Aprendizado Dinâmico - Práticas

por **Cibele Russo**

ICMC/USP - São Carlos SP

Tutores: Caio Moura Quina, Carlos Miguel Toste Sisto, Leonardo Piccaro Rezende, Luna Wagner Cunha, Maria Pinheiro Garcia Blanco, Tobias Mesquita Silva da Veiga.

Setembro de 2020.

MBA em Ciências de Dados.

1 Prática 1

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Considere os dados de COVID-19 para a cidade de São Paulo. Nesta prática, aplicaremos os conhecimentos adquiridos na Aula 1, fazendo:

1. Visualização de dados completos e parciais
2. Construiremos gráficos da média móvel simples e exponencialmente ponderada, para casos e para mortes separadamente.
3. Faremos a decomposição em tendência e sazonalidade

1.1 Exercício 1:

1. Leia os dados de COVID-19 da base covid_caso.csv.
2. Considere os dados de casos e mortes diárias (diferenças). Salve os dados da cidade de São Paulo num arquivo covidSaoPaulo.csv.

3. Produza um gráfico da séries de casos e mortes diários da cidade de São Paulo desde o primeiro caso.
4. Em seguida, considere o gráfico do log de casos a partir somente do centésimo caso.

[1]:
```
# Bibliotecas

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

1. Faça a leitura dos dados a partir do arquivo covid_caso.csv disponível no material do curso.

[2]:
```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

# COVID - Leitura dos dados
covid = pd.read_csv(f'{pkgdir}/covid_caso.csv', index_col='date',␣
 ↪parse_dates=True)
covid.head()
```

[2]:
```
            state city place_type  confirmed  deaths  order_for_place  is_last  \
date
2020-09-02     AP  NaN      state      43800     668              166     True
2020-09-01     AP  NaN      state      43514     663              165    False
2020-08-31     AP  NaN      state      43193     661              164    False
2020-08-30     AP  NaN      state      42892     660              163    False
2020-08-29     AP  NaN      state      42771     659              162    False

            estimated_population_2019  city_ibge_code  \
date
2020-09-02                   845731.0            16.0
2020-09-01                   845731.0            16.0
2020-08-31                   845731.0            16.0
2020-08-30                   845731.0            16.0
2020-08-29                   845731.0            16.0

            confirmed_per_100k_inhabitants  death_rate
date
2020-09-02                      5178.95170      0.0153
2020-09-01                      5145.13480      0.0152
2020-08-31                      5107.17947      0.0153
2020-08-30                      5071.58896      0.0154
2020-08-29                      5057.28181      0.0154
```

Considere apenas os dados de casos e óbitos diários da cidade de São Paulo. Tome as diferenças para obter dados diários.

[3]:
```
covid = pd.read_csv(f'{pkgdir}/covid_caso.csv', index_col=0, parse_dates=True)

covid = covid.loc[(covid['city']=='São␣
 ↪Paulo')&(covid['place_type']=='city'),['confirmed','deaths']]

covid = covid.sort_values(by=['date'])

covid.head()
```

[3]:

date	confirmed	deaths
2020-02-25	1	0
2020-02-26	1	0
2020-02-27	1	0
2020-02-28	2	0
2020-02-29	2	0

[4]:
```
y1 = list(np.diff(covid['confirmed']))
y2 = list(np.diff(covid['deaths']))
x = covid.index[1:] # Note o subconjunto de dados em x iniciando em 1 pois␣
 ↪interessa as diferenças

covid = pd.DataFrame({'date':x,'confirmed':y1,'deaths':y2})

covid = covid.set_index('date') # Estabelece o índice

covid.head()
```

[4]:

date	confirmed	deaths
2020-02-26	0	0
2020-02-27	0	0
2020-02-28	1	0
2020-02-29	0	0
2020-03-01	0	0

Salve os dados de São Paulo num arquivo covidSaoPaulo.csv.

[5]:
```
covid.to_csv('covidSaoPaulo.csv')
```

Produza um gráfico da séries de casos e mortes diários da cidade de São Paulo desde o primeiro caso.

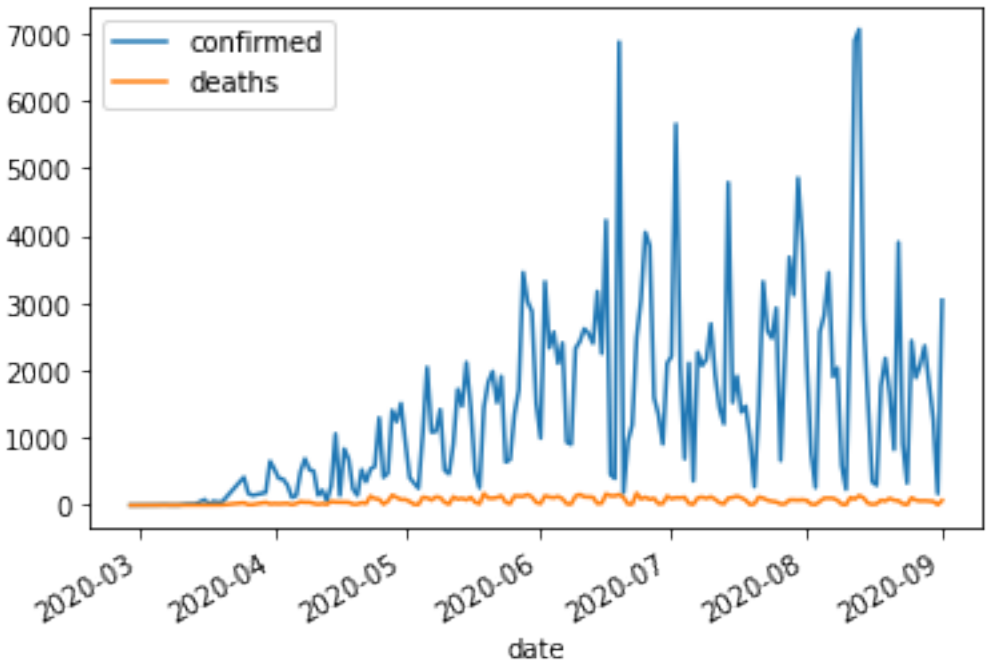
[6]:
```
covid[covid['confirmed']>0].head(3)
```

[6]:

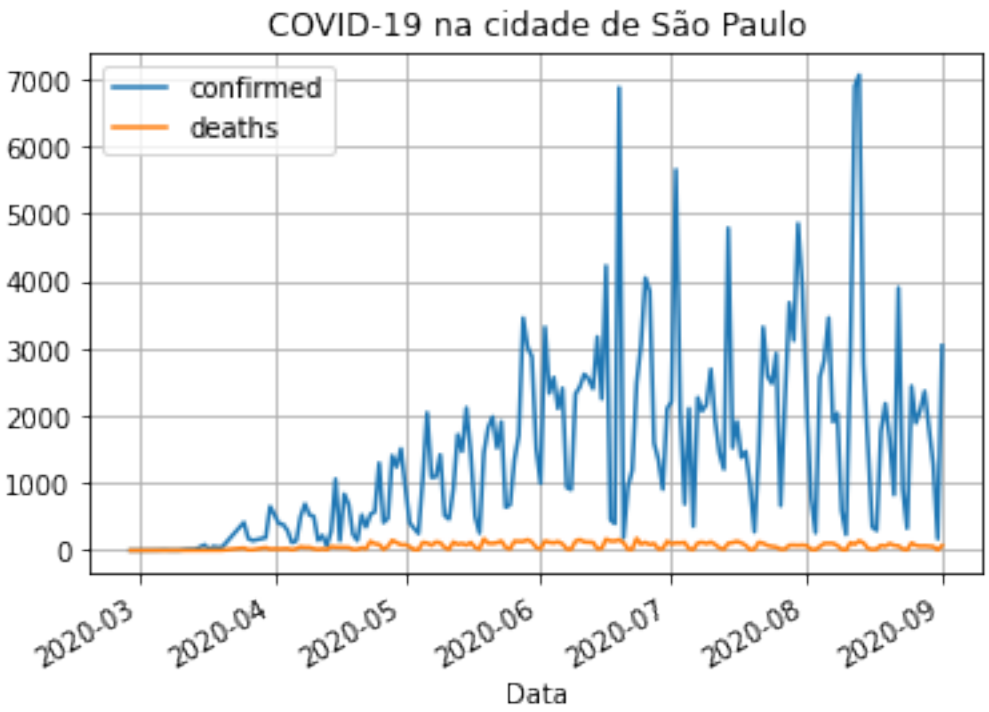
date	confirmed	deaths
2020-02-28	1	0

