo com sucesso!")

    if st.button('Prever futuro com Random Forest'):

        prever\_futuro\_rf(df, dias\_previstos) # previsão dias previstos

XGBoost:

import streamlit as st

import pandas as pd

import numpy as np

import plotly.graph\_objects as go

import plotly.express as px

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from xgboost import XGBRegressor

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

import os

import joblib

from connect import conn, encerra\_conn

import warnings

warnings.filterwarnings('ignore')

from sklearn.model\_selection import KFold, cross\_validate, TimeSeriesSplit

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

# ============================

# Funções auxiliares

# ============================

def criar\_caracteristicas(brent):

    brent = brent.sort\_values('data').reset\_index(drop=True)

    # Criar colunas lag usando shift, para usar valores anteriores

    for lag in [1, 2, 3, 7, 14, 30]:

        brent[f'lag\_{lag}'] = brent['brent\_price'].shift(lag)

    # Converter colunas lag para numérico (precaução)

    lags = [f'lag\_{lag}' for lag in [1, 2, 3, 7, 14, 30]]

    for lag in lags:

        brent[lag] = pd.to\_numeric(brent[lag], errors='coerce')

    # Preencher valores ausentes nos lags (NaNs das primeiras linhas)

    brent[lags] = brent[lags].fillna(method='bfill')  # ou fillna(0) se preferir

    # Remover linhas com NaN em brent\_price ou lags (se houver)

    brent = brent.dropna(subset=lags + ['brent\_price']).reset\_index(drop=True)

    return brent

def treinar\_modelo\_xgboost(brent):

    lags = ['lag\_1', 'lag\_2', 'lag\_3', 'lag\_7', 'lag\_14', 'lag\_30']

    for lag in lags:

        brent[lag] = pd.to\_numeric(brent[lag], errors='coerce')

    brent[lags] = brent[lags].fillna(method='bfill')

    features = lags

    target = 'brent\_price'

    brent = brent.dropna(subset=features + [target])

    X = brent[features]

    y = brent[target]

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

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

    import xgboost as xgb

    modelo = xgb.XGBRegressor()

    modelo.fit(X\_train, y\_train)

    y\_pred = modelo.predict(X\_test)

    return y\_test, y\_pred, modelo, X\_test, y\_test

def avaliar\_modelo(y\_teste, y\_pred):

    mse = mean\_squared\_error(y\_teste, y\_pred)

    rmse = np.sqrt(mse)

    mae = mean\_absolute\_error(y\_teste, y\_pred)

    r2 = r2\_score(y\_teste, y\_pred)

    st.subheader('Avaliação do Modelo XGBoost')

    st.write(f'MSE: {mse:.2f} | RMSE: {rmse:.2f} | MAE: {mae:.2f} | R²: {r2:.2f}')

    fig = go.Figure()

    fig.add\_trace(go.Scatter(x=y\_teste, y=y\_pred, mode='markers', name='Previsões'))

    fig.add\_trace(go.Scatter(x=[min(y\_teste), max(y\_teste)], y=[min(y\_teste), max(y\_teste)], mode='lines', name='Ideal', line=dict(color='red')))

    fig.update\_layout(title='Valores Reais vs. Previstos', xaxis\_title='Valor Real', yaxis\_title='Valor Previsto')

    st.plotly\_chart(fig, use\_container\_width=True)

#########################################

# codigo novo previsão

def prever\_futuro\_xgboost1(brent, dias\_previstos):

    try:

        modelo = joblib.load('modelos/xgboost\_model.joblib')

    except FileNotFoundError:

        st.error("Modelo XGBoost não encontrado. Por favor, treine o modelo primeiro.")

        return

    # ✅ Conversão da coluna 'data' para datetime

    brent['data'] = pd.to\_datetime(brent['data'])

