Nota 0,5

Séries temporais ARIMA

Instalar pmdarima

```
pip install pmdarima
```

```
Requirement already satisfied: pmdarima in
/usr/local/lib/python3.10/dist-packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (2.2.2)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.2)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.13.1)
Requirement already satisfied: statsmodels>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.4)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (2.2.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (71.0.4)
Requirement already satisfied: packaging>=17.1 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima)
(2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22-
>pmdarima) (3.5.0)
Requirement already satisfied: patsy>=0.5.6 in
/usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2-
>pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

Bibliotecas e parâmetros

```
#Bibliotecas

import numpy as np
import pandas as pd

from pmdarima.arima import auto_arima
from pmdarima import arima
from pmdarima.utils import tsdisplay, decomposed_plot, plot_acf,
plot_pacf
from pmdarima.arima import ADFTest


from matplotlib import pyplot as plt
from matplotlib.pylab import rcParams
import matplotlib.dates as mdates

#Tamanho das imagens
rcParams['figure.figsize'] = 14, 6

#Ignorar warnings
import warnings
warnings.filterwarnings('ignore')
```

Conectar com Google Drive

```
# Conectar com o Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly
remount, call drive.mount("/content/gdrive", force_remount=True).
```

Funções

Teste Dickey-Fuller

```
def teste_ADF(serie):
    # Teste de Dickey-Fuller:
    resultado_teste = adfuller(serie, autolag='AIC')
    # Formata o resultado do teste em um dataframe
    resultado_formatado = pd.Series(resultado_teste[0:4],
                                    index=['Estatística do teste','p-
value','Lags',
                                           'Quantidade de observações'])
    # Formata os valores c´riticos do teste
    for significancia,valor in resultado_teste[4].items():
        resultado_formatado['Valor crítico (%s)'%significancia] =
valor
```

```
    # Imprime os resultados
    print ('Resultados do Teste Dickey-Fuller:')
    print (resultado_formatado)
```

Cálculo de métricas de previsão

```
# Métricas de acuidade de previsão
def metricas(previsto, observado):
    erro = previsto - observado                    # erro
    me = np.mean(erro)                             # ME
    mse = np.square(erro).mean()                   # MSE
    rmse = np.sqrt(mse)                            # RMSE
    mae = np.abs(erro).mean()                      # MAE
    mpe = (erro / observado).mean()                # MPE
    mape = np.abs(erro / observado).mean()         # MAPE
    mins = np.amin(np.hstack([previsto[:,None],
                              observado[:,None]]), axis=1)
    maxs = np.amax(np.hstack([previsto[:,None],
                              observado[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)                # MINMAX
    return({'ME':me, 'MSE':mse, 'RMSE':rmse,
            'MAE': mae, 'MPE': mpe, 'MAPE':mape,
            'MIN-MAX':minmax})
```

Ler dados

Dados mensais de requisição diária aos servidores.

```
# Ler os dados
serie_requisicoes_dia =
pd.read_csv('/content/gdrive/MyDrive/DADOS/AD/servidores_requisicoes_dia.csv',
                                    index_col=0, parse_dates=True)

serie_requisicoes_dia.head(3)
```


```
{"summary":"{\n  \"name\": \"serie_requisicoes_dia\",\n  \"rows\": 4383,\n  \"fields\": [\n    {\n      \"column\": \"Data\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"2006-01-01 00:00:00\",\n        \"max\": \"2017-12-31 00:00:00\",\n        \"num_unique_values\": 4383,\n        \"samples\": [\n          \"2007-11-02 00:00:00\",\n          \"2012-08-14 00:00:00\",\n          \"2007-08-20 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"WEB\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 165.7757102347912,\n        \"min\": 842.395,\n        \"max\": 1709.5679999999998,\n
```