```
2020-03-04          1       0
2020-03-05          3       0
```

[7]:
```
covid[covid['confirmed']>0].plot();
```

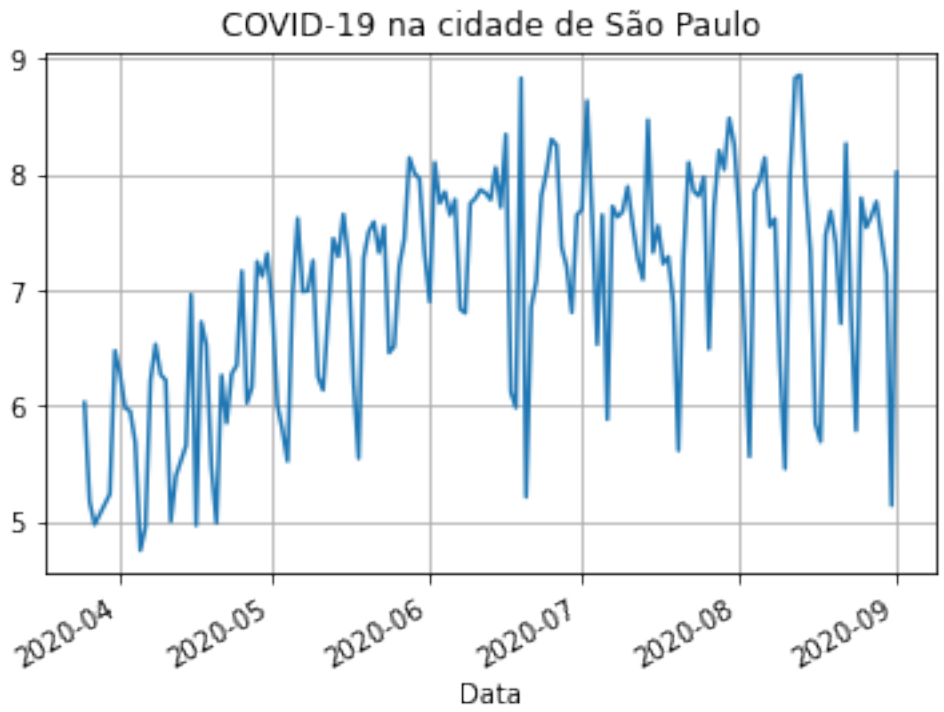


[8]:
```
title = 'COVID-19 na cidade de São Paulo'
ylabel = ''
xlabel = 'Data'

ax=covid[covid['confirmed']>0].plot(title=title);
ax.autoscale(axis='both');
ax.set(xlabel=xlabel,ylabel=ylabel);

ax.xaxis.grid(True) # Com grades
ax.yaxis.grid(True)
```



4. Considere o gráfico do log de casos a partir somente do centésimo caso.

[9]:
```
title = 'COVID-19 na cidade de São Paulo'
ylabel = ''
xlabel = 'Data'

ax=np.log(covid[covid['confirmed']>100]['confirmed']).plot(title=title);
ax.autoscale(axis='both');
ax.set(xlabel=xlabel,ylabel=ylabel);

