**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**BRUNO LOVATTI**

**PREVISÃO DE VENDAS COMO FERRAMENTA PARA REDUÇÃO DE RUPTURA E EXCESSO DE PRODUTOS**

Vitória - ES

2022

**Bruno Lovatti**

**PREVISÃO DE VENDAS COMO FERRAMENTA PARA REDUÇÃO DE RUPTURA E EXCESSO DE PRODUTOS**

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Vitória - ES

2022**SUMÁRIO**

[1. Introdução 5](#_Toc74563991)

[1.1. Contextualização 5](#_Toc74563992)

[1.2. Problema Proposto – Compra Otimizada 5](#_Toc74563993)

[1.3. **Objetivo – Criação de Programa para Previsão de Vendas com Boa Qualidade** 6](#_Toc74563994)

[2. **Coleta de Dados** 7](#_Toc74563994)

[2.1. **Vendas** 7](#_Toc74563994)

[2.2. **Dados Meteorológicos** 9](#_Toc74563994)

[3. Processamento/Tratamento de Dados 11](#_Toc74563995)

[3.1. Vendas 11](#_Toc74563995)

[3.2. Dados Meteorológicos 11](#_Toc74563995)

[4. Análise e Exploração dos Dados 13](#_Toc74563996)

[5. Criação de Modelos de Machine Learning 16](#_Toc74563997)

[5.1. sktime 16](#_Toc74563997)

[5.1.1. Modelo Naive 17](#_Toc74563997)

[5.1.2. Modelo AUTOARIMA 17](#_Toc74563997)

[5.1.3. Modelo AUTOETS 17](#_Toc74563997)

[5.1.4. Modelo BATS 18](#_Toc74563997)

[5.1.5. Modelo Exponential Smoothing 18](#_Toc74563997)

[5.1.6. Modelo Prophet com sktime 19](#_Toc74563997)

[5.2. Facebook Prophet 19](#_Toc74563997)

[6. Interpretação dos Resultados 22](#_Toc74563998)

[6.1.1. Modelo Naive 22](#_Toc74563997)

[6.1.2. Modelo AUTOARIMA 22](#_Toc74563997)

[6.1.3. Modelo AUTOESTS 22](#_Toc74563997)

[6.1.4. Modelo BATS 22](#_Toc74563997)

[6.1.5. Modelo Exponential Smoothing 22](#_Toc74563997)

[6.1.6. Modelo Prophet com sktime 22](#_Toc74563997)

[6.2. Facebook Prophet 22](#_Toc74563997)

[7. Apresentação dos Resultados 24](#_Toc74563999)

[7.1. Gráfico com a Previsão de Vendas 24](#_Toc74563999)

[7.2. Gráfico com a Tendência e Sazonalidade 24](#_Toc74563999)

[7.3. Gráfico com a Métrica MAPE 26](#_Toc74563999)

[7.4. Gráfico com Valores Previstos e Realizados 26](#_Toc74563999)

[7.5. Gráfico com Valores Previstos e Realizados 27](#_Toc74563999)

[8. Links 28](#_Toc74564000)

[REFERÊNCIAS 29](#_Toc74564001)

[APÊNDICE 30](#_Toc74564002)

# 1. Introdução

## 1.1. Contextualização

Dentre os vários problemas operacionais que afetam os comércios varejistas no Brasil e no mundo, falta de estoque (ruptura) e excesso de estoque (cobertura excedente) são especialmente relevantes e impactam negativamente o negócio.

A ruptura de estoque é a indisponibilidade de um produto na prateleira, no momento da compra pelo cliente, e as consequências negativas vão desde a perda de vendas até a insatisfação de consumidores.

Um estudo pioneiro foi realizado por Peckham (1963) e identificou taxas de ruptura de estoque de 8,5%, praticamente ratificados por outros estudos (CORSTEN; GRUEN, 2003) que apontaram para uma taxa mundial de 8,3%.

A cobertura excedente, por outro lado, é a sobra de estoque de um produto e também provoca sérios problemas às empresas, impactam o capital de giro e o CMV (custo da mercadoria vendida), por exemplo.

## 1.2. Problema Proposto – Compra Otimizada

Otimizar o estoque, ou seja, promover um equilíbrio que possa manter o estoque baixo o suficiente para atender à demanda é um grande desafio e comprar a quantidade correta é muito importante. Sendo assim, a utilização de um algoritmo de previsão de demanda (“Demand Forecasting”) ou vendas, no caso, pode fornecer o apoio necessário.

Logicamente, a ruptura de produto possui outras causas, mas neste trabalho será abordado apenas o problema da compra inadequada, que pode ser atenuado com base boas previsões e compras.

Foram analisadas as vendas de sorvetes de uma rede de supermercados do Espírito Santo, assim como os dados meteorológicos obtidos do INMET (Instituto Nacional de Meteorologia).

# **1.3. Objetivo – Criação de Programa para Previsão de Vendas com Boa Qualidade**

O objetivo deste trabalho é a criação de um programa para previsão de vendas que tenha boa qualidade para o grupo de produtos analisado.

Dentre as opções de algoritmos disponíveis na comunidade, optou-se por utilizar o Facebook Prophet com regressores de dados meteorológicos. Foi também demonstrada a utilização do framework sktime, que encapsula e simplifica a utilização de alguns algoritmos.

A qualidade da previsão do programa foi avaliada com a utilização de algumas métricas conhecidas como MAE – Erro Absoluto Médio e MAPE – Erro Percentual Médio.

O MAE mede a média da diferença entre o valor real com o predito. Mas por haver valores positivos e negativos, é utilizado o módulo da diferença dos valores. Além disso, esta métrica não é afetada por valores discrepantes ou *outliers*.

O MAPE é uma métrica que mostra a porcentagem de erro em relação aos valores reais. Se o resultado de MAPE for igual a 20% significa que o modelo faz previsões que em média a diferença entre o valor previsto e o real equivale a 20% do valor real tanto para mais quanto para menos.

2. Coleta de Dados

O programa de previsão de vendas deste trabalho de conclusão de curso utiliza dois *datasets*, vendas de sorvetes da rede de supermercados analisada e dados meteorológicos do INMET (Instituto Nacional de Meteorologia), relacionados por data.

## 2.1. Vendas

Os dados das vendas foram obtidos diretamente do SGBD (Sistema Gerenciador de Banco de Dados), Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit, sendo executado em um servidor ODA X8-2-HA (*Oracle Database Appliance*).

Um procedimento armazenado SP\_EXPT\_VENDA foi criado para a exportação dos dados da tabela TAB\_VENDA\_GERAL para o formato colunado CSV (*Comma Separated File*) e enviados para o Google Drive, para serem utilizados, posteriormente, pelo Google Colab.

Além disso, as vendas são apenas dos produtos que compõem a seção Sorvetes de um grupo de lojas da rede, no período de 01/01/2018 a 31/03/2022, agrupadas por data, por loja fictícia 99 e por produto fictício 97.

|  |  |  |
| --- | --- | --- |
| Nome da coluna/campo | Descrição | Tipo |
| VDA\_DATA | Data da venda | Data |
| VDA\_LOJA | Código da loja da venda | Número Inteiro |
| VDA\_PROD | Código do produto da venda | Número Inteiro |
| VDA\_QTD | Quantidade da venda | Número Decimal |

Para o grupo de produtos sorvetes e período de 01/01/2018 a 31/03/2022, são 471.781 registros, distribuídos em 97 produtos e 15 lojas. O código A abaixo retorna todos os registros de vendas e o código B faz o agrupamento por data e por loja e produto fictícios a fim de generalizar a análise das vendas de sorvetes no programa Python, somente para fins deste trabalho, e retorna 1.483 registros, que é a quantidade de dias que possuem vendas, no período.

Código A.

Todos os registros de vendas.

SELECT

TRUNC(VDA\_DATAHORA) AS VDA\_DATA,

VDA\_LOJA || '\_' || VDA\_PROD AS VDA\_LOJA\_PROD,

VDA\_PROD,

VDA\_QTD

FROM

PROJETOS.V\_TAB\_VENDA\_GERAL VD

WHERE

VD.VDA\_DATAHORA BETWEEN to\_date('20180101','yyyymmdd')

AND to\_date('20220331','yyyymmdd') + 86399 / 86400

AND VD.VDA\_KIT IN (0,2)

AND VD.VDA\_PROD IN

(

SELECT PROD\_CODIGO

FROM PROJETOS.TAB\_PRODUTO T

WHERE T.PROD\_SECAO = 14

AND T.PROD\_GRUPO = 2

AND T.PROD\_SUBGRUPO = 2)

Código B.

Vendas agrupadas por data do ano, assim como loja e produto fictícios.

SELECT

TRUNC(VDA\_DATAHORA) AS VDA\_DATA,

99 AS VDA\_LOJA,

97 AS VDA\_PROD,

SUM(VDA\_QTD) AS VDA\_QTD

FROM

PROJETOS.V\_TAB\_VENDA\_GERAL VD

WHERE

VD.VDA\_DATAHORA BETWEEN to\_date('20180101','yyyymmdd')

AND to\_date('20181231','yyyymmdd') + 86399 / 86400

AND VD.VDA\_KIT IN ( 0,2 )

AND VD.VDA\_PROD IN

(

SELECT PROD\_CODIGO

FROM PROJETOS.TAB\_PRODUTO T

WHERE T.PROD\_SECAO = 14

AND T.PROD\_GRUPO = 2

AND T.PROD\_SUBGRUPO = 2

)

GROUP BY

TRUNC(VDA\_DATAHORA),

99,

97

## 2.2. Dados Meteorológicos

Os dados meteorológicos foram obtidos do site do INMET, na opção Histórico de Dados Meteorológicos, diretamente pelo programa principal em Python.

Os arquivos disponibilizados pelo INMET estão em formato compactado ZIP (Ex.: 2022.zip), contendo arquivos CSV de dados por estação meteorológica (Ex.: INMET\_SE\_ES\_A612\_VITORIA\_01-01-2022\_A\_31-01-2022.CSV).

Importante ressaltar que os arquivos CSV, dentro dos arquivos ZIP, possuem colunas com nomes diferentes entre os anos utilizados e foram aproveitadas apenas três colunas para o trabalho.