```
\"num_unique_values\": 4374,\n        \"samples\": [\n
1357.79,\n          1392.855,\n          1337.078\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 143.69273168412707,\n
\"min\": 5.756999999999999,\n        \"max\": 826.2779999999998,\n
\"num_unique_values\": 2913,\n        \"samples\": [\n
19.296000000000006,\n          95.60299999999998,\n
588.2339999999998\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"APLIC\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 58.550099470055486,\n        \"min\": 1.968,\n
\"max\": 241.58,\n        \"num_unique_values\": 2185,\n
\"samples\": [\n          55.154,\n          26.01,\n          5.776\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"serie_requisicoes_dia"}
```

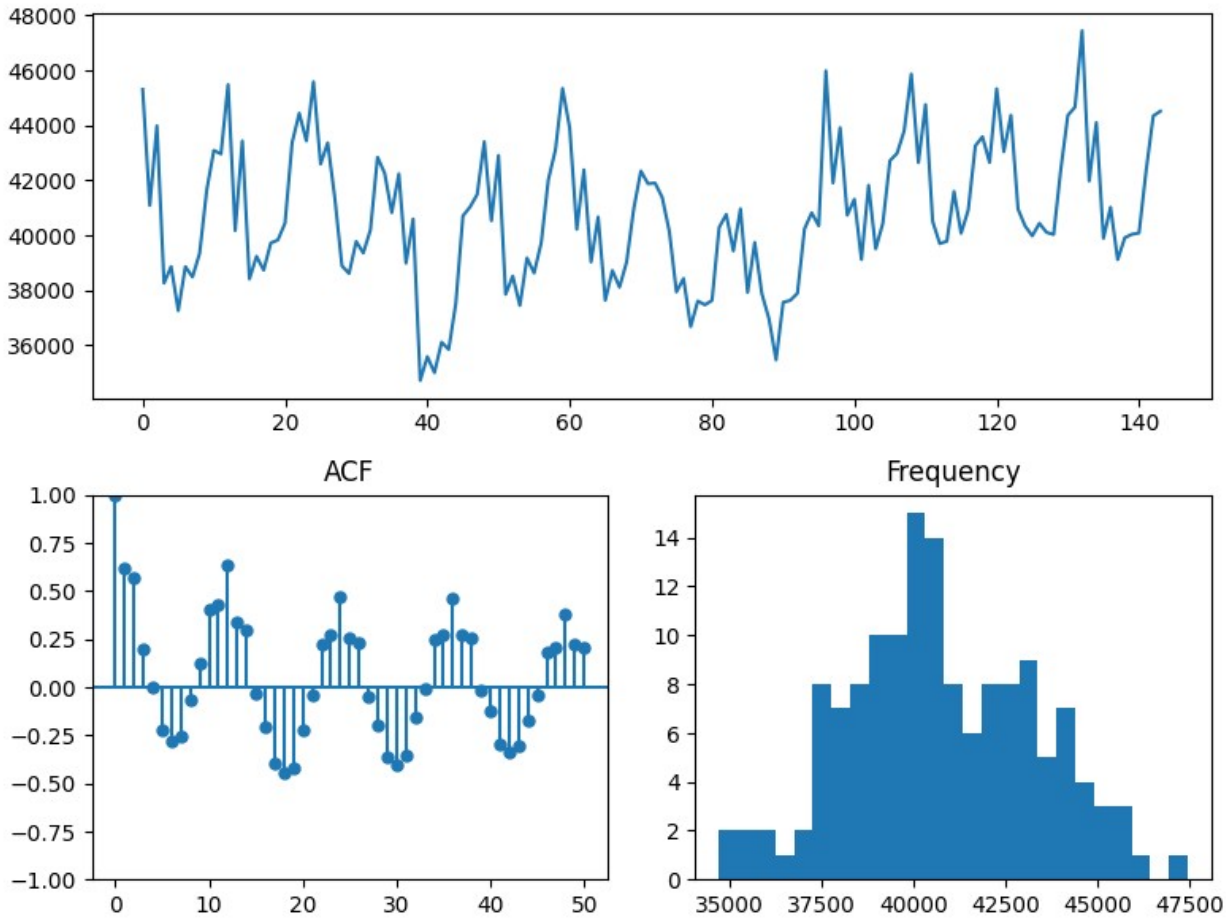

Modelo - Série WEB AUTOARIMA

```
# Seleciona coluna WEB e reformata para requisições mensais (soma)
serie_WEB = serie_requisicoes_dia['WEB'].resample('M').sum()
serie_WEB.columns = ['Valor']
print(serie_WEB.head(3))
```

```
Data
2006-01-31    45304.704
2006-02-28    41078.993
2006-03-31    43978.124
Freq: ME, Name: WEB, dtype: float64
```

Plotar série, ACF e frequência