ax.xaxis.grid(True) # Com grades
ax.yaxis.grid(True)
```



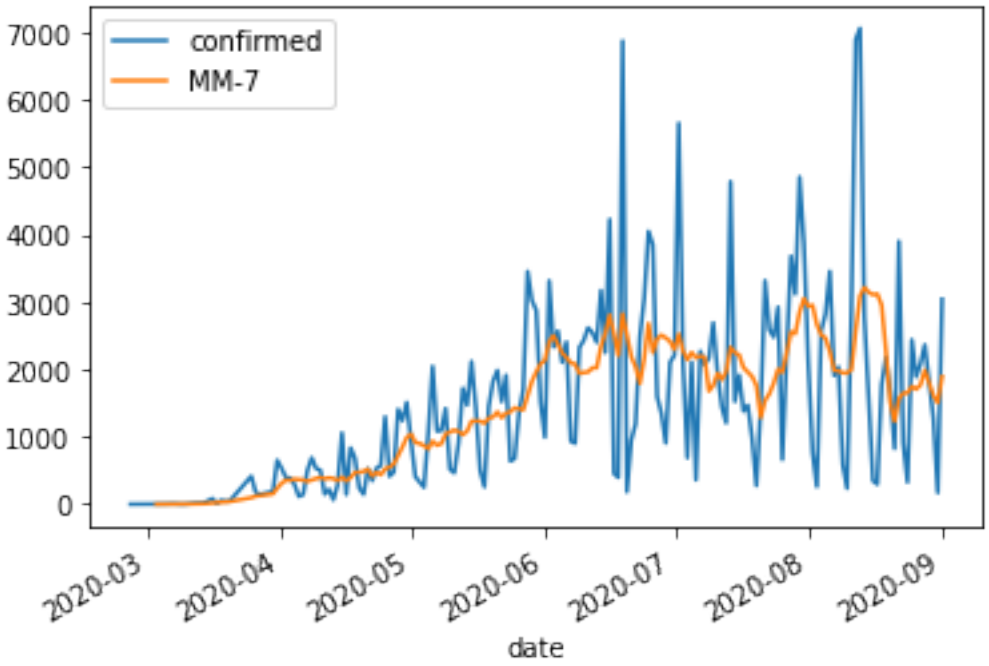
1.2 Exercício 2

1. Construa gráficos da média móvel simples e exponencialmente ponderada para casos. Utilize janela de 7 dias para a MMS e span=7 para MMEP.
2. Construa gráficos da média móvel simples e exponencialmente ponderada para mortes. Utilize janela de 7 dias para a MMS e span=7 para MMEP.

Construa gráficos da média móvel simples e exponencialmente ponderada para a variável casos.

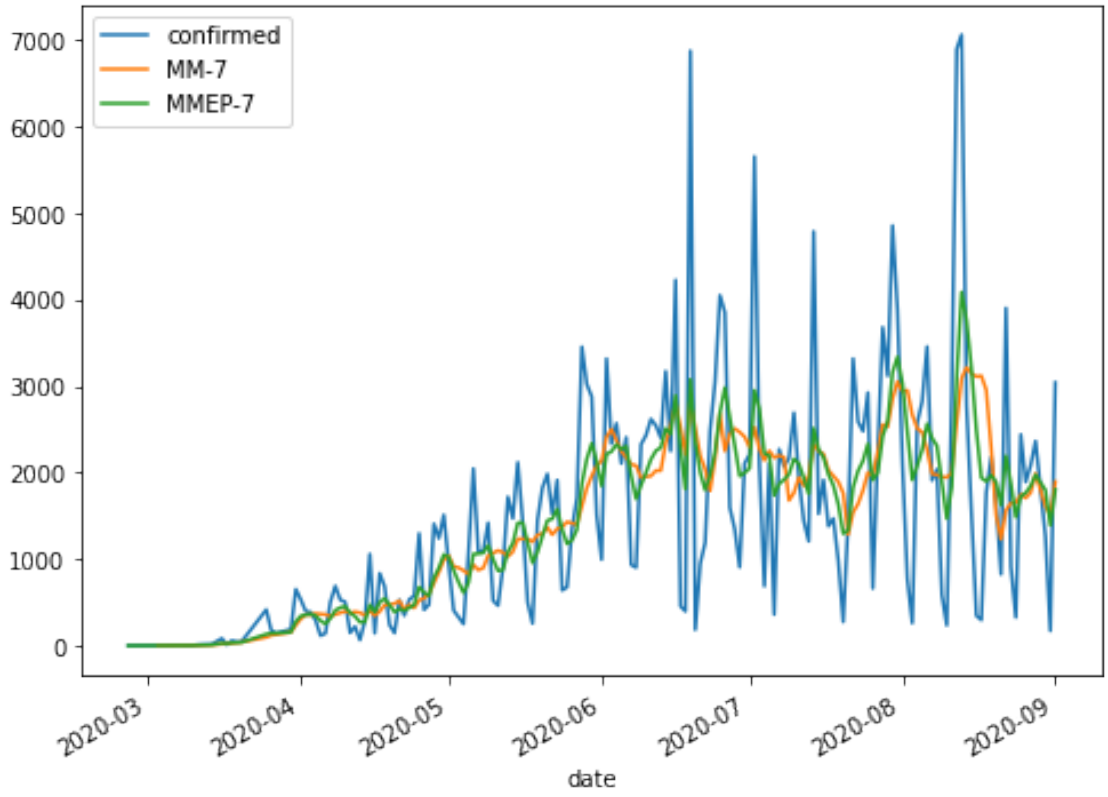
[10]:
```
covid['MM-7'] = covid['confirmed'].rolling(window=7).mean()
```

[11]:
```
covid[['confirmed','MM-7']].plot();
```



[12]:
```
covid['MMEP-7'] = covid['confirmed'].ewm(span=7,adjust=False).mean()
```

[13]:
```
covid[['confirmed','MM-7','MMEP-7']].plot(figsize=(8,6));
```



Construa gráficos da média móvel simples e exponencialmente ponderada para a variável mortes (deaths).

1.3 Exercício 3

Faça a decomposição em tendência e sazonalidade do número de casos utilizando a função seasonal_decompose do módulo statsmodels.

[14]:
```
covidSP = pd.read_csv('covidSaoPaulo.csv', index_col='date')

covidSP.index = pd.DatetimeIndex(covidSP.index)