O código 1.4.1 abaixo faz o download do arquivo ZIP de cada ano de 2018 a 2022 para o Google Drive, a extração dos arquivos CSV da região de Vitória, no Espírito Santo, a leitura de apenas três colunas e a concatenação em um Dataframe.

|  |  |  |
| --- | --- | --- |
| Nome da coluna/campo | Descrição | Tipo |
| DATA (YYYY-MM-DD) ou Data | Data da venda | Data |
| PRECIPITAÇÃO TOTAL, HORÁRIO (mm) | Código da loja da venda | Número Decimal |
| TEMPERATURA DO AR - BULBO SECO, HORARIA (°C) | Código do produto da venda | Número Decimal |

Código 1.4.1.

Download, extração e leitura de dados do INMET.

from datetime import datetime

custom\_date\_parser\_1 = lambda x: datetime.strptime(x, '%Y-%m-%d')

custom\_date\_parser\_2 = lambda x: datetime.strptime(x, '%Y/%m/%d')

import requests

import zipfile

for ano in range(2018, 2023):

  file\_url = f"https://portal.inmet.gov.br/uploads/dadoshistoricos/{ano}.zip"

  file\_gdrive = f"INMET-{ano}.zip"

  if not os.path.exists(file\_gdrive):

    r = requests.get(file\_url, stream = True)

    with open(file\_gdrive, "wb") as file:

      for block in r.iter\_content(chunk\_size = 1024):

        if block:

          file.write(block)

  with zipfile.ZipFile(file\_gdrive) as myzip:

    # print(myzip.namelist())

    if ano in [2018, 2019]:

      file\_csv = f"{ano}/INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_31-12-{ano}.CSV"

    elif ano in [2022]:

      file\_csv = f"INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_28-02-{ano}.CSV"

    else:

      file\_csv = f"INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_31-12-{ano}.CSV"

    print(file\_csv)

    with myzip.open(file\_csv) as myfile:

        var\_name = 'df\_INMET\_{}'.format(str(ano))

        if ano in [2018]:

          col = 'DATA (YYYY-MM-DD)'

          parser = custom\_date\_parser\_1

          globals()[var\_name] = pd.read\_csv(myfile, sep=";", encoding = "ISO-8859-1", header=8, parse\_dates=[col], date\_parser=parser, decimal=",")

          globals()[var\_name] = globals()[var\_name][['DATA (YYYY-MM-DD)', 'PRECIPITAÇÃO TOTAL, HORÁRIO (mm)', 'TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)']].rename(columns={"DATA (YYYY-MM-DD)": "ds", "PRECIPITAÇÃO TOTAL, HORÁRIO (mm)": "precipitacao", "TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)": "temperatura"})

        else:

          col = 'Data'

          parser = custom\_date\_parser\_2

          globals()[var\_name] = pd.read\_csv(myfile, sep=";", encoding = "ISO-8859-1", header=8, parse\_dates=[col], date\_parser=parser, decimal=",")

          globals()[var\_name] = globals()[var\_name][['Data', 'PRECIPITAÇÃO TOTAL, HORÁRIO (mm)', 'TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)']].rename(columns={"Data": "ds", "PRECIPITAÇÃO TOTAL, HORÁRIO (mm)": "precipitacao", "TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)": "temperatura"})

dados\_meteor\_all = pd.concat([globals()["df\_INMET\_2018"],globals()["df\_INMET\_2019"],globals()["df\_INMET\_2020"],globals()["df\_INMET\_2021"],globals()["df\_INMET\_2022"]])

dados\_meteor\_all

# 

# 3. Processamento/Tratamento de Dados

## 3.1. Vendas

As vendas foram agrupadas desde a coleta no banco de dados, exportadas para CSV e enviadas para o Google Drive. Em seguida, em Python, foram lidas e agrupadas no código 1.3.3. Uma função assegura a leitura do campo data seguindo o padrão ISO existente no arquivo. Não foram encontrados registros duplicados pois a consulta ao banco de dados elimina esta possibilidade.

custom\_date\_parser = lambda x: datetime.strptime(x, '%Y-%m-%d')

def r(f):

  return pd.read\_csv(f, sep='|', parse\_dates=['VDA\_DATA'], date\_parser=custom\_date\_parser)

vendas = pd.concat(map(r, all\_files))

## 3.2. Dados Meteorológicos

No código 1.4.1, os dados meteorológicos foram lidos diretamente do portal INMET em formato ZIP e salvos no Google Drive.

file\_url = f"https://portal.inmet.gov.br/uploads/dadoshistoricos/{ano}.zip"

  file\_gdrive = f"INMET-{ano}.zip"

  if not os.path.exists(file\_gdrive):

    r = requests.get(file\_url, stream = True)

    with open(file\_gdrive, "wb") as file:

      for block in r.iter\_content(chunk\_size = 1024):

        if block:

          file.write(block)

Cada arquivo ZIP foi lido em memória.

with zipfile.ZipFile(file\_gdrive) as myzip:

Em seguida, lido o arquivo CSV referente à estação meteorológica de Vitória, Espírito Santo.

with myzip.open(file\_csv) as myfile:

Foram lidas apenas algumas colunas dos dados e gravadas em variáveis globais.

globals()[var\_name] = pd.read\_csv(myfile, sep=";", encoding = "ISO-8859-1", header=8, parse\_dates=[col], date\_parser=parser, decimal=",")

globals()[var\_name] = globals()[var\_name][['DATA (YYYY-MM-DD)', 'PRECIPITAÇÃO TOTAL, HORÁRIO (mm)', 'TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)']].rename(columns={"DATA (YYYY-MM-DD)": "ds", "PRECIPITAÇÃO TOTAL, HORÁRIO (mm)": "precipitacao", "TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)": "temperatura"})

E agrupados em um único Dataframe.

dados\_meteor\_all = pd.concat([globals()["df\_INMET\_2018"],globals()["df\_INMET\_2019"],globals()["df\_INMET\_2020"],globals()["df\_INMET\_2021"],globals()["df\_INMET\_2022"]])

No código 1.4.2, as ocorrências com valores nulos foram convertidas para zero. Em seguida, foram identificados problemas em algumas datas, conforme demonstrado na tabela abaixo, então foi feita a exclusão destas ocorrências, filtrando somente ocorrências com temperaturas maiores que zero.

# tratamento de valores nulos

dados\_meteor\_all = dados\_meteor\_all.fillna({'precipitacao':0})

dados\_meteor\_all = dados\_meteor\_all.fillna({'temperatura':0})

# remoção de ocorrências com valores iguais ou inferiores a 0

dados\_meteor\_all = dados\_meteor\_all[dados\_meteor\_all['temperatura'] > 0]

|  |  |  |
| --- | --- | --- |
| **ds** | **precipitacao** | **temperatura** |
| 17/05/2018 | -39996 | -1647,416667 |
| 30/10/2018 | -39996 | -1647,3625 |
| 05/12/2018 | -19998 | -808,675 |
| 26/05/2018 | -9988,2 | -397,8083333 |
| 28/07/2018 | -9999 | -396,3458333 |
| 12/08/2018 | -9998,6 | -396,0708333 |
| 09/03/2018 | -9989,4 | -392,4625 |
| 04/04/2018 | -9997,4 | -391,65 |
| 10/10/2018 | -9999 | -391,3 |
| 13/12/2018 | -9999 | -390,0958333 |

Como podem existir 24 ocorrências para hora de cada data, foi necessário agrupar pela data sem o horário e criar uma média do campo temperatura e soma da precipitação.

dados\_meteor = pd.pivot\_table(

   dados\_meteor\_all,

   index=['ds'],

   aggfunc={'precipitacao': np.mean, 'temperatura': np.mean}

)

# 4. Análise e Exploração dos Dados

Contar a quantidade de linhas do dataset vendas, no código 1.3.4, retorna 1.483 registros.

vendas.info()

vendas.count()

Para listar os anos de dados existentes no dataset vendas, no código 1.3.5, foi criada uma coluna VDA\_ANO e executada uma função para retornar os anos distintos 2018, 2019, 2020, 2021 e 2022.

vendas['VDA\_ANO'] = pd.DatetimeIndex(vendas['VDA\_DATA']).year

anos = vendas['VDA\_ANO'].unique()

Foi feita a soma da quantidade de todas as vendas de todo o dataset apenas para efeito de informação, no código 1.3.6, com resultado 483.432.

vendas['VDA\_QTD'].sum()

Foi feita a soma da quantidade agrupada por ano, no código 1.3.7.

vendas\_ano = vendas.groupby(['VDA\_ANO']).sum()

Foi feita a soma da quantidade agrupada por ano e mês, no código 1.3.8, assim como uma plotagem de gráfico de barra que demonstrou sazonalidade, principalmente em dezembro.

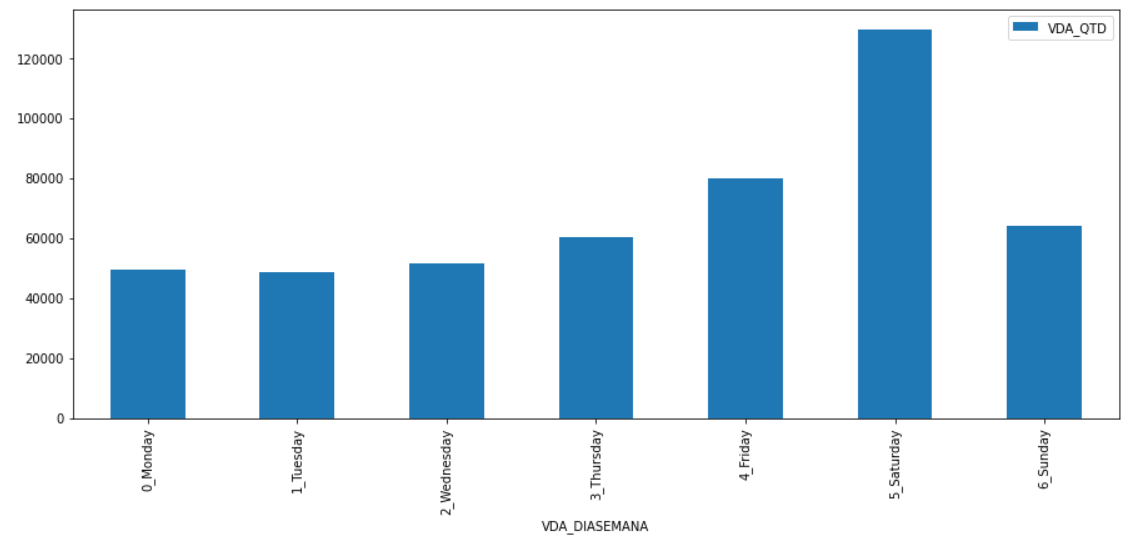
vendas['VDA\_ANOMES'] = vendas['VDA\_DATA'].dt.strftime('%Y-%m')