    # ✅ Ordenar e resetar índice

    brent = brent.sort\_values('data').reset\_index(drop=True)

    ult\_data = brent['data'].iloc[-1]

    ult\_precos = brent['brent\_price'].tolist()

    previsoes = []

    for i in range(dias\_previstos):

        lags = []

        for lag in [1, 2, 3, 7, 14, 30]:

            idx = len(ult\_precos) - lag

            if idx >= 0:

                lags.append(ult\_precos[idx])

            else:

                lags.append(ult\_precos[0])  # fallback

        X\_pred = pd.DataFrame([lags], columns=[f'lag\_{l}' for l in [1, 2, 3, 7, 14, 30]])

        X\_pred = X\_pred.astype(float)

        pred = modelo.predict(X\_pred)[0]

        ult\_precos.append(pred)

        previsoes.append(pred)

    # ✅ Gerar datas futuras corretamente a partir da última data do dataset

    datas\_futuras = [ult\_data + pd.Timedelta(days=i + 1) for i in range(dias\_previstos)]

    if len(datas\_futuras) != len(previsoes):

        st.error(f"Tamanho incompatível: datas\_futuras={len(datas\_futuras)}, previsoes={len(previsoes)}")

        return

    df\_previsao = pd.DataFrame({'data': datas\_futuras, 'previsao': previsoes})

    st.subheader(f'Previsão para os próximos {dias\_previstos} dias')

    st.dataframe(df\_previsao.set\_index('data'))

    fig = px.line(df\_previsao, x='data', y='previsao', title='Previsão XGBoost')

    st.plotly\_chart(fig, use\_container\_width=True)

##############################################

def prever\_futuro\_xgboost(brent, dias\_previstos):

    try:

        modelo = joblib.load('modelos/xgboost\_model.joblib')

    except FileNotFoundError:

        st.error("Modelo XGBoost não encontrado. Por favor, treine o modelo primeiro.")

        return

    ##########

    brent['data'] = pd.to\_datetime(brent['data'], errors='coerce')

    ##########

    #df = df.sort\_values('data').reset\_index(drop=True)

    #####################

    # ✅ Remover possíveis linhas com data inválida

    brent = brent.dropna(subset=['data'])

    brent = brent.sort\_values('data').reset\_index(drop=True)

    ########################################

    ult\_data = brent['data'].iloc[-1]

    st.write(f"Última data no dataset: {ult\_data}")

    ult\_precos = brent['brent\_price'].tolist()

    previsoes = []

    for i in range(dias\_previstos):

        lags = []

        for lag in [1, 2, 3, 7, 14, 30]:

            idx = len(ult\_precos) - lag

            if idx >= 0:

                lags.append(ult\_precos[idx])

            else:

                lags.append(ult\_precos[0])  # fallback

        X\_pred = pd.DataFrame([lags], columns=[f'lag\_{l}' for l in [1, 2, 3, 7, 14, 30]])

        X\_pred = X\_pred.astype(float)

        pred = modelo.predict(X\_pred)[0]

        ult\_precos.append(pred)

        previsoes.append(pred)

    datas\_futuras = [ult\_data + pd.Timedelta(days=i+1) for i in range(dias\_previstos)]

    # Verificação de segurança

    if len(datas\_futuras) != len(previsoes):

        st.error(f"Tamanho incompatível: datas\_futuras={len(datas\_futuras)}, previsoes={len(previsoes)}")

        return

    df\_previsao = pd.DataFrame({'data': datas\_futuras, 'previsao': previsoes})

    st.subheader(f'Previsão para os próximos {dias\_previstos} dias')

    st.dataframe(df\_previsao.set\_index('data'))

    fig = px.line(df\_previsao, x='data', y='previsao', title='Previsão XGBoost')

    st.plotly\_chart(fig, use\_container\_width=True)

###################

def validar\_xgboost\_timeseries(brent):

    st.subheader('📊 Validação Cruzada com TimeSeriesSplit - XGBoost')

    lags = ['lag\_1', 'lag\_2', 'lag\_3', 'lag\_7', 'lag\_14', 'lag\_30']

    brent = brent.dropna(subset=lags + ['brent\_price']).copy()