covidSP.head()
```

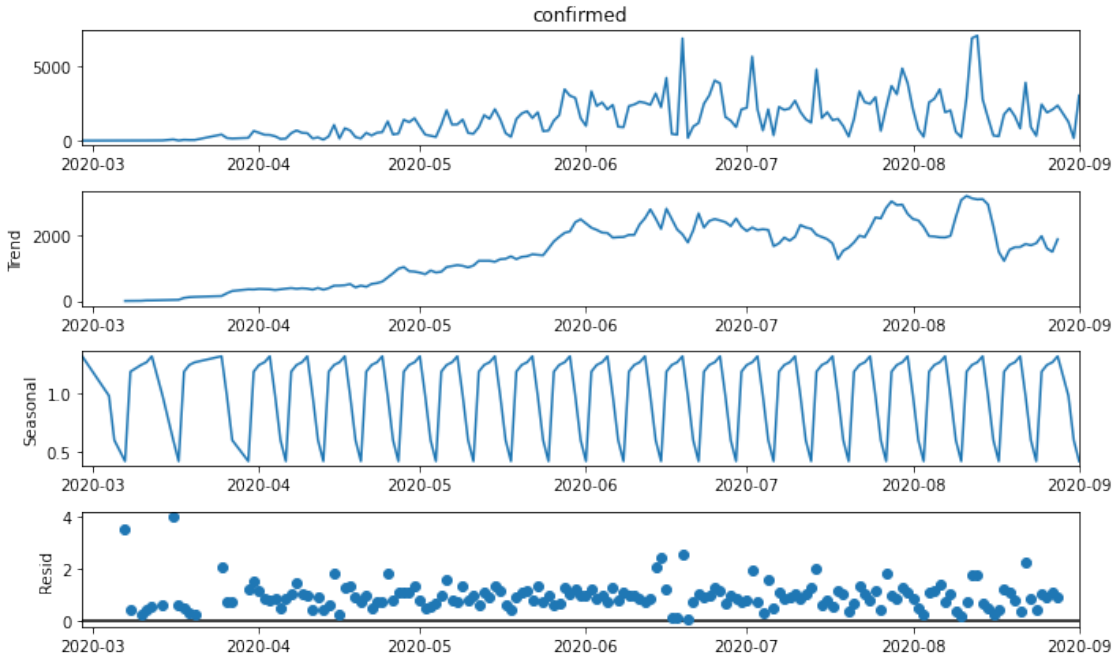
[14]:

date	confirmed	deaths
2020-02-26	0	0
2020-02-27	0	0
2020-02-28	1	0
2020-02-29	0	0
2020-03-01	0	0

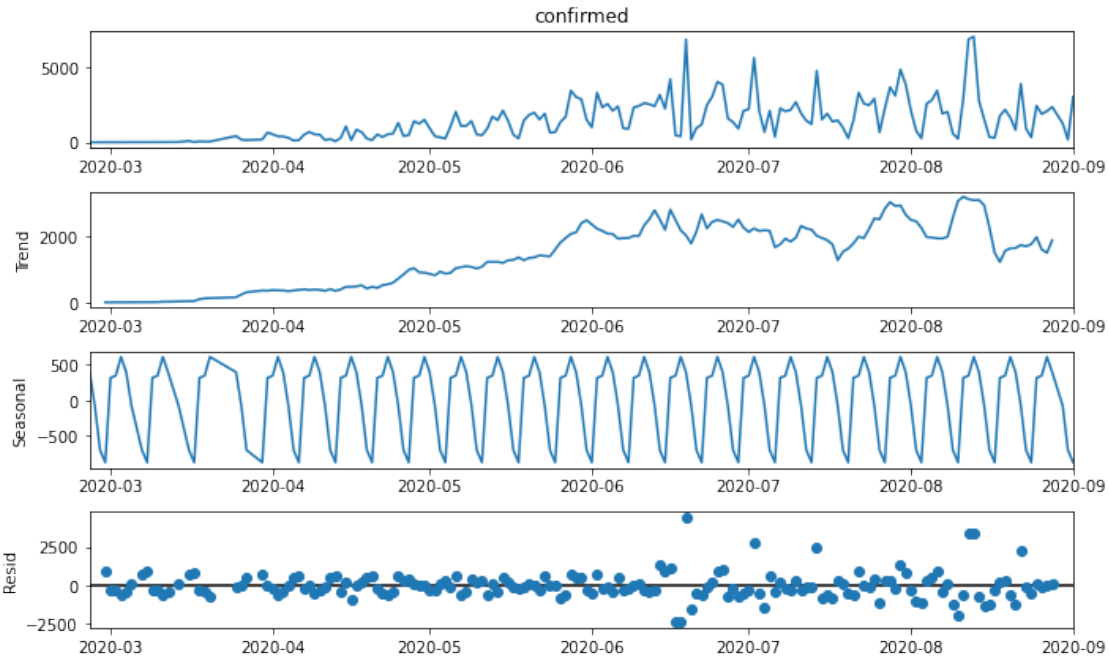
[15]:
```
covidSP.index
```

[15]:
```
DatetimeIndex(['2020-02-26', '2020-02-27', '2020-02-28', '2020-02-29',
               '2020-03-01', '2020-03-02', '2020-03-03', '2020-03-04',
               '2020-03-05', '2020-03-07',
               ...
               '2020-08-22', '2020-08-23', '2020-08-24', '2020-08-25',
               '2020-08-26', '2020-08-27', '2020-08-28', '2020-08-30',
               '2020-08-31', '2020-09-01'],
              dtype='datetime64[ns]', name='date', length=179, freq=None)
```

[16]:
```
import pandas as pd
from statsmodels.tsa.seasonal import seasonal_decompose
from pylab import rcParams

result = seasonal_decompose(covidSP[covidSP['confirmed']>0]['confirmed'],␣
 ↪model='multiplicative', period=7)

rcParams['figure.figsize'] = 10, 6

fig = result.plot()

plt.show()
```



[17]:
```
result = seasonal_decompose(covidSP['confirmed'], model='additive', period=7)

rcParams['figure.figsize'] = 10,6

fig = result.plot()

plt.show()
```



Na sua opinião, qual o tipo de sazonalidade mais adequada, aditiva ou multiplicativa? Justifique com base nos resíduos.

R: Embora haja pontos atípicos nos gráficos de resíduos de ambos os casos, a sazonalidade multiplicativa parece mais adequada a esses dados.

2 Prática 2

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Considere os dados de fechamento do papel PETR4 a partir de 18/3 no arquivo PETR4.SA.csv. (fonte dos dados completos, nesta prática utilizaremos uma parte deles: https://br.financas.yahoo.com/quote/PETR4.SA/history/)

Nesta prática, aplicaremos os conhecimentos adquiridos na Aula 2 a esses dados, a saber:

- Método de Holt
- Método de Holt-Winters
- Previsão
- Estacionariedade

Carregue as bibliotecas

[18]:
```
# Bibliotecas

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

1. Faça a leitura dos dados a partir do arquivo PETR4.SA.csv disponível no material do curso. Para essa análise, utilizaremos a variável que indica os fechamentos, Close. Quantas observações tem a série original? Visualize a série Close.