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_ANOMES', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

Foi feita a soma da quantidade agrupada por dia da semana, no código 1.3.9, assim como uma plotagem de gráfico de barra que demonstrou sazonalidade, principalmente no sábado.

vendas['VDA\_DIASEMANA'] = vendas['VDA\_DATA'].dt.dayofweek.astype(str) +'\_'+ vendas['VDA\_DATA'].dt.day\_name()

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_DIASEMANA', aggfunc='sum').plot(kind='bar', figsize=(15, 6))



Foi feita a soma da quantidade agrupada por semana em padrão ISO, no código 1.3.10. As semanas vão de 1 a 52 e, quando ocorre a semana 53, foi somada à semana 52. O que foi possível observar sazonalidade no verão, das últimas semanas de dezembro às primeiras semanas de março.

vendas['VDA\_SEMANA'] = vendas['VDA\_DATA'].dt.isocalendar().week

vendas["VDA\_SEMANA"].replace({53: 52}, inplace=True)

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_SEMANA', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

Foi feita a soma da quantidade agrupada por mês do ano, de 1 a 12, e um gráfico, no código 1.3.11. E foi possível observar sazonalidade no verão, de dezembro a março.

vendas['VDA\_MES'] = vendas['VDA\_DATA'].dt.month

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_MES', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

#plt.tight\_layout()

plt.show()

Foi feito um gráfico de temperatura por data, no código 1.4.3. E foi possível observar temperaturas mais elevadas no verão, que estão em sintonia com a sazonalidade observada nas vendas.

dados\_meteor\_temp = pd.pivot\_table(

   dados\_meteor\_all,

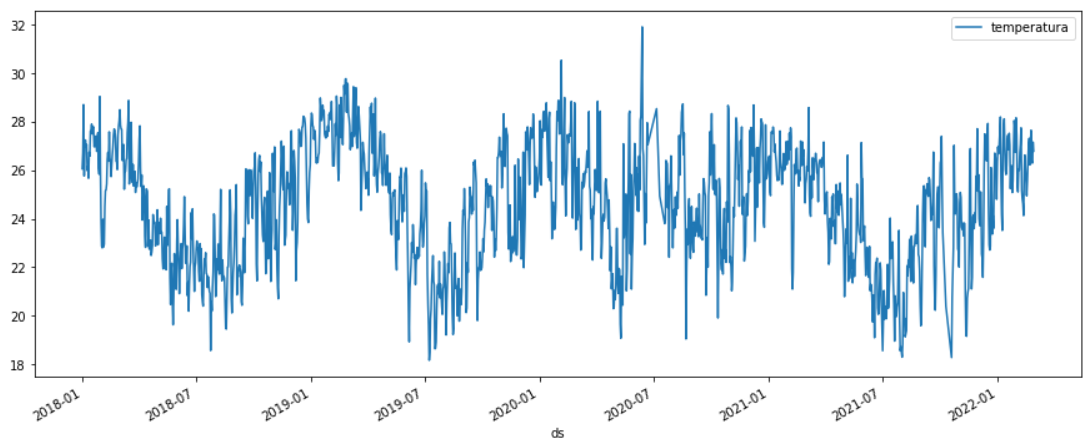
   index=['ds'],

   aggfunc={'temperatura': np.mean}

)

dados\_meteor\_temp.plot(kind='line', figsize=(15, 6))

plt.show()



Foi feito um gráfico de precipitação por data, no código 1.4.4. E não foi possível observar um padrão.

dados\_meteor\_temp = pd.pivot\_table(

   dados\_meteor\_all,

   index=['ds'],

   aggfunc={'precipitacao': np.sum}

)

dados\_meteor\_temp.plot(kind='line', figsize=(15, 6))

plt.show()

# 

# 5. Criação de Modelos de Machine Learning

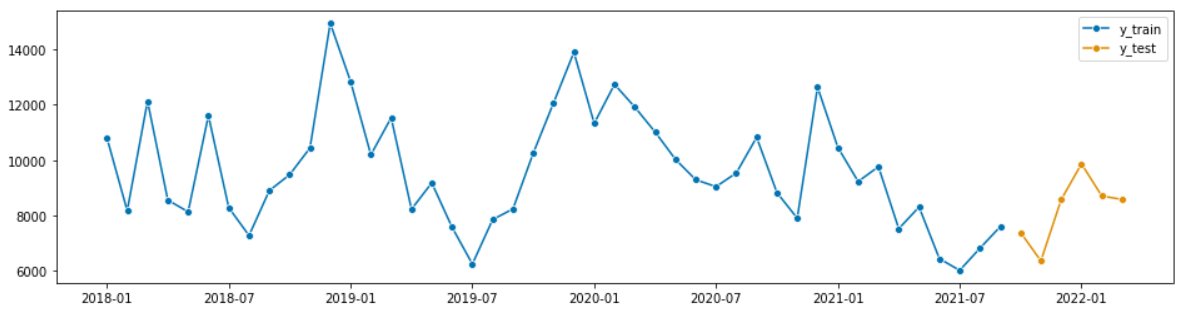
Para a previsão de vendas, foram utilizados vários dos algoritmos do *framework* sktime como Naive, AUTOARIMA, AUTOETS, Exponential Smoothing, BATS e Prophet, no grupo de código 2.

Os dados meteorológicos foram utilizados apenas com o algoritmo Facebook Prophet no grupo de código 3, executado do módulo original, de forma independente do sktime.

## 5.1. sktime

O sktime fornece um framework de código aberto fácil de usar, flexível e modular para uma ampla variedade de tarefas de aprendizagem de máquina de séries temporais. Ele oferece interfaces compatíveis com scikit-learn e ferramentas de composição de modelos, com o objetivo de tornar o ecossistema mais utilizável e interoperável como um todo.

Foi feita a soma da quantidade agrupada por ano e mês, a separação de duas partes de dados y\_train e y\_test, além de um gráfico representativo, no código 2.2. y\_train é a parte de dados ao qual os algoritmos utilizarão para o treino e y\_test é a parte de dados ao qual os algoritmos utilizarão para testar a qualidade da previsão confrontando com os dados previstos. Abaixo o gráfico resultante, representando os dados reais de vendas.



## 5.1.1. Modelo Naive

Foram feitas previsões de vendas com 4 estratégias diferentes disponíveis no modelo Naive, last, seasonal\_last, mean e drift, no código 2.4. E foram utilizadas funções de validação para mensuração da qualidade das previsões.

# Naive Forecaster com estratégia ''last''

naive\_forecaster\_last = NaiveForecaster(strategy="last")

naive\_forecaster\_last.fit(y\_train)

y\_last = naive\_forecaster\_last.predict(fh)

print(\*['Naive Estratégia last: ', 'mse =', mse(y\_test, y\_last), 'rmse =', rmse(y\_test, y\_last), 'mae =', mae(y\_test, y\_last), 'mape =', mape(y\_test, y\_last), 'mdape =', mdape(y\_test, y\_last)])

## 5.1.2. Modelo AUTOARIMA

Foi feita previsão de vendas com o algoritmo AUTOARIMA, que automaticamente seleciona a melhor parametrização do modelo ARIMA, no código 2.5. E foram utilizadas funções de validação para mensuração da qualidade da previsão.

arima\_1 = AutoARIMA(sp=12, suppress\_warnings=True)

arima\_1.fit(y\_train)

y\_arima\_1 = arima\_1.predict(fh)

print(\*[AUTOARIMA: ', 'mse =', mse(y\_test, y\_arima\_1), 'rmse =', rmse(y\_test, y\_arima\_1), 'mae =', mae(y\_test, y\_arima\_1), 'mape =', mape(y\_test, y\_arima\_1), 'mdape =', mdape(y\_test, y\_arima\_1)])

## 5.1.3. Modelo AUTOETS

Foi feita previsão de vendas com o algoritmo AUTOETS, no código 2.6. E foram utilizadas funções de validação para mensuração da qualidade da previsão.

ets\_1 = AutoETS(auto=True, sp=12, n\_jobs=-1)

ets\_1.fit(y\_train)

y\_ets\_1 = ets\_1.predict(fh)

print(\*[AUTOETS: ', 'mse =', mse(y\_test, y\_ets\_1), 'rmse =', rmse(y\_test, y\_ets\_1), 'mae =', mae(y\_test, y\_ets\_1), 'mape =', mape(y\_test, y\_ets\_1), 'mdape =', mdape(y\_test, y\_ets\_1)])

## 5.1.4. Modelo BATS

Foi feita previsão de vendas com o algoritmo BATS, no código 2.7. BATS é um acrônimo para:

- *Box-Cox transformation*

- *ARMA erros*

- *Trend*

- *Seasonal components*

E foram utilizadas funções de validação para mensuração da qualidade da previsão.

bats\_1 = BATS(sp=12, use\_trend=True, use\_box\_cox=False)

bats\_1.fit(y\_train)

y\_bats\_1 = bats\_1.predict(fh)

print(\*[BATS: ', 'mse =', mse(y\_test, y\_bats\_1), 'rmse =', rmse(y\_test, y\_bats\_1), 'mae =', mae(y\_test, y\_bats\_1), 'mape =', mape(y\_test, y\_bats\_1), 'mdape =', mdape(y\_test, y\_bats\_1)])

## 5.1.5. Modelo Exponential Smoothing

Foi feita previsão de vendas com o algoritmo Exponential Smoothing, no código 2.8. E foram utilizadas funções de validação para mensuração da qualidade da previsão.

trend\_1 = ExponentialSmoothing(trend='mul', seasonal='mul', sp=6)

trend\_1.fit(y\_train)

y\_trend\_1 = trend\_1.predict(fh)

print(\*[Exponential Smoothing: ', 'mse =', mse(y\_test, y\_trend\_1), 'rmse =', rmse(y\_test, y\_trend\_1), 'mae =', mae(y\_test, y\_trend\_1), 'mape =', mape(y\_test, y\_trend\_1), 'mdape =', mdape(y\_test, y\_trend\_1)])

## 5.1.6. Modelo Prophet com sktime

Foi feita previsão de vendas com o algoritmo Prophet, no código 2.9. E foram utilizadas funções de validação para mensuração da qualidade da previsão.

y2\_train, y2\_test = temporal\_train\_test\_split(y, test\_size=0.1)

fh2 = ForecastingHorizon(y2\_test.index, is\_relative=False)