    X = brent[lags]

    y = brent['brent\_price']

    modelo = XGBRegressor(

        n\_estimators=100,

        max\_depth=5,

        learning\_rate=0.1,

        random\_state=42

    )

    pipeline = make\_pipeline(StandardScaler(), modelo)

    tscv = TimeSeriesSplit(n\_splits=5)

    resultados = cross\_validate(

        pipeline, X, y,

        cv=tscv,

        scoring=['neg\_mean\_squared\_error', 'neg\_mean\_absolute\_error', 'r2'],

        return\_train\_score=False

    )

    mse = -resultados['test\_neg\_mean\_squared\_error'].mean()

    mae = -resultados['test\_neg\_mean\_absolute\_error'].mean()

    r2 = resultados['test\_r2'].mean()

    rmse = mse \*\* 0.5

    st.write(f'MSE: {mse:.2f} | RMSE: {rmse:.2f} | MAE: {mae:.2f} | R² Médio: {r2:.2f}')

#####################

def validar\_modelo\_xgboost\_com\_crossval(brent):

    lags = [f'lag\_{lag}' for lag in [1, 2, 3, 7, 14, 30]]

    # Remove valores ausentes

    brent = brent.dropna(subset=lags + ['brent\_price'])

    X = brent[lags]

    y = brent['brent\_price']

    modelo\_xgb = XGBRegressor(

        n\_estimators=100,

        max\_depth=5,

        learning\_rate=0.1,

        random\_state=42

    )

    pipeline = make\_pipeline(StandardScaler(), modelo\_xgb)

    kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

    resultados = cross\_validate(

        pipeline, X, y,

        cv=kf,

        scoring=['neg\_mean\_squared\_error', 'neg\_mean\_absolute\_error', 'r2'],

        return\_train\_score=False

    )

    mse = -resultados['test\_neg\_mean\_squared\_error'].mean()

    mae = -resultados['test\_neg\_mean\_absolute\_error'].mean()

    r2 = resultados['test\_r2'].mean()

    rmse = mse \*\* 0.5

    st.subheader('📊 Validação Cruzada (5-fold) - XGBoost')

    st.write(f'MSE: {mse:.2f} | RMSE: {rmse:.2f} | MAE: {mae:.2f} | R² Médio: {r2:.2f}')

# ============================

# Aplicativo principal

# ============================

def execute\_xgboost():

    st.title('📈 Previsão de Preço do Brent')

    st.write("Este app utiliza Random Forest e XGBoost para prever preços futuros com base na base de dados do Petróleo Bruto - Brent.")

    try:

        connection = conn()

        cursor = connection.cursor()

        cursor.execute('SELECT \* FROM safra\_cana.tbl\_brent')

        rows = cursor.fetchall()

        col\_names = [desc[0] for desc in cursor.description]

        df = pd.DataFrame(rows, columns=col\_names)

        cursor.close()

        encerra\_conn(connection)

    except Exception as e:

        st.error(f"Erro ao acessar o banco de dados: {e}")

        return

    df['data'] = pd.to\_datetime(df['data'], errors='coerce')

    st.subheader('🔎 Visualização dos dados')

    col1, col2 = st.columns(2)

    with col1:

        st.write("\*\*Início do DataFrame\*\*")

        st.write(df.head())

    with col2:

        st.write("\*\*Fim do DataFrame\*\*")

        st.write(df.tail())

    df = criar\_caracteristicas(df)

    dias\_previstos = st.slider('Selecione o número de dias para prever:', 1, 60, 7, help='Arraste para ver a previsão')