```
tsdisplay(serie_WEB)
```



Criar o modelo

```
modelo_AUTOARIMA_WEB = auto_arima(serie_WEB,
                      start_p=0,
                      start_q=0,
                      d=0,
                      max_p=6,
                      max_q=6,
                      max_d=2,
                      start_P=1,
                      start_Q=1,
                      D=1,
                      max_P=2, max_D=1, max_Q=2, max_order=5,
                      m=12,
                      seasonal=True,
                      trace=False,
                      error_action='ignore',suppress_warnings=True,
                      stepwise=True)
```

Avaliar desempenho

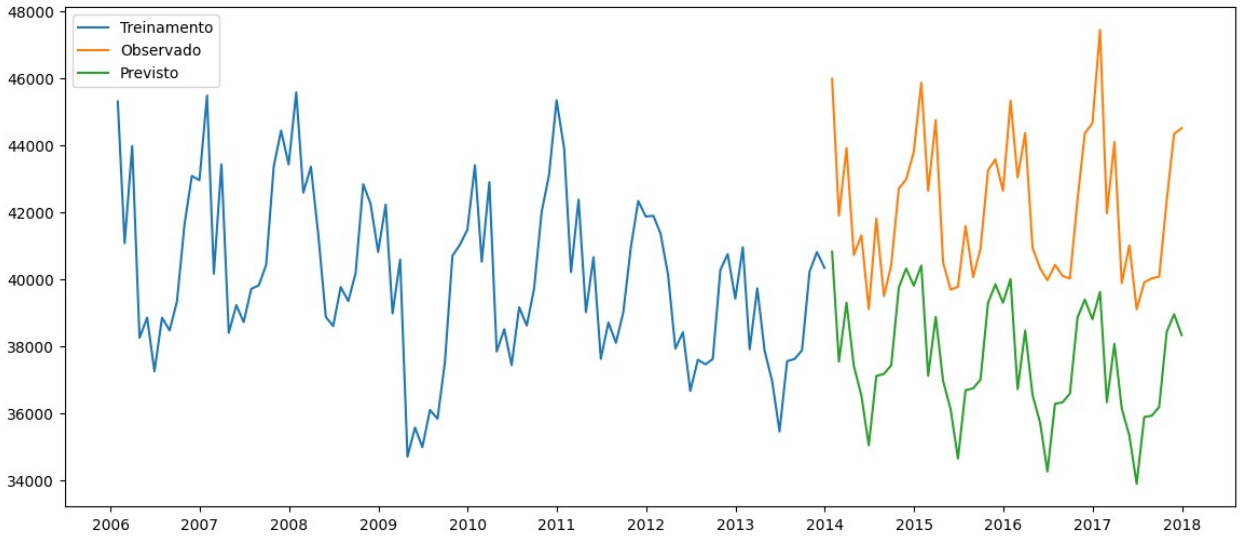
Criar séries de treinamento teste e previsão

- Usar 2/3 da série original para construir o modelo (serie_trein)
- Usar 1/3 da série original para comparar com previsões (serie_test)
- Criar criar série de previsões usando o modelo (serie_prev)

```
# Criar séries de treinamento, teste e previsão

# Definir tamanho das séries de treinamento e teste
lin, = serie_WEB.shape
tam_treino = int(np.ceil(lin*2/3))
tam_teste = int(lin - np.ceil(lin*2/3))

# Criar séries de treinamento e teste
serie_treino_AUTOARIMA_WEB = serie_WEB[:tam_treino]
serie_teste_AUTOARIMA_WEB = serie_WEB[-tam_teste:]

# Cria série de previsão
modelo_AUTOARIMA_WEB.fit(serie_treino_AUTOARIMA_WEB)
previsao_AUTOARIMA_WEB =
modelo_AUTOARIMA_WEB.predict(n_periods=tam_teste)
serie_previsao_AUTOARIMA_WEB = pd.DataFrame(previsao_AUTOARIMA_WEB)
```

Observar gráfico de previsões

```
# Plot
fig, ax = plt.subplots()
ax.plot(serie_treino_AUTOARIMA_WEB, label='Treinamento')
ax.plot(serie_teste_AUTOARIMA_WEB, label='Observado')
ax.plot(serie_previsao_AUTOARIMA_WEB, label='Previsto')
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.legend(loc='upper left', fontsize=10);
```



Calcular e imprimir métricas de previsão (WEB AUTOARIMA)

```
# Calcular métricas de previsão WEB AUTOARIMA
metricas_AUTOARIMA_WEB =
metricas(serie_previsao_AUTOARIMA_WEB.values.reshape(-1,1),
                        serie_teste_AUTOARIMA_WEB.values.reshape(-1,1))

# IMprimir méticas de previsão WEB AUTOARIMA
METRICAS = []
VALORES_AUTOARIMA_WEB = []
print('Metricas Modelo Baseline')
for metrica in metricas_AUTOARIMA_WEB:
    print(f'{metrica} = {metricas_AUTOARIMA_WEB[metrica]}')
    METRICAS.append(metrica)
    VALORES_AUTOARIMA_WEB.append(metricas_AUTOARIMA_WEB[metrica])
```

```
Metricas Modelo Baseline
ME = -4533.258909807301
MSE = 21785964.031487998
RMSE = 4667.5436828687525
MAE = 4533.258909807301
MPE = -0.10727448970930635
MAPE = 0.10727448970930635
MIN-MAX = 0.10727448970930631
```

Salvar métricas de previsão (WEB AUTOARIMA)

```
dfMetricas =
pd.read_csv('/content/gdrive/MyDrive/DADOS/AD/Métricas_Previsão.csv',
index_col=0)
dfMetricas = dfMetricas.assign(WEB_AUTOARIMA=VALORES_AUTOARIMA_WEB)
print(dfMetricas)
```

```
dfMetricas.to_csv('/content/gdrive/MyDrive/DADOS/AD/Métricas_Previsão.csv')
```

```
         ST_Baseline  WEB_Baseline   BD_Baseline  ST_AUTOARIMA
BD_AUTOARIMA  \
ME         -0.175799 -7.450912e+01 -3.879495e+02     49.857660 -
1.347929e+03
MSE       172.614151  4.322896e+06  6.146474e+06   7831.582677
5.482356e+06
RMSE       13.138270  2.079158e+03  2.479208e+03     88.496230
2.341443e+03
MAE        10.363013  1.653027e+03  1.862858e+03     71.980169
1.694711e+03
MPE         0.004776 -5.410408e-04  5.406480e-03      0.014344 -
1.300227e-01
MAPE        0.086862  3.921937e-02  2.499672e-01      0.020229
2.059837e-01
MIN-MAX     0.081098  3.780842e-02  2.116844e-01      0.019672
1.985067e-01