# Convert index to pd.DatetimeIndex

z = y.copy()

z = z.to\_timestamp(freq="M")

z\_train, z\_test = temporal\_train\_test\_split(z, test\_size=0.1)

prophet\_1 = Prophet(

    seasonality\_mode="multiplicative",

    n\_changepoints=int(len(y2\_train) / 12),

    add\_country\_holidays={"country\_name": "Brazil"},

    yearly\_seasonality=True,

    weekly\_seasonality=True,

    daily\_seasonality=False,

)

prophet\_1.fit(z\_train)

y\_prophet\_1 = prophet\_1.predict(fh2.to\_relative(cutoff=y2\_train.index[-1]))

y\_prophet\_1.index = y2\_test.index

print(\*['Prophet: ', 'mse =', mse(y\_test, y\_prophet\_1), 'rmse =', rmse(y\_test, y\_prophet\_1), 'mae =', mae(y\_test, y\_prophet\_1), 'mape =', mape(y\_test, y\_prophet\_1), 'mdape =', mdape(y\_test, y\_prophet\_1)])

## 5.2. Facebook Prophet com Dados Meteorológicos

O Prophet é um procedimento para prever dados de séries temporais com base em um modelo aditivo em que as tendências não lineares são ajustadas à sazonalidade anual, semanal e diária, além dos efeitos de feriados. Funciona melhor com séries temporais com fortes efeitos sazonais e várias temporadas de dados históricos. O Prophet é robusto a dados ausentes e mudanças na tendência e normalmente lida bem com valores discrepantes.

O Prophet é um software de código aberto lançado pela equipe Core Data Science do Facebook. Está disponível para download no CRAN e no PyPI.

Foi feita junção dos dados de vendas com os dados meteorológicos precipitação e temperatura, no código 3.2.

df\_p2 = df\_p2.merge(dados\_meteor, on='ds', how='left').fillna({'y':0})

Foi feito preenchimento de valores com zero.

df\_p2 = df\_p2.fillna({'precipitacao':0})

df\_p2 = df\_p2.fillna({'temperatura':0})

Foi feita a criação de uma nova coluna fictícia sensação com os valores frio, agradável, quente ou muito quente, baseados na coluna existente temperatura.

df\_p2.loc[df\_p2['temperatura'] == np.NaN, 'sensacao'] = 0 #'indisponivel'

df\_p2.loc[df\_p2['temperatura'] <= 0, 'sensacao'] = 0 #'indisponivel'

df\_p2.loc[df\_p2['temperatura'] > 0, 'sensacao'] = 1 # 'frio'

df\_p2.loc[df\_p2['temperatura'] >= 20, 'sensacao'] = 2 # 'agradavel'

df\_p2.loc[df\_p2['temperatura'] >= 30, 'sensacao'] = 3 # 'quente'

df\_p2.loc[df\_p2['temperatura'] >= 35, 'sensacao'] = 4 # 'muito quente'

Foi feita a criação de um array de parâmetros, no código 3.3, para a realização de previsões e validações de performance com a finalidade de se encontrar alguns dos parâmetros ideais a serem utilizados em previsões futuras.

param\_grid = {

    'changepoint\_prior\_scale': [ 0.05, 0.07, 0.08, 0.1],

    #'holidays\_prior\_scale'   : [0.01,  0.1,   5.0,  10.0],

    'seasonality\_prior\_scale': [0.05,  0.1,  0.5, 5.0, 7.0, 10.0]

    #'seasonality\_mode'       : ['multiplicative', 'additive']

}

Ainda no código 3.3, foram feitas as previsões com a função *Prophet* e as validações com as funções *cross\_validation* e *performance\_metrics*.

df\_cv\_p0 = cross\_validation(p0, period='365 days', horizon = '30 days', parallel="processes")

df\_p = performance\_metrics(df\_cv\_p0, rolling\_window=0.1)

Importante ressaltar a importante inclusão dos regressores precipitação, temperatura e sensação às previsões.

p0.add\_regressor('precipitacao')

p0.add\_regressor('temperatura')

p0.add\_regressor('sensacao')

O resultado das análises com as diferentes parametrizações foi apresentado para a escolha da parametrização ideal, no código 3.4.

tuning\_results.style

Com a parametrização ideal, é feito o Fit no código 3.5.

p2 = Prophet(growth='linear',

            yearly\_seasonality=True,

            weekly\_seasonality=True,

            daily\_seasonality=False,

            holidays=None,

            seasonality\_mode='multiplicative',

            seasonality\_prior\_scale=10,

            #holidays\_prior\_scale=5,

            changepoint\_prior\_scale=.1,

            mcmc\_samples=0

            ).add\_seasonality(name='monthly',

                                  period=30.5,

                                  fourier\_order=3,

                                  #prior\_scale=10,

                                  mode='additive')

p2.add\_regressor('precipitacao')

p2.add\_regressor('temperatura')

p2.add\_regressor('sensacao')

p2.fit(df\_p2)

Após o Fit, é feita a previsão de vendas, no código 3.6.

future\_p2 = df\_p2

forecast\_p2 = p2.predict(future\_p2)

forecast\_p2[forecast\_p2['ds'] >= '2022-01-01']

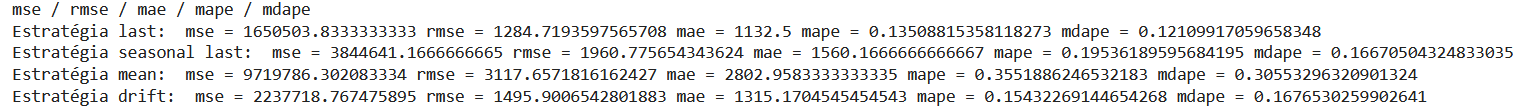
# 6. Interpretação dos Resultados

## 6.1. sktime

A qualidade da previsão de vendas do programa com sktime foi avaliada através das métricas conhecidas na comunidade como MAE e MAPE.

O resultado das análises com as diferentes opções de algoritmos do sktime foi apresentado, nos códigos 2.4, 2.5, 2.6, 2.7, 2.8 e 2.9.

## 6.1.1. Modelo Naive



## 6.1.2. Modelo AUTOARIMA



## 6.1.3. Modelo AUTOETS



## 6.1.4. Modelo BATS



## 6.1.5. Modelo Exponential Smoothing



## 6.1.6. Modelo Prophet

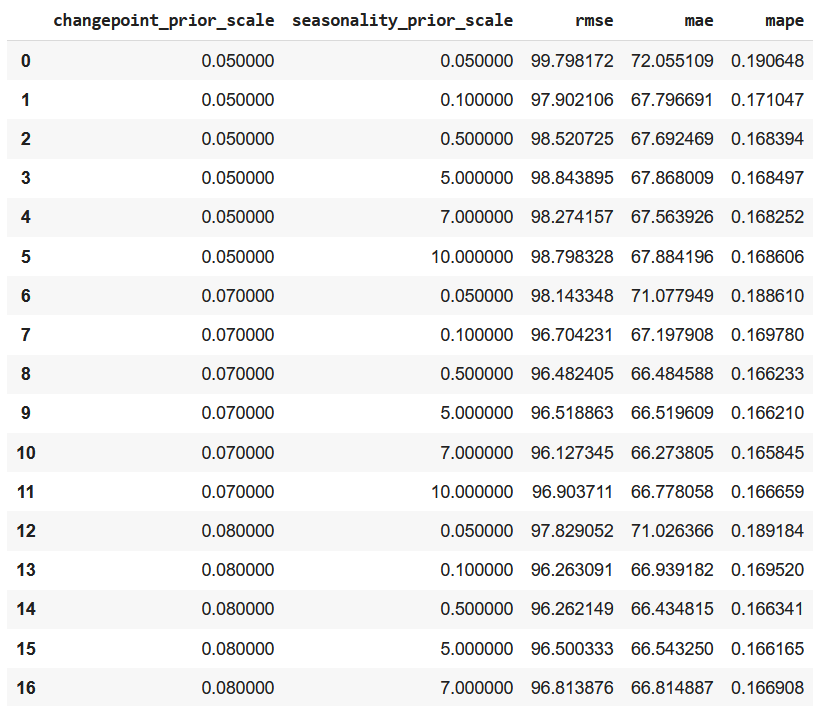


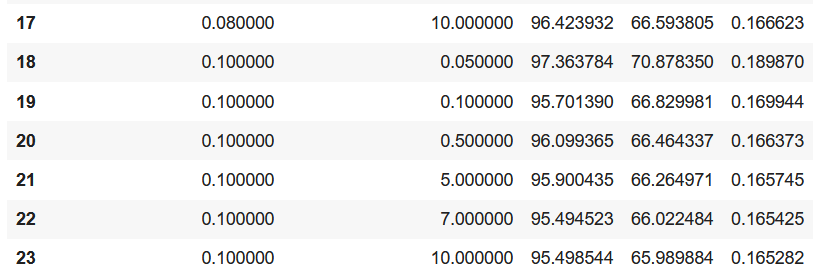
## 6.2. Facebook Prophet

A qualidade da previsão de vendas do programa com Facebook Prophet foi avaliada através das métricas conhecidas na comunidade como MAE e MAPE.

O resultado das análises com as diferentes parametrizações foi apresentado para a escolha da parametrização ideal, no código 3.4.

tuning\_results.style





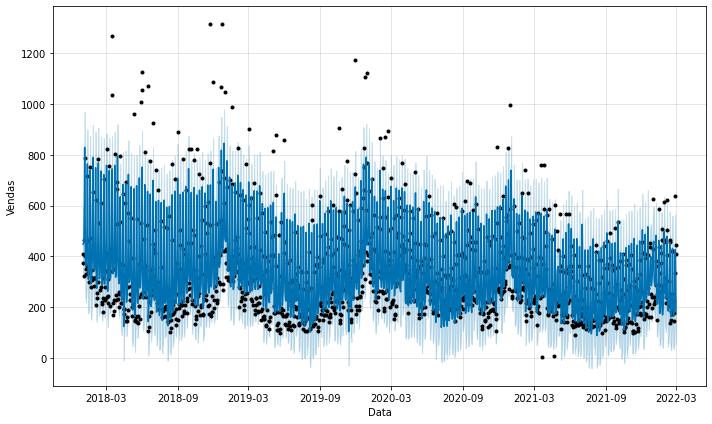
Os parâmetros change\_point\_prior\_scale e seasonality\_prior\_scale do Facebook Prophet que resultaram em menores MAE e MAPE foram selecionados como parametrização ideal para futuras previsões para este grupo de produtos.