    # Só roda XGBoost agora, sem seleção de modelo

    if st.button('Treinar modelo XGBoost'):

        y\_teste, y\_pred, modelo, X\_test, y\_test = treinar\_modelo\_xgboost(df)

        avaliar\_modelo(y\_teste, y\_pred)

        validar\_modelo\_xgboost\_com\_crossval(df) # nova função

        validar\_xgboost\_timeseries(df)

        st.write('A avaliação do Modelo XGBoost apresentou métricas (MSE, RMSE, MAE e R²) bem distintas.\

                  Quando analisado o Desempenho no Holdout (dados de teste) as métricas apresentam erros altos e\

                      R² Forte, o que indicam que o modelo está explicando quase toda a variabilidade dos dados.\

                          Contudo, a Validação Cruzada (k-fold tradicional) gerou saídas mais altas para erros e baixo R² também mais alto.\

                              Isto é, na Validação Cruzada o desempenho melhorou o que não é comun, sugere um desempenho muito ruim. O modelo não explica a variância dos dados. A validação com TimeSeriesSplit\

                                      é mais apropriada para analise de séries temporais e só veio a confirmar que o modelo pode estar \

                                          superajustando os dados ao conjunto de teste e não generalizando bem os períodos futuros.  Erros\

                                  significamente maiores sugerem overfitting ou dependência temporal. Portanto,\

                                              o modelo que deve ser escolhido nesta availiação é o RANDOM-FOREST.')

        if not os.path.exists('modelos'):

            os.makedirs('modelos')

        joblib.dump(modelo, 'modelos/xgboost\_model.joblib')  # SALVA O MODELO TREINADO

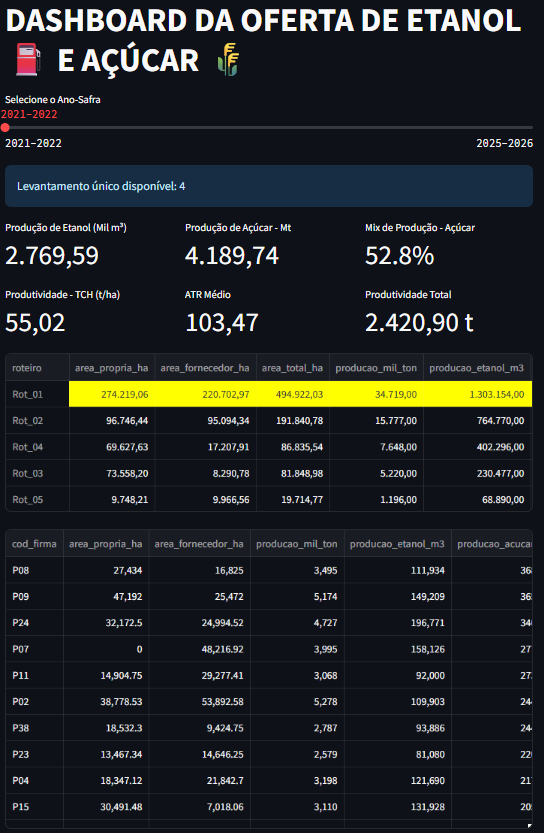
        st.success("Modelo treinado e salvo com sucesso!")

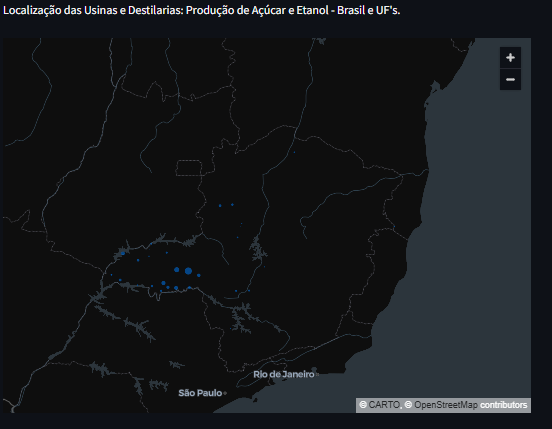
    #if st.button('Prever futuro com XGBoost'):

    #       prever\_futuro\_xgboost(df, dias\_previstos)

**Telas:** Todas as aplicações de cada tela estão interativas, ao clicar em um ano respectivo ou ano-safra, os valores mudam e o cenário dos gráficos e mapa também mudam.