         WEB_AUTOARIMA           ST_GB        WEB_GB          BD_GB
ST_RN  \
ME       -4.533259e+03    -255.863571  4.217371e+02 -4.634744e+03  -
0.994816
MSE       2.178596e+07  367645.818074  1.059666e+06  2.692134e+07
58.959614
RMSE      4.667544e+03     606.338039  1.029401e+03  5.188578e+03
7.678516
MAE       4.533259e+03     466.513514  8.574755e+02  4.634744e+03
5.775603
MPE      -1.072745e-01      -0.058831  1.091842e-02 -3.790256e-01  -
0.005137
MAPE      1.072745e-01       0.138581  2.098227e-02  3.790256e-01
0.048112
MIN-MAX   1.072745e-01       0.130537  2.046146e-02  3.790256e-01
0.046630

               WEB_RN         BD_RN      ST_RNN
ME      -4.216761e+02 -6.021710e+02  -10.972016
MSE      1.004794e+07  2.444044e+07  261.003387
RMSE     3.169849e+03  4.943727e+03   16.155600
MAE      2.531802e+03  4.277532e+03   13.120908
MPE     -6.723076e-03  1.341376e-01   -0.079707
MAPE     5.931779e-02  6.762009e-01    0.103218
MIN-MAX  5.765807e-02  4.538536e-01    0.101939
```

Formativa: Modelo - Série BD AUTOARIMA

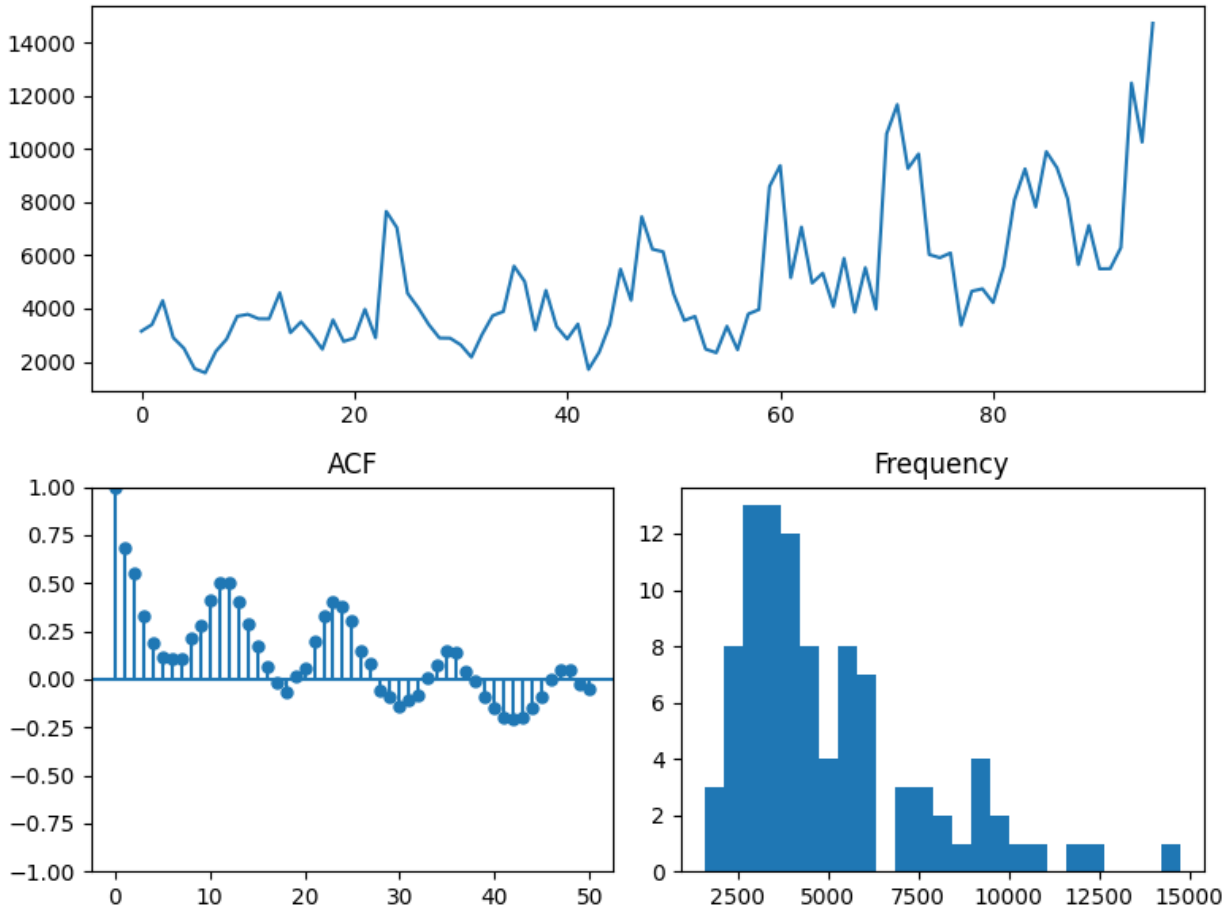
Criar série

- Selecionar coluna BD e transformar em medições mensais
- Exluir valores não existentes (anteriores a janeiro de 2010)