# 7. Apresentação dos Resultados

O programa apresentou os seguintes resultados para o algoritmo Facebook Prophet.

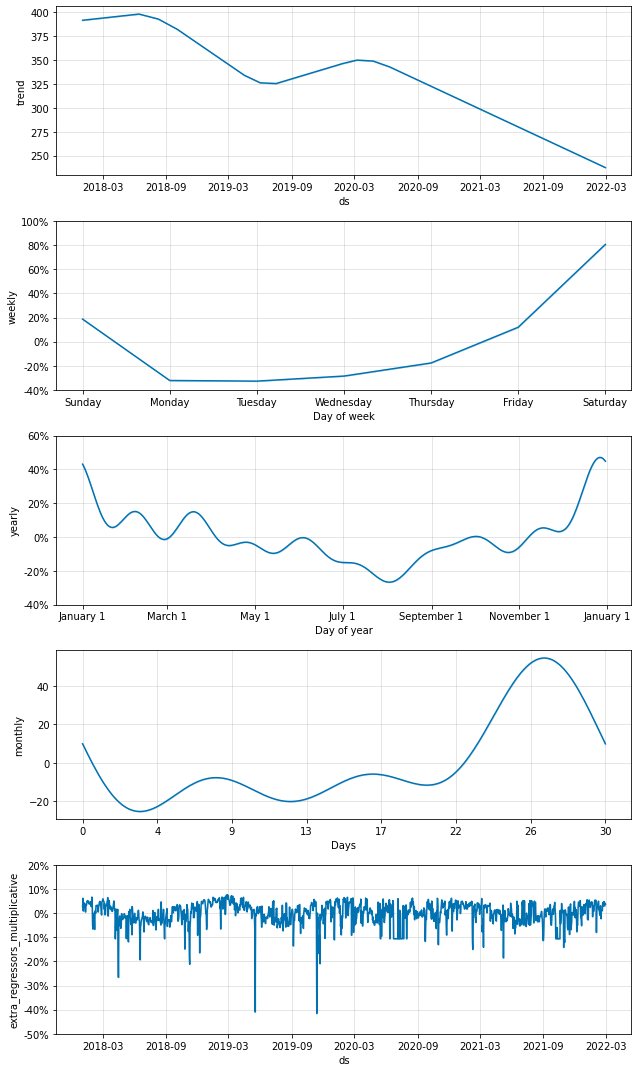
# 7.1. Gráfico com a Previsão de Vendas

No código 3.7, em preto, os pontos utilizados para previsão, em azul escuro o valor previsto, em azul claro a região com valores previstos mínimo e máximo.



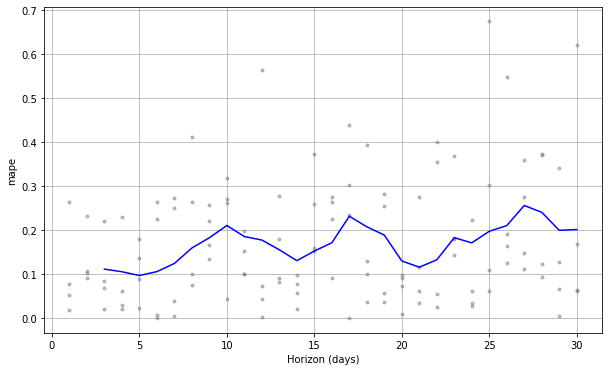
# 7.2. Gráfico com a Tendência e Sazonalidade

No código 3.8, os componentes dos dados e da previsão. Com gráficos, nesta ordem, de tendência e sazonalidades anual, semanal e mensal. Assim como regressores.



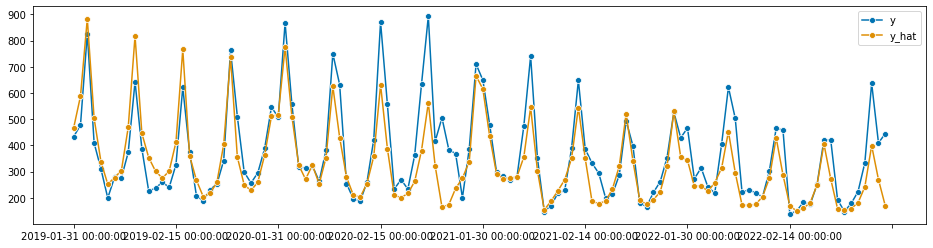
# 7.3. Gráfico com a Métrica MAPE

No código 4.0, o gráfico com a métrica de performance MAPE. Erros percentuais absolutos nos pontos. Em azul a linha do MAPE.



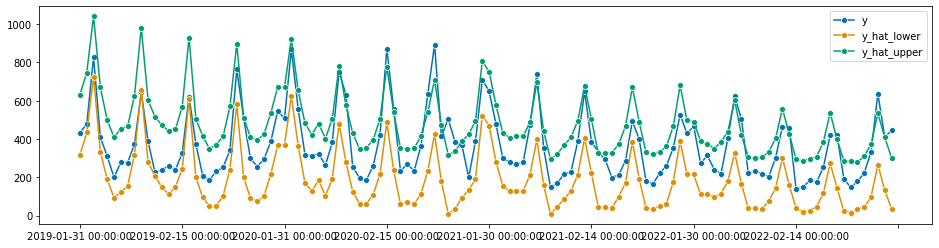
# 7.4. Gráfico com Valores Previstos e Realizados

No código 4.1, gráfico com comparativo entre os valores previstos e realizados. Em laranja, valores previstos. Em azul, valores reais. É possível observar uma boa previsão nos períodos (*cutoffs*) analisados.



# 7.5. Gráfico com Valores Previstos e Realizados

No código 4.2, gráfico com comparativo entre os valores previstos mínimos e máximos e valores realizados. Em laranja, valores previstos mínimos. Em verde, valores previstos máximos. Em azul, valores reais. É possível observar uma boa previsão nos períodos (*cutoffs*) analisados.



# 8. Links

Vídeo

https://youtu.be/3Wn1kI24PjA

Repositório Github

https://github.com/lovatti/tcc\_puc\_mg

# REFERÊNCIAS

Peckham, James, **The Consumer Speaks.** Journal of Marketing, 1963.

CORSTEN, Daniel; GRUEN, Thomas; **Desperately seeking shelf availability: An examination of the extent, the causes, and the efforts to address retail out-of-stock**; International Journal of Retail & Distribution Management, 2003.

Get Started. sktime, 2022. Programação. Disponível em: https://www.sktime.org/en/stable/get\_started.html. Acesso em: 01, março de 2022.

Get Started in Python. Facebook Prophet, 2022. Programação. Disponível em: https://facebook.github.io/prophet/docs/quick\_start.html. Acesso em: 01, março de 2022.

# APÊNDICE

**Programação/Scripts**

**SQL**

PROCEDURE SP\_EXPT\_VENDA (

PAR\_FROM\_DATE IN DATE,

PAR\_TO\_DATE IN DATE,

PAR\_COD\_LOJA IN NUMBER DEFAULT NULL,

PAR\_COD\_FOR IN NUMBER DEFAULT NULL

) IS

T\_TYPE\_FILE\_TARGET UTL\_FILE.FILE\_TYPE;

V\_NOME\_FILE\_TARGET NVARCHAR2(100);

V\_DIR\_FILE NVARCHAR2(100);

V\_ERRO CLOB;

/\* CURSOR DE BUSCA DO ARQUIVO \*/

CURSOR CUR IS

SELECT

'VDA\_DATA' || '|' || 'VDA\_LOJA' || '|' || 'VDA\_PROD' || '|' || 'VDA\_QTD' AS LINE\_TARGET

FROM

DUAL

UNION ALL

SELECT

TO\_CHAR(VDA\_DATA,'YYYY-MM-DD')

|| '|"' || TO\_CHAR(VDA\_LOJA) || '"' || '|"' || TO\_CHAR(VDA\_PROD) || '"'

|| '|' || TO\_CHAR((NVL(VDA\_QTD,0)),'FM99999999990.000')

FROM (

SELECT

TRUNC(VDA\_DATAHORA) AS VDA\_DATA,

--VDA\_LOJA || '\_' || VDA\_PROD as VDA\_LOJA\_PROD,

99 AS VDA\_LOJA,

--VDA\_PROD,

97 AS VDA\_PROD, -- quando exportei tinha 97 produtos

SUM(VDA\_QTD) AS VDA\_QTD

FROM

PROJETOS.V\_TAB\_VENDA\_GERAL VD

WHERE

VD.VDA\_DATAHORA BETWEEN PAR\_FROM\_DATE AND PAR\_TO\_DATE + 86399 / 86400

AND VD.VDA\_KIT IN ( 0,2 )

AND VD.VDA\_PROD IN (

SELECT

PROD\_CODIGO

FROM

PROJETOS.TAB\_PRODUTO T

WHERE

T.PROD\_SECAO = 14

AND T.PROD\_GRUPO = 2

AND T.PROD\_SUBGRUPO = 2

)

GROUP BY TRUNC(VDA\_DATAHORA),

99,

97

) QRY;

BEGIN

IF PAR\_FROM\_DATE IS NULL THEN

V\_ERRO := 'Data De não informada.';

DBMS\_OUTPUT.PUT\_LINE(V\_ERRO);

RETURN;

END IF;

IF PAR\_TO\_DATE IS NULL THEN

V\_ERRO := 'Data Até não informada.';

DBMS\_OUTPUT.PUT\_LINE(V\_ERRO);

RETURN;

END IF;

-- SE A DATA INFORMADA ATÉ FOR MENOR QUE A DATA DE, ENTÃO SAIR

IF PAR\_TO\_DATE < PAR\_FROM\_DATE THEN

V\_ERRO := 'Data ATÉ não pode ser menor que data DE.';

DBMS\_OUTPUT.PUT\_LINE(V\_ERRO);

RETURN;

END IF;

BEGIN

V\_ERRO := NULL;

/\* NOMES E DIRETÓRIO DOS ARQUIVOS \*/

V\_NOME\_FILE\_TARGET := 'lovatti\_venda\_sorvetes\_' || TO\_CHAR(PAR\_FROM\_DATE,'YYYYMMDD') || '\_' || TO\_CHAR(PAR\_TO\_DATE,'YYYYMMDD') || '.csv';