[19]:
```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

# PETR4 - Leitura dos dados
PETR4 = pd.read_csv(f'{pkgdir}/PETR4.csv', index_col='Date', parse_dates=True)

PETR4.head()
```

[19]:

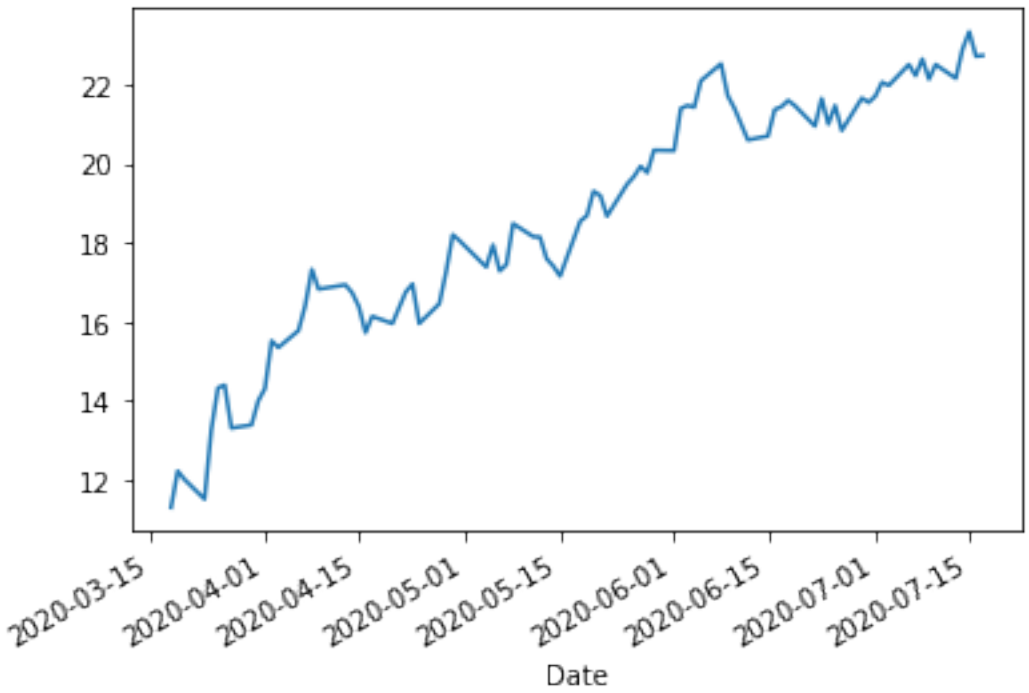
Date	Open	High	Low	Close	Adj Close	Volume
2020-03-18	11.79	12.27	10.87	11.29	11.289389	162515200
2020-03-19	11.07	13.07	10.85	12.21	12.209339	157065200
2020-03-20	13.11	13.50	11.83	12.00	11.999351	149029900
2020-03-23	12.11	12.18	11.28	11.50	11.499378	107092400
2020-03-24	12.57	13.54	12.28	13.25	13.249283	114259200

[20]:
```
len(PETR4)
```

[20]:
```
84
```

[21]:
```
PETR4['Close'].plot()
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb34ac56850>
```



2. Quais as datas mínima e máxima de observação da série Close? Repita o procedimento feito em aula para preencher os dados faltantes com a função PETR4.fillna(method='ffill', inplace=True). Nesse caso, a frequência usada em pd.date_range deve ser 'B' pois a bolsa de valores não funciona aos fins de semana. O que faz o método ffill de fillna?

[22]:
```
PETR4.index.min()
```

[22]:
```
Timestamp('2020-03-18 00:00:00')
```

[23]:
```
PETR4.index.max()
```

[23]:
```
Timestamp('2020-07-17 00:00:00')
```

[24]:
```
idx = pd.date_range(start=PETR4.index.min(), end=PETR4.index.max(), freq='B')
idx
```

[24]:
```
DatetimeIndex(['2020-03-18', '2020-03-19', '2020-03-20', '2020-03-23',
               '2020-03-24', '2020-03-25', '2020-03-26', '2020-03-27',
               '2020-03-30', '2020-03-31', '2020-04-01', '2020-04-02',
               '2020-04-03', '2020-04-06', '2020-04-07', '2020-04-08',
               '2020-04-09', '2020-04-10', '2020-04-13', '2020-04-14',
               '2020-04-15', '2020-04-16', '2020-04-17', '2020-04-20',
               '2020-04-21', '2020-04-22', '2020-04-23', '2020-04-24',
               '2020-04-27', '2020-04-28', '2020-04-29', '2020-04-30',
```

```
               '2020-05-01', '2020-05-04', '2020-05-05', '2020-05-06',
               '2020-05-07', '2020-05-08', '2020-05-11', '2020-05-12',
               '2020-05-13', '2020-05-14', '2020-05-15', '2020-05-18',
               '2020-05-19', '2020-05-20', '2020-05-21', '2020-05-22',
               '2020-05-25', '2020-05-26', '2020-05-27', '2020-05-28',
               '2020-05-29', '2020-06-01', '2020-06-02', '2020-06-03',
               '2020-06-04', '2020-06-05', '2020-06-08', '2020-06-09',
               '2020-06-10', '2020-06-11', '2020-06-12', '2020-06-15',
               '2020-06-16', '2020-06-17', '2020-06-18', '2020-06-19',
               '2020-06-22', '2020-06-23', '2020-06-24', '2020-06-25',
               '2020-06-26', '2020-06-29', '2020-06-30', '2020-07-01',
               '2020-07-02', '2020-07-03', '2020-07-06', '2020-07-07',
               '2020-07-08', '2020-07-09', '2020-07-10', '2020-07-13',
               '2020-07-14', '2020-07-15', '2020-07-16', '2020-07-17'],
              dtype='datetime64[ns]', freq='B')
```