```
# Seleciona coluna BD e reformata para requisições mensais (soma)
serie_requisicoes_BD_mes =
serie_requisicoes_dia['BD'].resample('M').sum()

# Selecionar dados a partir de Janeiro de 2010
serie_BD = pd.Series(serie_requisicoes_BD_mes.loc['2010-01':'2017-
12'], copy=True)
```

Plotar série, ACF e frequência

```
# Plotar série, ACF e frequência
tsdisplay(serie_BD)
```



Criar o modelo

```
##########################################
##########  Criar o modelo ################
##########################################
####    Colocar seu código aqui  ###########
modelo_AUTOARIMA_BD = auto_arima(serie_BD,
                    start_p=0,
                    start_q=0,
                    d=0,
                    max_p=6,
                    max_q=6,
                    max_d=2,
                    start_P=1,
                    start_Q=1,
                    D=1,
                    max_P=2, max_D=1, max_Q=2, max_order=5,
                    m=12,
                    seasonal=True,
                    trace=False,
                    error_action='ignore',suppress_warnings=True,
                    stepwise=True)
```

correto

Avaliar desempenho

Criar séries de treinamento teste e previsão

- Usar 2/3 da série original para construir o modelo (serie_treino_AUTOARIMA_BD)
- Usar 1/3 da série original para comparar com previsões (serie_teste_AUTOARIMA_BD)
- Criar criar série de previsões usando o modelo (serie_previsao_AUTOARIMA_BD)

```
#################################################################
####     Criar séries de treinamento, teste e previsão   #######
#################################################################

#################################################################
###### Definir tamanho das séries de treinamento e teste #######
#################################################################
########  Colocar seu código aqui ###############################
lin2, = serie_BD.shape
tam_treino2 = int(np.ceil(lin2*2/3))
tam_teste2 = int(lin2 - np.ceil(lin2*2/3))
#################################################################
#########    Criar séries de treinamento e teste   #############
#################################################################
########  Colocar seu código aqui ###############################
serie_treino_AUTOARIMA_BD = serie_BD[:tam_treino2]
serie_teste_AUTOARIMA_BD = serie_BD[-tam_teste2:]
#################################################################
############     Criar séries de previsão     ##################
```

correto

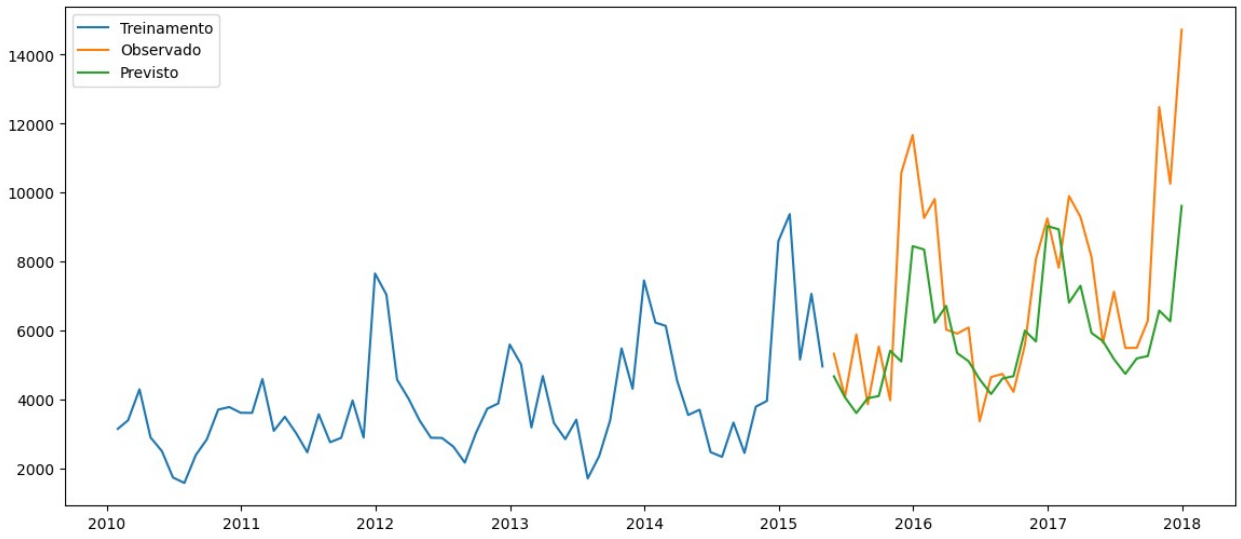
```
#################################################################
########  Colocar seu código aqui ################################
modelo_AUTOARIMA_BD.fit(serie_treino_AUTOARIMA_BD)
previsao_AUTOARIMA_BD =
modelo_AUTOARIMA_BD.predict(n_periods=tam_teste2)
```

ok

Plotar gráfico de previsões