V\_DIR\_FILE := '/cloudfs/dbx/backup/EXPORTACOES/Lovatti';

/\* ABRE OS ARQUIVOS PARA GRAVAÇÃO \*/

T\_TYPE\_FILE\_TARGET := UTL\_FILE.FOPEN(V\_DIR\_FILE,V\_NOME\_FILE\_TARGET,'W');

/\* LOOP NO CURSOR PARA GRAVAÇÃO DAS LINHAS \*/

FOR REG IN CUR LOOP

/\* ARQUIVO TARGET\_SERIES \*/

UTL\_FILE.PUTF(T\_TYPE\_FILE\_TARGET,REG.LINE\_TARGET);

UTL\_FILE.NEW\_LINE(T\_TYPE\_FILE\_TARGET);

END LOOP;

/\* FECHAR OS ARQUIVOS \*/

UTL\_FILE.FFLUSH(T\_TYPE\_FILE\_TARGET);

UTL\_FILE.FCLOSE(T\_TYPE\_FILE\_TARGET);

EXCEPTION

WHEN OTHERS THEN

V\_ERRO := 'Erro ao criar os arquivos csv em disco.'; -- GRAVA EVENTO DE ERRO

DBMS\_OUTPUT.PUT\_LINE(V\_ERRO);

RAISE\_APPLICATION\_ERROR(-20001,V\_ERRO,TRUE);

RETURN; -- FINALIZA A PROCEDURE

END;

END SP\_EXPT\_VENDA;

**PYTHON**

# -\*- coding: utf-8 -\*-

"""TCC\_Lovatti.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Ljz-LCUGXA88j\_Lx-7vcTO-Te\_8DRcC0

# \*\*1. Coleta e Análise de Dados\*\*

1.1. Mostrar versão do Python

"""

import sys

print("Python Version: ", sys.version)

"""1.2. Importar alguns módulos

"""

import pandas as pd

import numpy as np

"""\*\*1.3. Coleta das Vendas\*\*

1.3.1. Conectar ao google drive e à pasta onde estão os arquivos de vendas que foram extraídos do banco de dados

"""

from google.colab import drive

drive.mount('/content/drive/')

import os

os.chdir('drive/My Drive/TCC\_LOVATTI')

"""1.3.2. Listar todos os arquivos CSV de vendas"""

import glob

all\_files = glob.glob('\*sorvetes\*.csv')

all\_files.sort()

all\_files

"""1.3.3. Ler os dados contidos nos arquivos"""

import pandas as pd

from datetime import datetime

custom\_date\_parser = lambda x: datetime.strptime(x, '%Y-%m-%d')

def r(f):

return pd.read\_csv(f, sep='|', parse\_dates=['VDA\_DATA'], date\_parser=custom\_date\_parser)

vendas = pd.concat(map(r, all\_files))

vendas.sort\_values(by=['VDA\_DATA'])

# apresentar informações do arquivo

vendas.info()

# apresentar algumas linhas de dados e os nomes das colunas

vendas.head(5)

#vendas.tail(30)

"""1.3.4. Contar a quantidade de linhas do dataframe"""

# contar a quantidade de linhas

vendas.count()

"""1.3.5. Listar anos de dados"""

vendas['VDA\_ANO'] = pd.DatetimeIndex(vendas['VDA\_DATA']).year

anos = vendas['VDA\_ANO'].unique()

anos

"""1.3.6.Somar quantidade de vendas"""

vendas['VDA\_QTD'].sum()

"""1.3.7. Somar quantidade de vendas por ano"""

vendas\_ano = vendas.groupby(['VDA\_ANO']).sum()

vendas\_ano

"""1.3.8. Plotar um gráfico de quantidade de vendas por data"""

import matplotlib.pyplot as plt

vendas['VDA\_ANOMES'] = vendas['VDA\_DATA'].dt.strftime('%Y-%m')

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_ANOMES', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

#plt.tight\_layout()

plt.show()

"""1.3.9. Plotar um gráfico de quantidade de vendas por dia da semana"""

vendas['VDA\_DIASEMANA'] = vendas['VDA\_DATA'].dt.dayofweek.astype(str) +'\_'+ vendas['VDA\_DATA'].dt.day\_name()

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_DIASEMANA', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

#plt.tight\_layout()

plt.show()

"""1.3.10. Plotar um gráfico de quantidade de vendas por semana do ano - semanas agrupadas"""

vendas['VDA\_SEMANA'] = vendas['VDA\_DATA'].dt.isocalendar().week

vendas["VDA\_SEMANA"].replace({53: 52}, inplace=True)

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_SEMANA', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

#plt.tight\_layout()

plt.show()

"""1.3.11. Plotar um gráfico de quantidade de vendas por mês do ano - meses agrupados"""

vendas['VDA\_MES'] = vendas['VDA\_DATA'].dt.month

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_MES', aggfunc='sum').plot(kind='bar', figsize=(15, 6))

#plt.tight\_layout()

plt.show()

"""1.3.12. Plotar um gráfico de quantidade de vendas por semana do ano - semanas separadas"""

vendas['VDA\_SEMANA2'] = vendas['VDA\_DATA'].dt.isocalendar().week.astype(str).str.zfill(2)

vendas['VDA\_SEMANA2'] = vendas['VDA\_DATA'].dt.isocalendar().year.astype(str) +'-'+ vendas['VDA\_SEMANA2']

vendas.pivot\_table(values = 'VDA\_QTD', index = 'VDA\_SEMANA2', aggfunc='sum').sort\_values(('VDA\_SEMANA2'), ascending=True).plot(kind='bar', figsize=(20, 6))

#plt.tight\_layout()

plt.show()

"""\*\*1.4. Coleta dos Dados Meteorológicos\*\*

1.4.1. Download dados meteorologicos do INMET

Necessário:

- baixar os arquivos zip para o drive

- ler o arquivo com o nome correto da cidade de Vitória (não está 100% padronizado entre os arquivos)

- converter para um dataframe (o formato de data não está 100% padronizado entre os arquivos)

"""

from datetime import datetime

custom\_date\_parser\_1 = lambda x: datetime.strptime(x, '%Y-%m-%d')

custom\_date\_parser\_2 = lambda x: datetime.strptime(x, '%Y/%m/%d')

import requests

import zipfile

for ano in range(2018, 2023):

file\_url = f"https://portal.inmet.gov.br/uploads/dadoshistoricos/{ano}.zip"

file\_gdrive = f"INMET-{ano}.zip"

if not os.path.exists(file\_gdrive):

r = requests.get(file\_url, stream = True)

with open(file\_gdrive, "wb") as file:

for block in r.iter\_content(chunk\_size = 1024):

if block:

file.write(block)

with zipfile.ZipFile(file\_gdrive) as myzip:

# print(myzip.namelist())

if ano in [2018, 2019]:

file\_csv = f"{ano}/INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_31-12-{ano}.CSV"

elif ano in [2022]:

file\_csv = f"INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_28-02-{ano}.CSV"

else:

file\_csv = f"INMET\_SE\_ES\_A612\_VITORIA\_01-01-{ano}\_A\_31-12-{ano}.CSV"

print(file\_csv)

with myzip.open(file\_csv) as myfile:

var\_name = 'df\_INMET\_{}'.format(str(ano))

if ano in [2018]:

col = 'DATA (YYYY-MM-DD)'

parser = custom\_date\_parser\_1

globals()[var\_name] = pd.read\_csv(myfile, sep=";", encoding = "ISO-8859-1", header=8, parse\_dates=[col], date\_parser=parser, decimal=",")

globals()[var\_name] = globals()[var\_name][['DATA (YYYY-MM-DD)', 'PRECIPITAÇÃO TOTAL, HORÁRIO (mm)', 'TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)']].rename(columns={"DATA (YYYY-MM-DD)": "ds", "PRECIPITAÇÃO TOTAL, HORÁRIO (mm)": "precipitacao", "TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)": "temperatura"})

else:

col = 'Data'

parser = custom\_date\_parser\_2

globals()[var\_name] = pd.read\_csv(myfile, sep=";", encoding = "ISO-8859-1", header=8, parse\_dates=[col], date\_parser=parser, decimal=",")

globals()[var\_name] = globals()[var\_name][['Data', 'PRECIPITAÇÃO TOTAL, HORÁRIO (mm)', 'TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)']].rename(columns={"Data": "ds", "PRECIPITAÇÃO TOTAL, HORÁRIO (mm)": "precipitacao", "TEMPERATURA DO AR - BULBO SECO, HORARIA (°C)": "temperatura"})

dados\_meteor\_all = pd.concat([globals()["df\_INMET\_2018"],globals()["df\_INMET\_2019"],globals()["df\_INMET\_2020"],globals()["df\_INMET\_2021"],globals()["df\_INMET\_2022"]])

dados\_meteor\_all

"""1.4.2. Agrupar por data e fazer a média das colunas precipitação e temperatura"""

# Total de linhas da tabela de dados meteorológicos

#dados\_meteor.head(30)

dados\_meteor\_all.count()

# tratamento de valores nulos

dados\_meteor\_all = dados\_meteor\_all.fillna({'precipitacao':0})

dados\_meteor\_all = dados\_meteor\_all.fillna({'temperatura':0})

# remoção de ocorrências com valores iguais ou inferiores a 0

dados\_meteor\_all = dados\_meteor\_all[dados\_meteor\_all['temperatura'] > 0]

dados\_meteor = pd.pivot\_table(

dados\_meteor\_all,

index=['ds'],

aggfunc={'precipitacao': np.sum, 'temperatura': np.mean}

)

dados\_meteor.head(30)

"""1.4.3. Plotar um gráfico com a média de temperatura por data"""

dados\_meteor\_temp = pd.pivot\_table(

dados\_meteor\_all,

index=['ds'],

aggfunc={'temperatura': np.mean}

)

dados\_meteor\_temp.plot(kind='line', figsize=(15, 6))

plt.show()

"""1.4.4. Plotar um gráfico de quantidade de vendas por mês do ano - meses agrupados"""

dados\_meteor\_temp = pd.pivot\_table(

dados\_meteor\_all,

index=['ds'],

aggfunc={'precipitacao': np.sum}

)

dados\_meteor\_temp.plot(kind='line', figsize=(15, 6))

plt.show()

"""# \*\*2. Forecasting com SKTIME\*\*

\*\*sktime\*\*

sktime fornece um framework de código aberto fácil de usar, flexível e modular para uma ampla variedade de tarefas de aprendizagem de máquina de séries temporais. Ele oferece interfaces compatíveis com scikit-learn e ferramentas de composição de modelos, com o objetivo de tornar o ecossistema mais utilizável e interoperável como um todo.