[25]:
```
PETR4 = PETR4.reindex(idx)
PETR4.head()
```

[25]:

	Open	High	Low	Close	Adj Close	Volume
2020-03-18	11.79	12.27	10.87	11.29	11.289389	162515200.0
2020-03-19	11.07	13.07	10.85	12.21	12.209339	157065200.0
2020-03-20	13.11	13.50	11.83	12.00	11.999351	149029900.0
2020-03-23	12.11	12.18	11.28	11.50	11.499378	107092400.0
2020-03-24	12.57	13.54	12.28	13.25	13.249283	114259200.0

[26]:
```
PETR4.fillna(method='ffill', inplace=True)
```

3. Qual o novo número de observações da série Close após o procedimento em 2.?

[27]:
```
len(PETR4)
```

[27]: 88

4. Vamos aplicar os métodos de Holt e Holt-Winters para esses dados. Faça a leitura das bibliotecas necessárias e crie a série Holt na base de dados. Atribua a ela os valores ajustados pelo Método de Holt. Lembre-se que a série fica deslocada e corrija com a função shift(-1).

[28]:
```
# Método de Holt

from statsmodels.tsa.api import ExponentialSmoothing

modelo =  ExponentialSmoothing(PETR4['Close'], trend='add');

ajustadoH = modelo.fit();

PETR4['Holt'] = ajustadoH.fittedvalues.shift(-1);
```

```
PETR4.head()
```

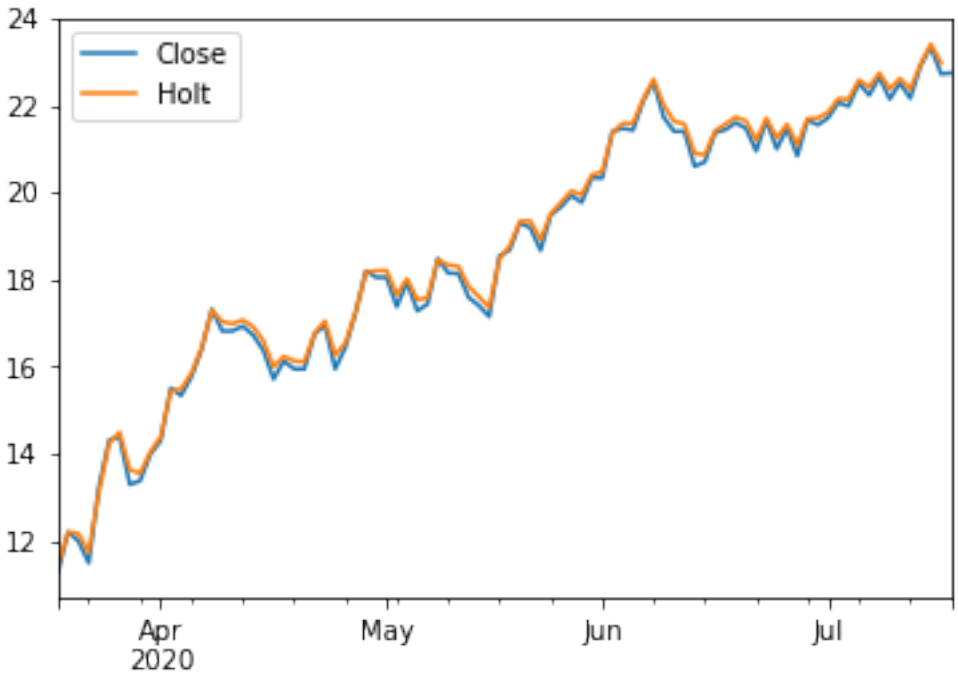
[28]:

	Open	High	Low	Close	Adj Close	Volume	Holt
2020-03-18	11.79	12.27	10.87	11.29	11.289389	162515200.0	11.441653
2020-03-19	11.07	13.07	10.85	12.21	12.209339	157065200.0	12.209526
2020-03-20	13.11	13.50	11.83	12.00	11.999351	149029900.0	12.166393
2020-03-23	12.11	12.18	11.28	11.50	11.499378	107092400.0	11.744354
2020-03-24	12.57	13.54	12.28	13.25	13.249283	114259200.0	13.123712

5. Visualize as séries Close e Holt. O método apresenta um bom ajuste?

[29]:
```
PETR4[['Close','Holt']].plot()
```

[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb330f260d0>



6. Faz sentido ajustar o método de Holt-Winters? Qual seria o período da sazonalidade? 7 dias pode fazer sentido? Aplique os métodos de Holt-Winters com sazonalidade aditiva e multiplicativa e visualize os resultados.

[30]:
```
# Ajuste do modelo pelo Método de Holt-Winters com sazonalidade aditiva

modelo = ␣
↪ExponentialSmoothing(PETR4['Close'],trend='add',seasonal='add',seasonal_periods=7);
↪
```

```
ajustadoHW = modelo.fit();

PETR4['Holt-Winters-adit-7'] = ajustadoHW.fittedvalues;

PETR4.head()
```