***OFERTA***

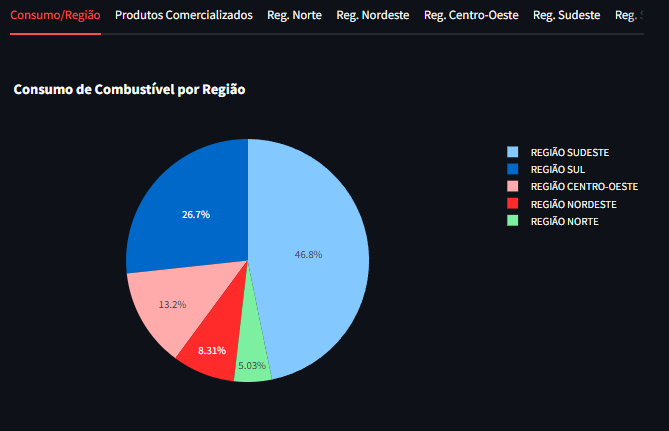


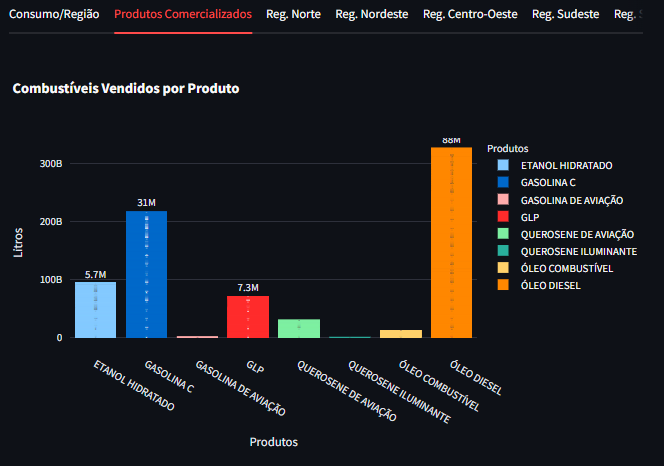


Obs.: dados utilizados somente do estado de MG (dados confidenciais)