```
#################################################################
######## Plotar treinamento previsto e observado ############
#################################################################
########  Colocar seu código aqui ###########################
fig, ax = plt.subplots()
ax.plot(serie_treino_AUTOARIMA_BD, label='Treinamento')
ax.plot(serie_teste_AUTOARIMA_BD, label='Observado')
ax.plot(previsao_AUTOARIMA_BD, label='Previsto')
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.legend(loc='upper left', fontsize=10);
```

correto



Calcular e imprimir métricas de previsão (BD AUTOARIMA)

```
#################################################################
########   Calcular métricas de previsão BD AUTOARIMA  #########
#################################################################
########       Colocar seu código aqui   ########################
metricas_AUTOARIMA_BD =
metricas(previsao_AUTOARIMA_BD.values.reshape(-1,1),
                    serie_teste_AUTOARIMA_BD.values.reshape(-1,1))

#################################################################
########   Imprimir métricas de previsão BD AUTOARIMA  #########
#################################################################
```

correto

```
########       Colocar seu código aqui   ######################
METRICAS2 = []
VALORES_AUTOARIMA_BD = []
print('Metricas Modelo Baseline')
for metrica in metricas_AUTOARIMA_BD:
  print(f'{metrica} = {metricas_AUTOARIMA_BD[metrica]}')
  METRICAS.append(metrica)
  VALORES_AUTOARIMA_BD.append(metricas_AUTOARIMA_BD[metrica])
```

```
Metricas Modelo Baseline
ME = -1347.9287864475903
MSE = 5482355.598968385
RMSE = 2341.443059091633
MAE = 1694.711065919138
MPE = -0.13002270543763972
MAPE = 0.20598367511987284
MIN-MAX = 0.19850668013132022
```

Salvar méticas de previsão (BD AUTOARIMA)

```
# Salver métricas de previsão
# Código completo, não é necessário alterar
dfMetricas =
pd.read_csv('/content/gdrive/MyDrive/DADOS/AD/Métricas_Previsão.csv',
index_col=0)
dfMetricas = dfMetricas.assign(BD_AUTOARIMA=VALORES_AUTOARIMA_BD)
print(dfMetricas)
dfMetricas.to_csv('/content/gdrive/MyDrive/DADOS/AD/Métricas_Previsão.
csv')
```

```
         ST_Baseline  WEB_Baseline   BD_Baseline  ST_AUTOARIMA
BD_AUTOARIMA  \
ME         -0.175799 -7.450912e+01 -3.879495e+02     49.857660 -
1.347929e+03
MSE       172.614151  4.322896e+06  6.146474e+06   7831.582677
5.482356e+06
RMSE       13.138270  2.079158e+03  2.479208e+03     88.496230
2.341443e+03
MAE        10.363013  1.653027e+03  1.862858e+03     71.980169
1.694711e+03
MPE         0.004776 -5.410408e-04  5.406480e-03      0.014344 -
1.300227e-01
MAPE        0.086862  3.921937e-02  2.499672e-01      0.020229
2.059837e-01
MIN-MAX     0.081098  3.780842e-02  2.116844e-01      0.019672
1.985067e-01

         WEB_AUTOARIMA          ST_GB        WEB_GB          BD_GB
ST_RN  \
ME       -4.533259e+03    -255.863571  4.217371e+02 -4.634744e+03  -
```

```
0.994816
MSE       2.178596e+07  367645.818074  1.059666e+06  2.692134e+07
58.959614
RMSE      4.667544e+03     606.338039  1.029401e+03  5.188578e+03
7.678516
MAE       4.533259e+03     466.513514  8.574755e+02  4.634744e+03
5.775603
MPE      -1.072745e-01      -0.058831  1.091842e-02 -3.790256e-01  -
0.005137
MAPE      1.072745e-01       0.138581  2.098227e-02  3.790256e-01
0.048112
MIN-MAX   1.072745e-01       0.130537  2.046146e-02  3.790256e-01
0.046630

                WEB_RN         BD_RN      ST_RNN
ME      -4.216761e+02 -6.021710e+02  -10.972016
MSE      1.004794e+07  2.444044e+07  261.003387
RMSE     3.169849e+03  4.943727e+03   16.155600
MAE      2.531802e+03  4.277532e+03   13.120908
MPE     -6.723076e-03  1.341376e-01   -0.079707
MAPE     5.931779e-02  6.762009e-01    0.103218
MIN-MAX  5.765807e-02  4.538536e-01    0.101939
```

Comparar métricas

Comparar as métricas da série WEB e série BD