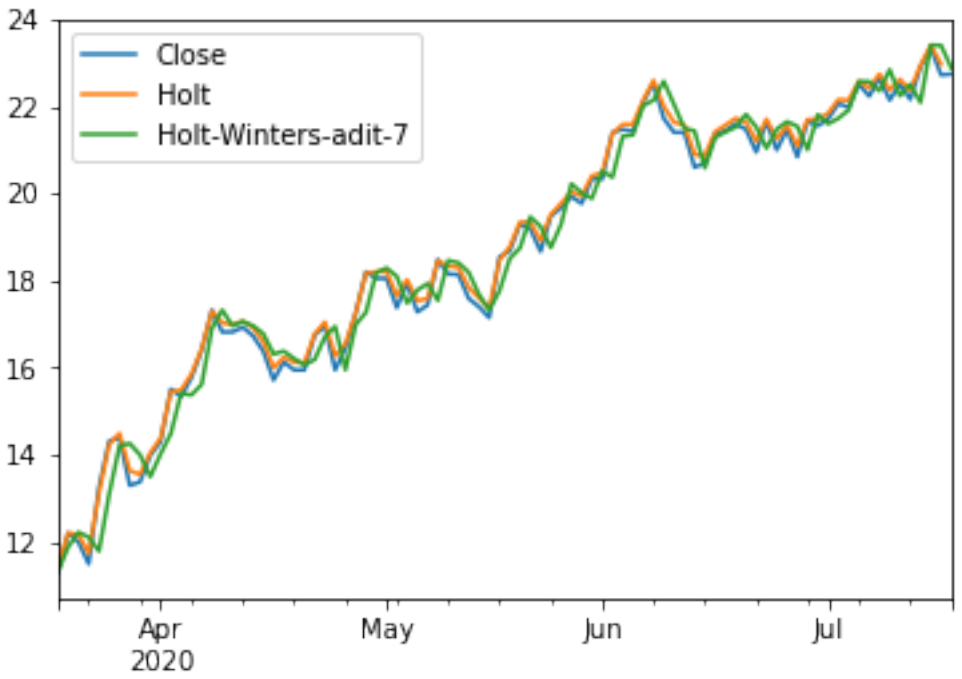
[30]:

	Open	High	Low	Close	Adj Close	Volume	Holt	\
2020-03-18	11.79	12.27	10.87	11.29	11.289389	162515200.0	11.441653	
2020-03-19	11.07	13.07	10.85	12.21	12.209339	157065200.0	12.209526	
2020-03-20	13.11	13.50	11.83	12.00	11.999351	149029900.0	12.166393	
2020-03-23	12.11	12.18	11.28	11.50	11.499378	107092400.0	11.744354	
2020-03-24	12.57	13.54	12.28	13.25	13.249283	114259200.0	13.123712	

	Holt-Winters-adit-7
2020-03-18	11.322575
2020-03-19	11.887505
2020-03-20	12.224830
2020-03-23	12.112010
2020-03-24	11.792121

[31]:
```
PETR4[['Close','Holt','Holt-Winters-adit-7']].plot()
```

[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb33023c750>



[32]:
```
# Ajuste do modelo pelo Método de Holt-Winters com sazonalidade aditiva

modelo = ␣
 ↪ExponentialSmoothing(PETR4['Close'],trend='add',seasonal='mul',seasonal_periods=7);
 ↪

ajustadoHW = modelo.fit();

PETR4['Holt-Winters-mul-7'] = ajustadoHW.fittedvalues;

PETR4.head()
```

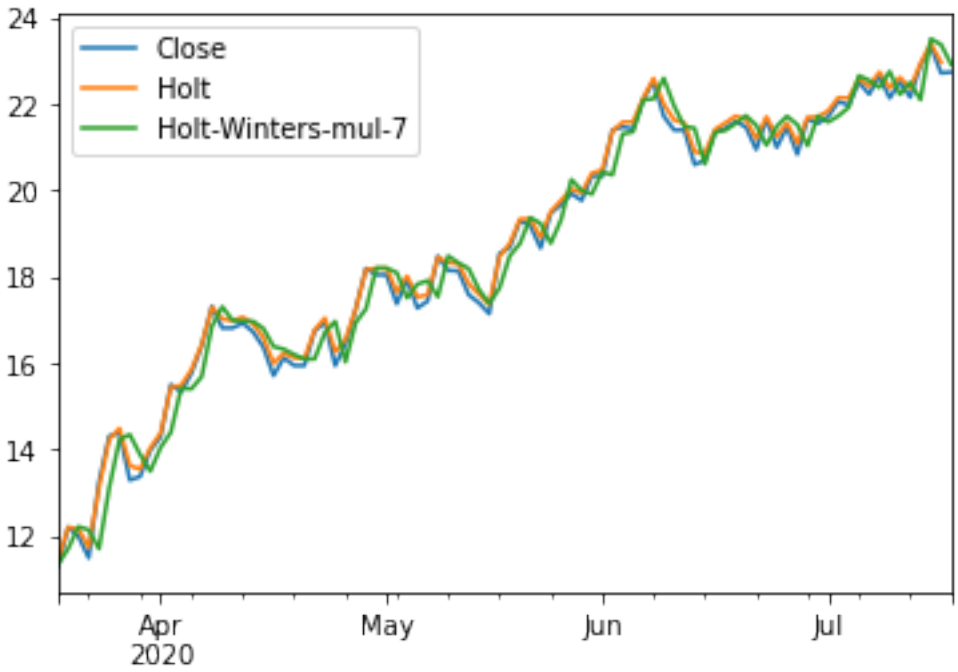
[32]:

	Open	High	Low	Close	Adj Close	Volume	Holt	\
2020-03-18	11.79	12.27	10.87	11.29	11.289389	162515200.0	11.441653	
2020-03-19	11.07	13.07	10.85	12.21	12.209339	157065200.0	12.209526	
2020-03-20	13.11	13.50	11.83	12.00	11.999351	149029900.0	12.166393	
2020-03-23	12.11	12.18	11.28	11.50	11.499378	107092400.0	11.744354	
2020-03-24	12.57	13.54	12.28	13.25	13.249283	114259200.0	13.123712	

	Holt-Winters-adit-7	Holt-Winters-mul-7
2020-03-18	11.322575	11.346998
2020-03-19	11.887505	11.711855
2020-03-20	12.224830	12.222012
2020-03-23	12.112010	12.148913
2020-03-24	11.792121	11.710787