2.1. Importar módulos do sktime com todos os extras

"""

pip install sktime[all\_extras]

"""2.2. Agrupar as vendas por ano e mês no formato YYYY-MM e apresentar um gráfico com as partes de treinamento e teste"""

import warnings

warnings.simplefilter("ignore")

from sktime.forecasting.model\_selection import temporal\_train\_test\_split

#from sktime.utils.plotting.forecasting import plot\_ys

from sktime.utils.plotting import plot\_series

vendas\_anomes = vendas[['VDA\_ANOMES', 'VDA\_QTD']]

vendas\_anomes['VDA\_AM'] = pd.to\_datetime(vendas['VDA\_ANOMES'], format='%Y-%m')

vendas\_anomes

vendas\_anomes = vendas\_anomes.groupby(['VDA\_AM', 'VDA\_ANOMES']).sum().reset\_index()

vendas\_anomes = vendas\_anomes[['VDA\_AM', 'VDA\_ANOMES', 'VDA\_QTD']]

vendas\_anomes

# adicionar linhas inexistentes

for ano in range(2018, 2023):

for mes in range(1, 13):

if ano == 2022 and mes > 3:

break

ano\_mes = str(ano)+'-'+str(mes).zfill(2)

#print(ano\_mes)

i = vendas\_anomes.query("VDA\_ANOMES == '"+ano\_mes+"'")

if i.empty:

#print(ano\_mes)

vendas\_anomes = vendas\_anomes.append({'VDA\_AM': pd.to\_datetime(ano\_mes, format='%Y-%m'), 'VDA\_ANOMES': ano\_mes, 'VDA\_QTD': 0}, ignore\_index=True)

index = pd.period\_range(start='2018-01-01', end='2022-03-01', freq='M')

index

y = pd.Series(vendas\_anomes['VDA\_QTD'].values, index=index)

y = y.sort\_index() # This is an important step if 'time' field is not sorted

#y.head()

y\_train, y\_test = temporal\_train\_test\_split(y, test\_size=0.1) #default 0.25 = 25%

plot\_series(y\_train, y\_test, labels=["y\_train", "y\_test"]);

"""2.3. Apresentar um gráfico com as partes de treinamento e teste sem agrupar as vendas"""

# vendas\_datas = vendas[['VDA\_DATA', 'VDA\_QTD']]

# y = pd.Series(vendas\_datas['VDA\_QTD'].values, index=vendas\_datas['VDA\_DATA'].values)

# y = y.sort\_index() # This is an important step if 'time' field is not sorted

# y\_train, y\_test = temporal\_train\_test\_split(y, test\_size=0.1) #default 0.25 = 25%

# plot\_series(y\_train, y\_test, labels=["y\_train", "y\_test"])

y

"""2.4. Algumas importações do SKTIME"""

# usar mesmas métricas que visualizaremos também mais à frente no outro algoritmo Facebook Prophet

# mse rmse mae mape mdape

# https://www.sktime.org/en/stable/api\_reference/performance\_metrics.html

# Mean squared error (MSE) or root mean squared error (RMSE).

from sktime.performance\_metrics.forecasting import MeanSquaredError

# MAE Mean absolute error (MAE).

from sktime.performance\_metrics.forecasting import MeanAbsoluteError

# MAPE Mean absolute percentage error (MAPE) or symmetric version.

from sktime.performance\_metrics.forecasting import MeanAbsolutePercentageError

# Median absolute percentage error (MdAPE) or symmetric version.

from sktime.performance\_metrics.forecasting import MedianAbsolutePercentageError

from sktime.forecasting.model\_evaluation import evaluate

from sktime.forecasting.model\_selection import ExpandingWindowSplitter

mse = MeanSquaredError(square\_root=False)

rmse = MeanSquaredError(square\_root=True)

mae = MeanAbsoluteError()

mape = MeanAbsolutePercentageError(symmetric=False)

mdape = MedianAbsolutePercentageError(symmetric=False)

#fh = ForecastingHorizon(y\_test.index, is\_relative=False,)

#fh = np.array([1, 2, 3, 4, 5])

fh = np.arange(len(y\_test)) + 1

fh

"""2.4. Modelo NaiveForecaster"""

from sktime.forecasting.naive import NaiveForecaster

from sktime.forecasting.base import ForecastingHorizon

# Naive Forecaster com estratégia ''last''

naive\_forecaster\_last = NaiveForecaster(strategy="last")

naive\_forecaster\_last.fit(y\_train)

y\_last = naive\_forecaster\_last.predict(fh)

# Naive Forecaster com estratégia ''last'' e sp

naive\_forecaster\_seasonal\_last = NaiveForecaster(strategy="last", sp = 12)

naive\_forecaster\_seasonal\_last.fit(y\_train)

y\_seasonal\_last = naive\_forecaster\_seasonal\_last.predict(fh)

# Naive Forecaster com estratégia ''mean''

naive\_forecaster\_mean = NaiveForecaster(strategy="mean", sp=12)

naive\_forecaster\_mean.fit(y\_train)

y\_mean = naive\_forecaster\_mean.predict(fh)

# Naive Forecaster com estratégia ''drift''

naive\_forecaster\_drift = NaiveForecaster(strategy="drift")

naive\_forecaster\_drift.fit(y\_train)

y\_drift = naive\_forecaster\_drift.predict(fh)

plot\_series(y\_train, y\_test, y\_last, y\_seasonal\_last, y\_mean, y\_drift, labels=["y\_train", "y\_test", "y\_pred\_last", "y\_pred\_seasonal\_last", "y\_pred\_mean", "y\_pred\_drift"]);

"""MSE, RMSE, MAE, MAPE, MDAPE - NaiveForecaster"""

print('mse / rmse / mae / mape / mdape')

print(\*['Estratégia last: ', 'mse =', mse(y\_test, y\_last), 'rmse =', rmse(y\_test, y\_last), 'mae =', mae(y\_test, y\_last), 'mape =', mape(y\_test, y\_last), 'mdape =', mdape(y\_test, y\_last)])

print(\*['Estratégia seasonal last: ', 'mse =', mse(y\_test, y\_seasonal\_last), 'rmse =', rmse(y\_test, y\_seasonal\_last), 'mae =', mae(y\_test, y\_seasonal\_last), 'mape =', mape(y\_test, y\_seasonal\_last), 'mdape =', mdape(y\_test, y\_seasonal\_last)])

print(\*['Estratégia mean: ', 'mse =', mse(y\_test, y\_mean), 'rmse =', rmse(y\_test, y\_mean), 'mae =', mae(y\_test, y\_mean), 'mape =', mape(y\_test, y\_mean), 'mdape =', mdape(y\_test, y\_mean)])

print(\*['Estratégia drift: ', 'mse =', mse(y\_test, y\_drift), 'rmse =', rmse(y\_test, y\_drift), 'mae =', mae(y\_test, y\_drift), 'mape =', mape(y\_test, y\_drift), 'mdape =', mdape(y\_test, y\_drift)])

"""2.5. Modelo AUTOARIMA"""

from sktime.forecasting.arima import AutoARIMA

arima\_1 = AutoARIMA(sp=12, suppress\_warnings=True)

arima\_1.fit(y\_train)

y\_arima\_1 = arima\_1.predict(fh)

plot\_series(y\_train, y\_test, y\_arima\_1, labels=["y\_train", "y\_test", "y\_pred\_arima\_1"]);

"""MSE, RMSE, MAE, MAPE, MDAPE - AUTOARIMA"""

print('mse / rmse / mae / mape / mdape')

print(\*['AUTOARIMA: ', 'mse =', mse(y\_test, y\_arima\_1), 'rmse =', rmse(y\_test, y\_arima\_1), 'mae =', mae(y\_test, y\_arima\_1), 'mape =', mape(y\_test, y\_arima\_1), 'mdape =', mdape(y\_test, y\_arima\_1)])

"""2.6. Modelo AUTOETS"""

from sktime.forecasting.ets import AutoETS

ets\_1 = AutoETS(auto=True, sp=12, n\_jobs=-1)

ets\_1.fit(y\_train)

y\_ets\_1 = ets\_1.predict(fh)

plot\_series(y\_train, y\_test, y\_ets\_1, labels=["y\_train", "y\_test", "y\_pred\_ets\_1"]);

"""MSE, RMSE, MAE, MAPE, MDAPE - AUTOETS"""

print('mse / rmse / mae / mape / mdape')

print(\*['AUTOETS: ', 'mse =', mse(y\_test, y\_ets\_1), 'rmse =', rmse(y\_test, y\_ets\_1), 'mae =', mae(y\_test, y\_ets\_1), 'mape =', mape(y\_test, y\_ets\_1), 'mdape =', mdape(y\_test, y\_ets\_1)])

"""2.7. Modelo BATS"""

from sktime.forecasting.bats import BATS

from sktime.forecasting.base import ForecastingHorizon

bats\_1 = BATS(sp=12, use\_trend=True, use\_box\_cox=False)

bats\_1.fit(y\_train)

y\_bats\_1 = bats\_1.predict(fh)

plot\_series(y\_train, y\_test, y\_bats\_1, labels=["y\_train", "y\_test", "y\_pred\_bats\_1"])

"""MSE, RMSE, MAE, MAPE, MDAPE - BATS"""

print('mse / rmse / mae / mape / mdape')

print(\*['BATS: ', 'mse =', mse(y\_test, y\_bats\_1), 'rmse =', rmse(y\_test, y\_bats\_1), 'mae =', mae(y\_test, y\_bats\_1), 'mape =', mape(y\_test, y\_bats\_1), 'mdape =', mdape(y\_test, y\_bats\_1)])

"""2.8. Modelo Exponential Smoothing"""

from sktime.forecasting.exp\_smoothing import ExponentialSmoothing

trend\_1 = ExponentialSmoothing(trend='mul', seasonal='mul', sp=6)

trend\_1.fit(y\_train)

y\_trend\_1 = trend\_1.predict(fh)

plot\_series(y\_train, y\_test, y\_trend\_1, labels=["y\_train", "y\_test", "y\_pred\_trend\_1"]);