```python
# Comparar métricas
# Código completo, não é necessário altera
print('Metricas Série WEB')
for metrica in metricas_AUTOARIMA_WEB:
    print(f'{metrica} = {metricas_AUTOARIMA_WEB[metrica]}')

print('\nMetricas Série BD')
for metrica in metricas_AUTOARIMA_BD:
    print(f'{metrica} = {metricas_AUTOARIMA_BD[metrica]}')
```

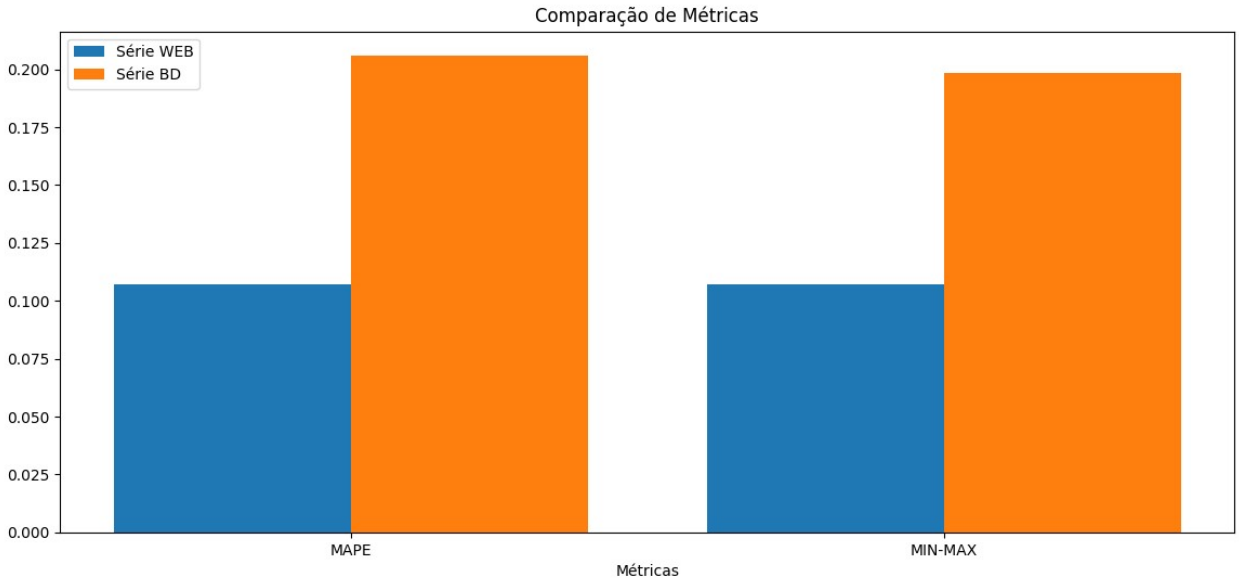
```
Metricas Série WEB
ME = -4533.258909807301
MSE = 21785964.031487998
RMSE = 4667.5436828687525
MAE = 4533.258909807301
MPE = -0.10727448970930635
MAPE = 0.10727448970930635
MIN-MAX = 0.10727448970930631

Metricas Série BD
ME = -1347.9287864475903
MSE = 5482355.598968385
RMSE = 2341.443059091633
```

```
MAE = 1694.711065919138
MPE = -0.13002270543763972
MAPE = 0.20598367511987284
MIN-MAX = 0.19850668013132022
```

Plotar gráfico de comparação AUTOARIMA WEB e AUTOARIMA BD

- Métricas: MAPE MIN-MAX

```
# Plotar gráfico de comparação AUTOARIMA WEB e AUTOARIMA BD
# Código ccompleto, não é necessário alterar
X = ['MAPE', 'MIN-MAX']
mWEB = []
mBD = []
for x in X:
    mWEB.append(metricas_AUTOARIMA_WEB[x])
    mBD.append(metricas_AUTOARIMA_BD[x])

eixo_X = np.arange(len(X))

plt.bar(eixo_X - 0.2, mWEB, 0.4, label = 'Série WEB')
plt.bar(eixo_X + 0.2, mBD, 0.4, label = 'Série BD')

plt.xticks(eixo_X, X)
plt.xlabel('Métricas')
plt.title('Comparação de Métricas')
plt.legend()
plt.show;
```

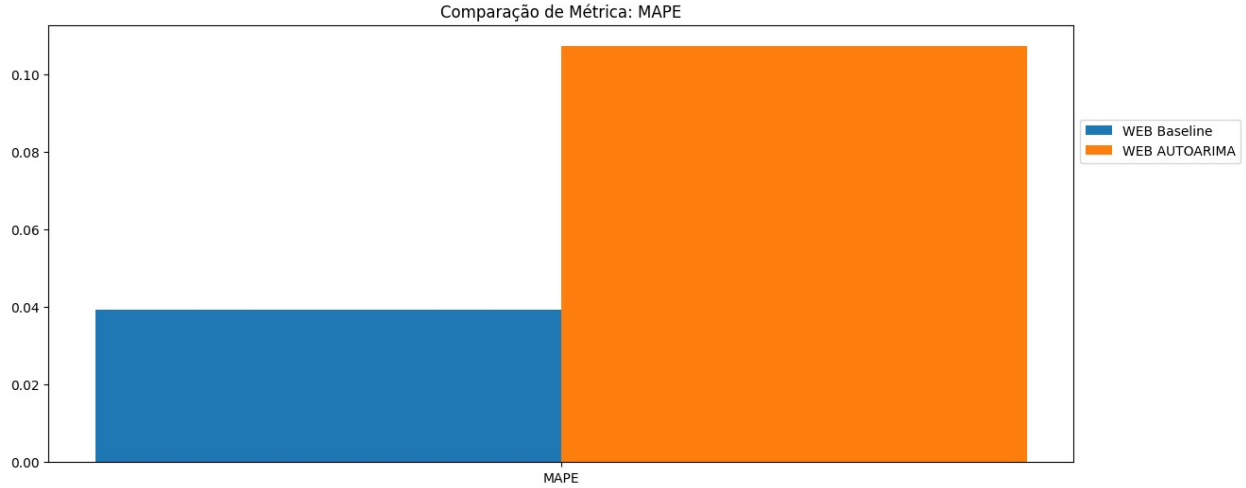


Comparar desempenho de modelos

Comparar desempenho dos modelos WEB_Baseline e WEB_AUTOARIMA

- Métrica MAPE