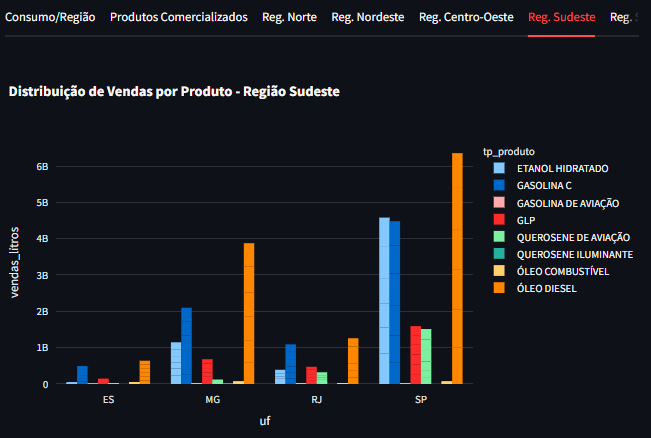
***CONSUMO***



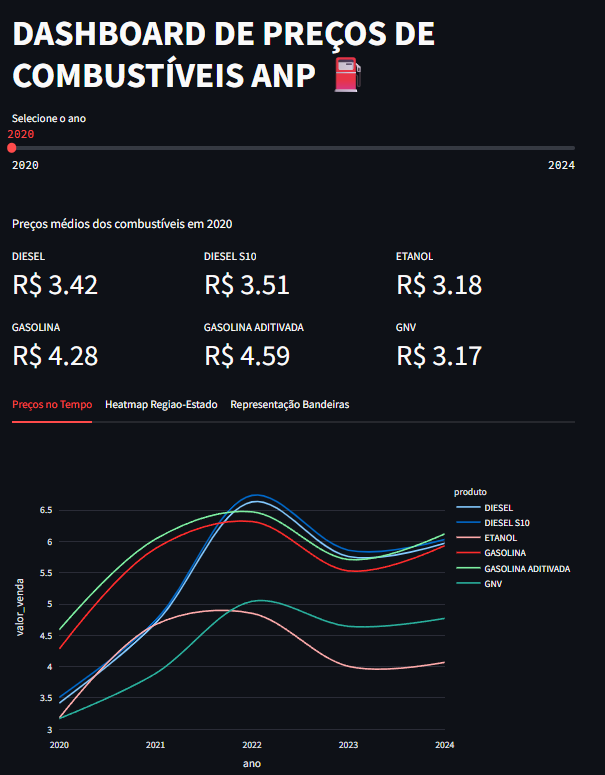


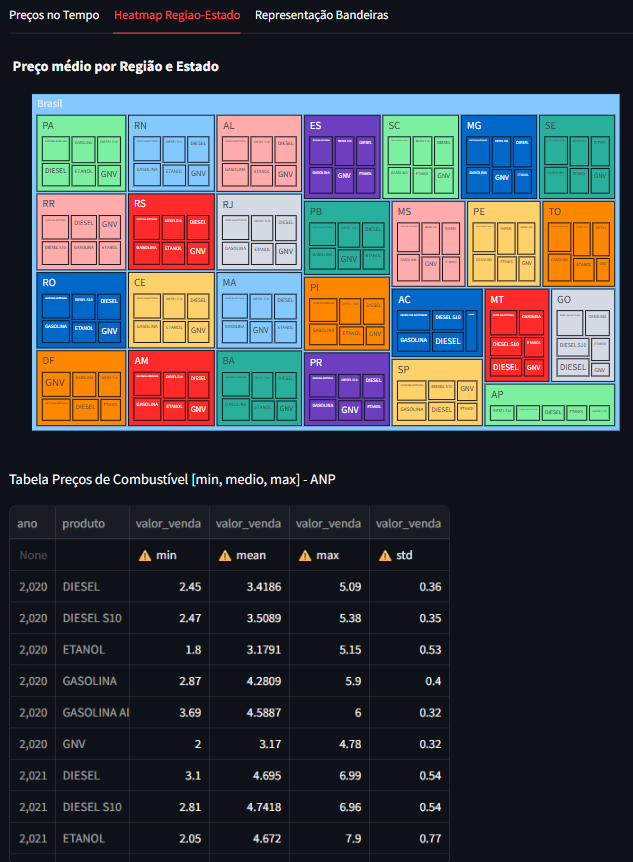


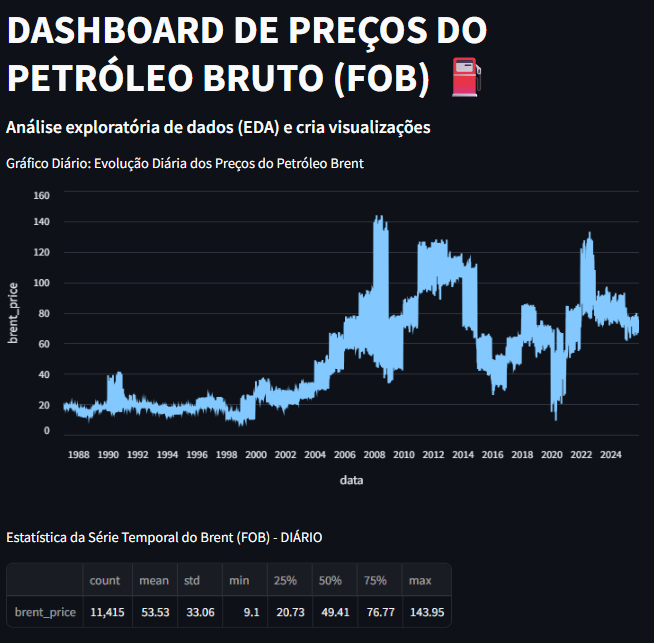
Os gráficos da distribuição por produto estão por região geográfica: Norte, Nordeste, Centro-Oeste, Sudeste e Sul. (padrão de gráfico para todas as regiões)