"""MSE, RMSE, MAE, MAPE, MDAPE - Exponential Smoothing"""

print('mse / rmse / mae / mape / mdape')

print(\*['Exponential Smoothing: ', 'mse =', mse(y\_test, y\_trend\_1), 'rmse =', rmse(y\_test, y\_trend\_1), 'mae =', mae(y\_test, y\_trend\_1), 'mape =', mape(y\_test, y\_trend\_1), 'mdape =', mdape(y\_test, y\_trend\_1)])

"""2.9. Modelo Prophet"""

from sktime.forecasting.fbprophet import Prophet

from sktime.forecasting.base import ForecastingHorizon

y2\_train, y2\_test = temporal\_train\_test\_split(y, test\_size=0.1)

fh2 = ForecastingHorizon(y2\_test.index, is\_relative=False)

# Convert index to pd.DatetimeIndex

z = y.copy()

z = z.to\_timestamp(freq="M")

z\_train, z\_test = temporal\_train\_test\_split(z, test\_size=0.1)

prophet\_1 = Prophet(

seasonality\_mode="multiplicative",

n\_changepoints=int(len(y2\_train) / 12),

add\_country\_holidays={"country\_name": "Brazil"},

yearly\_seasonality=True,

weekly\_seasonality=True,

daily\_seasonality=False,

)

prophet\_1.fit(z\_train)

y\_prophet\_1 = prophet\_1.predict(fh2.to\_relative(cutoff=y2\_train.index[-1]))

y\_prophet\_1.index = y2\_test.index

plot\_series(y2\_train, y2\_test, y\_prophet\_1, labels=["y\_train", "y\_test", "y\_pred\_prophet\_1"]);

"""MSE, RMSE, MAE, MAPE, MDAPE - Prophet"""

print('mse / rmse / mae / mape / mdape')

print(\*['Prophet: ', 'mse =', mse(y\_test, y\_prophet\_1), 'rmse =', rmse(y\_test, y\_prophet\_1), 'mae =', mae(y\_test, y\_prophet\_1), 'mape =', mape(y\_test, y\_prophet\_1), 'mdape =', mdape(y\_test, y\_prophet\_1)])

"""# \*\*3. Forecasting com Facebook Prophet e Dados Meteorológicos\*\*

O Prophet é um procedimento para prever dados de séries temporais com base em um modelo aditivo em que as tendências não lineares são ajustadas à sazonalidade anual, semanal e diária, além dos efeitos de feriados. Funciona melhor com séries temporais com fortes efeitos sazonais e várias temporadas de dados históricos. O Prophet é robusto a dados ausentes e mudanças na tendência e normalmente lida bem com valores discrepantes.

O Prophet é um software de código aberto lançado pela equipe Core Data Science do Facebook. Está disponível para download no CRAN e no PyPI.

3.1. Instalar o facebook prophet, se necessário

"""

#pip install pystan==2.19.1.1

!pip install prophet

from prophet import Prophet

from prophet.diagnostics import cross\_validation

from prophet.diagnostics import performance\_metrics

"""3.2. Merge dos dados meteorológicos com o dataframe de vendas"""

df\_p2 = vendas[['VDA\_DATA', 'VDA\_QTD']].rename(columns={"VDA\_DATA": "ds", "VDA\_QTD": "y"}).sort\_values(by=['ds'])

df\_p2 = df\_p2[df\_p2['ds'] <= '2022-02-28']

df\_p2 = df\_p2.merge(dados\_meteor, on='ds', how='left').fillna({'y':0})

df\_p2 = df\_p2.fillna({'precipitacao':0})

df\_p2 = df\_p2.fillna({'temperatura':0})

df\_p2.loc[df\_p2['temperatura'] == np.NaN, 'sensacao'] = 0 #'indisponivel'

df\_p2.loc[df\_p2['temperatura'] <= 0, 'sensacao'] = 0 #'indisponivel'

df\_p2.loc[df\_p2['temperatura'] > 0, 'sensacao'] = 1 # 'frio'

df\_p2.loc[df\_p2['temperatura'] >= 20, 'sensacao'] = 2 # 'agradavel'

df\_p2.loc[df\_p2['temperatura'] >= 30, 'sensacao'] = 3 # 'quente'

df\_p2.loc[df\_p2['temperatura'] >= 35, 'sensacao'] = 4 # 'muito quente'

df\_p2.style

"""3.3. Procurar Melhores Parâmetros"""

import itertools

# usado no Hyperparameter tuning

df\_p0 = df\_p2

param\_grid = {

'changepoint\_prior\_scale': [ 0.05, 0.07, 0.08, 0.1],

#'holidays\_prior\_scale' : [0.01, 0.1, 5.0, 10.0],

'seasonality\_prior\_scale': [0.05, 0.1, 0.5, 5.0, 7.0, 10.0]

#'seasonality\_mode' : ['multiplicative', 'additive']

}

# Generate all combinations of parameters

all\_params = [dict(zip(param\_grid.keys(), v)) for v in itertools.product(\*param\_grid.values())]

rmses = [] # Store the RMSEs for each params here

maes = []

mapes = []

index = 0

# Use cross validation to evaluate all parameters

for params in all\_params:

index += 1

print(index)

p0 = Prophet(

growth='linear',

yearly\_seasonality=True,

weekly\_seasonality=True,

daily\_seasonality=False,

holidays=None,

seasonality\_mode='multiplicative',

#seasonality\_prior\_scale=5,

#holidays\_prior\_scale=10,

#changepoint\_prior\_scale=.001,

mcmc\_samples=0,

\*\*params

).add\_seasonality(name='monthly',

period=30.5,

fourier\_order=3,

#prior\_scale=10,

mode='additive')

# .add\_seasonality(name='yearly',

# period=365.25,

# fourier\_order=5,

# #prior\_scale=0.1,

# mode='additive')

#p0.add\_country\_holidays(country\_name='Brazil')

p0.add\_regressor('precipitacao')

p0.add\_regressor('temperatura')

p0.add\_regressor('sensacao')

p0.fit(df\_p0) # Fit model with given params

#df\_cv\_p0 = cross\_validation(p0, cutoffs=cutoffs, horizon='90 days', parallel="processes")

#df\_cv\_p0 = cross\_validation(p0, initial='1460 days', period='365 days', horizon = '30 days', parallel="processes")

df\_cv\_p0 = cross\_validation(p0, period='365 days', horizon = '30 days', parallel="processes")

df\_p = performance\_metrics(df\_cv\_p0, rolling\_window=0.2)

rmses.append(df\_p['rmse'].values[0])

maes.append(df\_p['mae'].values[0])

mapes.append(df\_p['mape'].values[0])

# Find the best parameters

tuning\_results = pd.DataFrame(all\_params)

tuning\_results['rmse'] = rmses

tuning\_results['mae'] = maes

tuning\_results['mape'] = mapes

"""3.4. Apresentação das análises com diferentes parametrizações"""

#tuning\_results.head(150)

#display(tuning\_results)

tuning\_results.style

"""3.5. Fit"""

p2 = Prophet(growth='linear',

yearly\_seasonality=True,

weekly\_seasonality=True,

daily\_seasonality=False,

holidays=None,

seasonality\_mode='multiplicative',

seasonality\_prior\_scale=0.5,

#holidays\_prior\_scale=5,

changepoint\_prior\_scale=.05,

mcmc\_samples=0

).add\_seasonality(name='monthly',

period=30.5,

fourier\_order=3,

#prior\_scale=10,

mode='additive')

# .add\_seasonality(name='yearly',

# period=365.25,

# fourier\_order=3,

# #prior\_scale=0.1,

# mode='multiplicative')

#p2.add\_country\_holidays(country\_name='Brazil')

p2.add\_regressor('precipitacao')

p2.add\_regressor('temperatura')

p2.add\_regressor('sensacao')

p2.fit(df\_p2)

"""3.6. Previsão de vendas"""

future\_p2 = df\_p2

forecast\_p2 = p2.predict(future\_p2)

forecast\_p2[forecast\_p2['ds'] >= '2022-01-01']

"""3.7. Plotar um gráfico com a previsão de vendas"""

fig1\_p2 = p2.plot(forecast\_p2, xlabel='Data', ylabel='Vendas')

"""3.8. Plotar um gráfico com os componentes da previsão de vendas"""

fig2\_p2 = p2.plot\_components(forecast\_p2)

"""3.9. Validação e métricas de performance"""

from fbprophet.diagnostics import cross\_validation

df\_cv\_p2 = cross\_validation(p2, period='365 days', horizon = '30 days', parallel="processes")

df\_cv\_p2.style

from fbprophet.diagnostics import performance\_metrics

df\_perf\_p2 = performance\_metrics(df\_cv\_p2)

df\_perf\_p2.style

"""4.0. Plotar um gráfico com a métrica de performance MAE e MAPE"""

from fbprophet.plot import plot\_cross\_validation\_metric

fig3\_p2 = plot\_cross\_validation\_metric(df\_cv\_p2, metric='mape')

"""4.1. Plotar um gráfico de comparação entre previsto e real nos períodos analisados"""

pht\_y = pd.Series(df\_cv\_p2['y'].values, index=df\_cv\_p2['ds'].values)

pht\_y = pht\_y.sort\_index() # This is an important step if 'time' field is not sorted

pht\_y\_hat = pd.Series(df\_cv\_p2['yhat'].values, index=df\_cv\_p2['ds'].values)

pht\_y\_hat = pht\_y\_hat.sort\_index() # This is an important step if 'time' field is not sorted

plot\_series(pht\_y, pht\_y\_hat, labels=["y", "y\_hat"])

"""4.2 Plotar um gráfico de comparação entre o valor real e os valores previstos possíveis para mínimo e máximo, nos períodos analisados"""

pht\_y = pd.Series(df\_cv\_p2['y'].values, index=df\_cv\_p2['ds'].values)

pht\_y = pht\_y.sort\_index() # This is an important step if 'time' field is not sorted

pht\_y\_hat\_lower = pd.Series(df\_cv\_p2['yhat\_lower'].values, index=df\_cv\_p2['ds'].values)

pht\_y\_hat\_lower = pht\_y\_hat\_lower.sort\_index() # This is an important step if 'time' field is not sorted

pht\_y\_hat\_upper = pd.Series(df\_cv\_p2['yhat\_upper'].values, index=df\_cv\_p2['ds'].values)

pht\_y\_hat\_upper = pht\_y\_hat\_upper.sort\_index() # This is an important step if 'time' field is not sorted

plot\_series(pht\_y, pht\_y\_hat\_lower, pht\_y\_hat\_upper, labels=["y", "y\_hat\_lower", "y\_hat\_upper"])