```
# Comparar desempenho dos modelos WEB_Baseline e WEB_AUTOARIMA
# Código completo, não é necessário alterar
X = ['MAPE']
mWEB_Baseline = []
mWEB_AUTOARIMA = []
for x in X:
    mWEB_Baseline.append(dfMetricas['WEB_Baseline'].loc[x])
    mWEB_AUTOARIMA.append(dfMetricas['WEB_AUTOARIMA'].loc[x])

eixo_X = np.arange(len(X))

plt.bar(eixo_X - 0.2, mWEB_Baseline, 0.4, label = 'WEB Baseline')
plt.bar(eixo_X + 0.2, mWEB_AUTOARIMA, 0.4, label = 'WEB AUTOARIMA')

plt.xticks(eixo_X, X)
plt.title('Comparação de Métrica: MAPE')
plt.legend(bbox_to_anchor=(1, 0.8))
plt.show;
```



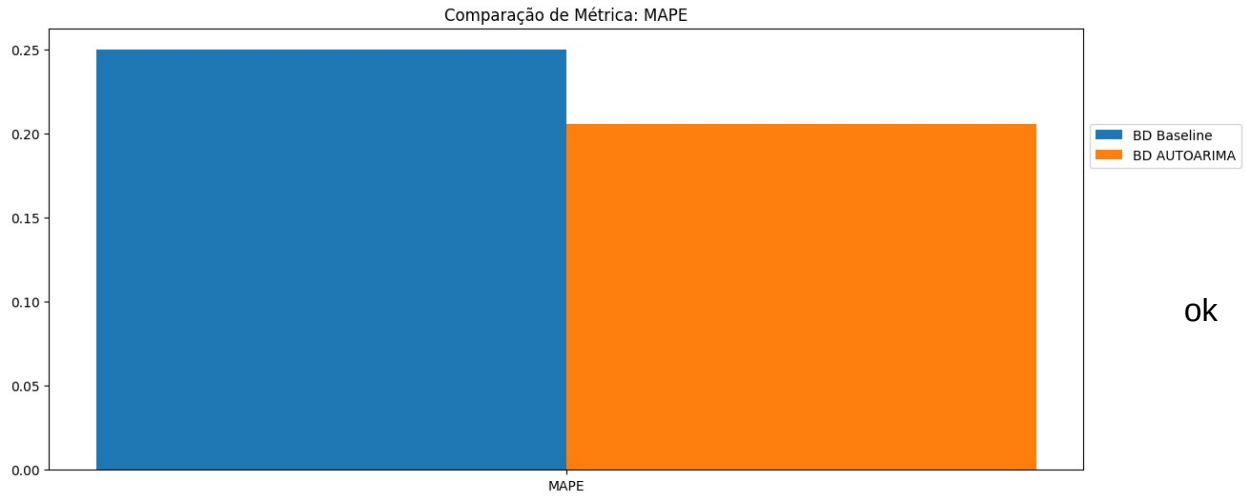
Comparar desempenho dos modelos BD_Baseline e BD_AUTOARIMA

- Métrica MAPE

```
#####################################################################
#### Comparar desempenho dos modelos BD_Baseline e BD_AUTOARIMA #####
#####################################################################
########## Clocar seu código aqui ###################################
X = ['MAPE']
```

```
mBD_Baseline = []
mBD_AUTOARIMA = []
for x in X:
    mBD_Baseline.append(dfMetricas['BD_Baseline'].loc[x])
    mBD_AUTOARIMA.append(dfMetricas['BD_AUTOARIMA'].loc[x])

eixo_X = np.arange(len(X))

plt.bar(eixo_X - 0.2, mBD_Baseline, 0.4, label = 'BD Baseline')
plt.bar(eixo_X + 0.2, mBD_AUTOARIMA, 0.4, label = 'BD AUTOARIMA')

plt.xticks(eixo_X, X)
plt.title('Comparação de Métrica: MAPE')
plt.legend(bbox_to_anchor=(1, 0.8))
plt.show;
```

correto



ok

Entrega

- Completar os códigos necessários
- Imprimir para pdf
- Fazer upload no AVA