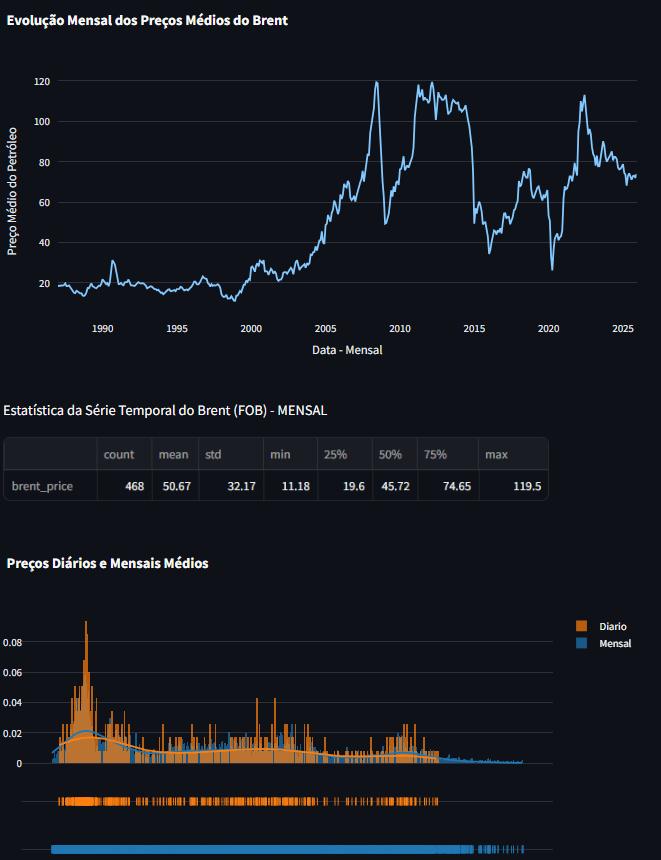


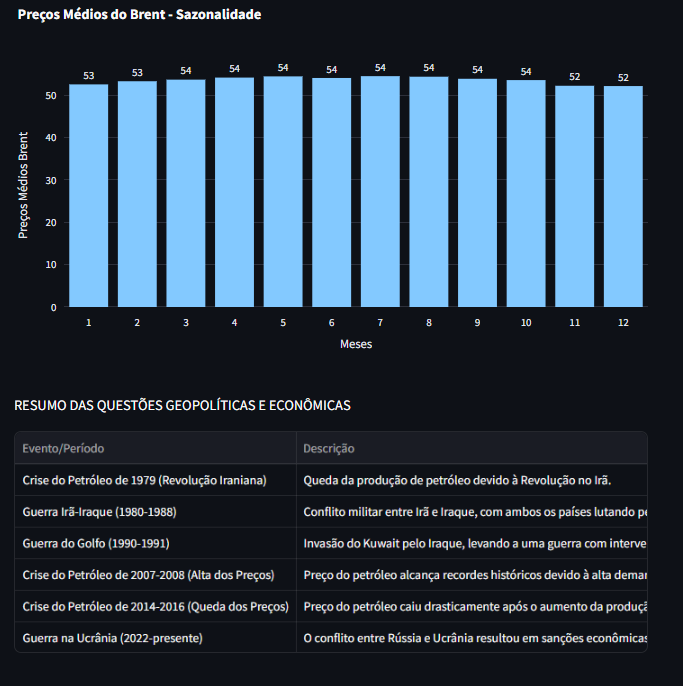
***PREÇOS ANP (VAREJO)***











***MODELOS***