[33]:
```
PETR4[['Close','Holt','Holt-Winters-mul-7']].plot()
```

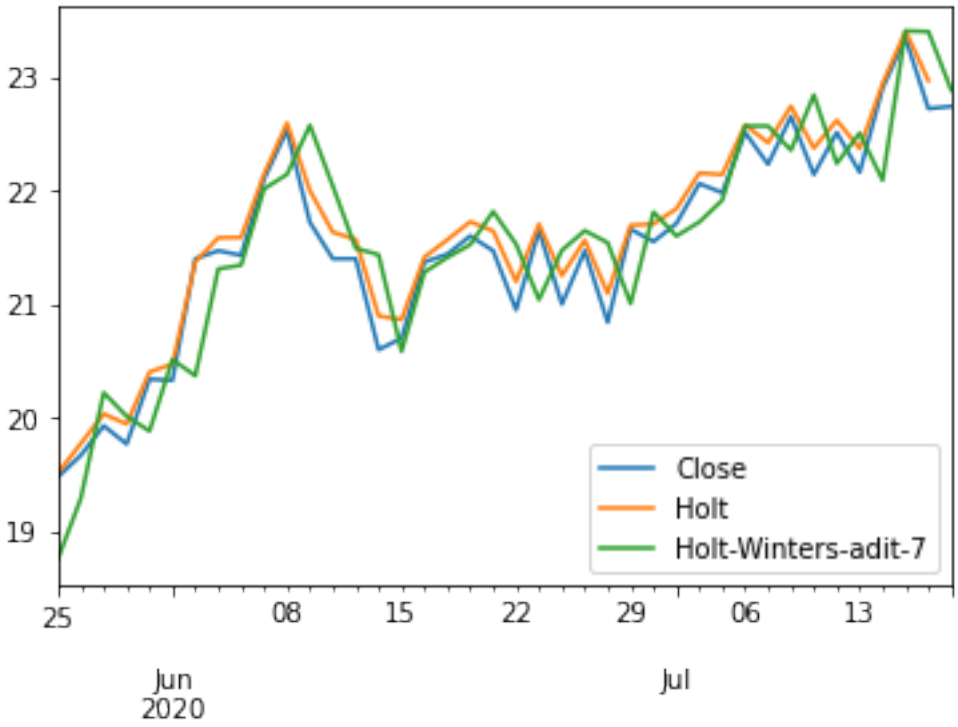
[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb33017c810>
```



7. Olhe as previsões para as últimas 40 observações

[34]:
```
PETR4[['Close','Holt','Holt-Winters-adit-7']][-40:].plot()
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3300a9dd0>
```



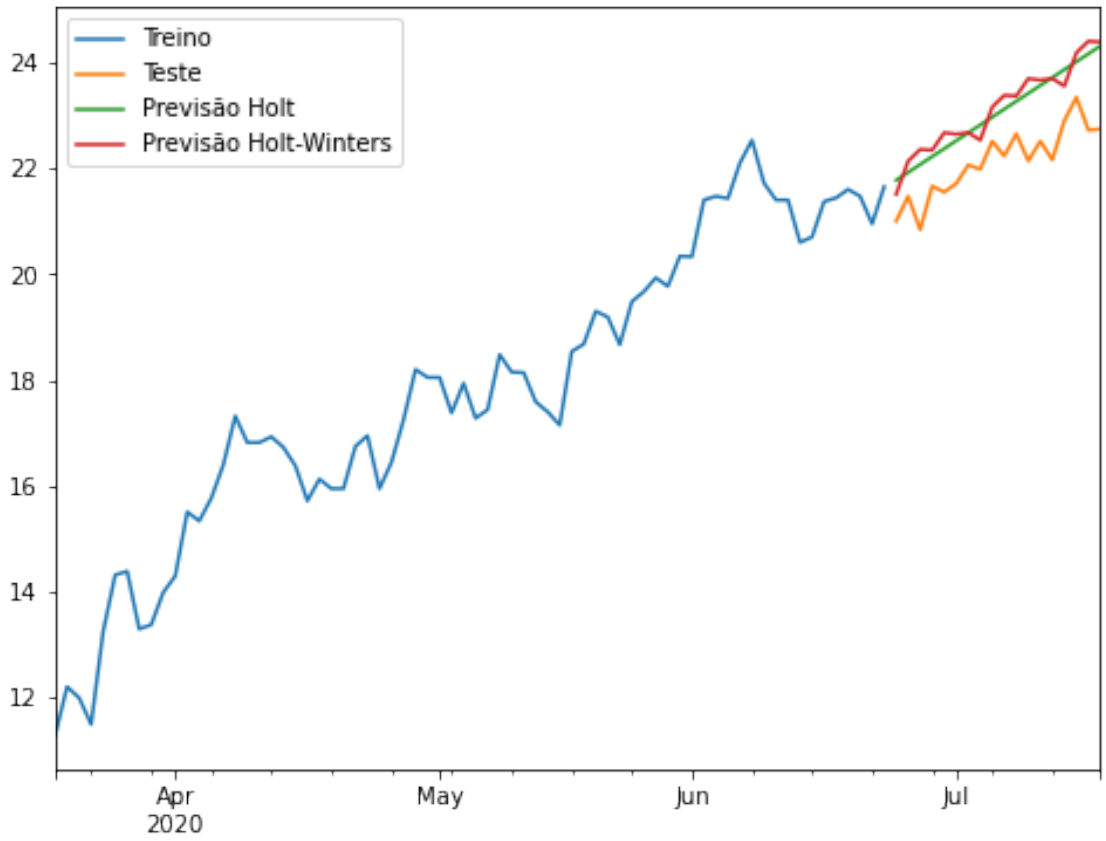
8. Como ficam as previsões? Use cerca de 80% da base para treino e 20% para teste, ou seja, 70 observações para treino e 18 observações para teste.

[35]:
```
dados_treino = PETR4.iloc[:70] # Dados de treinamento até observação 210, sem␣
 ↪incluí-la
dados_teste = PETR4.iloc[70:] # Dados de teste a partir da observação 210
```

[36]:
```
ajustado_H = ExponentialSmoothing(dados_treino['Close'],trend='add').fit()
ajustado_HW =␣
 ↪ExponentialSmoothing(dados_treino['Close'],trend='add',seasonal='add',seasonal_periods=7).
 ↪fit()

predito_H = ajustado_H.forecast(18).rename('Previsão Holt')
predito_HW = ajustado_HW.forecast(18).rename('Previsão Holt')

predito_H.index = PETR4.index[70:]
predito_HW.index = PETR4.index[70:]

dados_treino['Close'].plot(legend=True,label='Treino')
dados_teste['Close'].plot(legend=True,label='Teste',figsize=(8,6))
predito_H.plot(legend=True,label='Previsão Holt');
predito_HW.plot(legend=True,label='Previsão Holt-Winters');
```

```
/home/cibele/anaconda3/lib/python3.7/site-
```

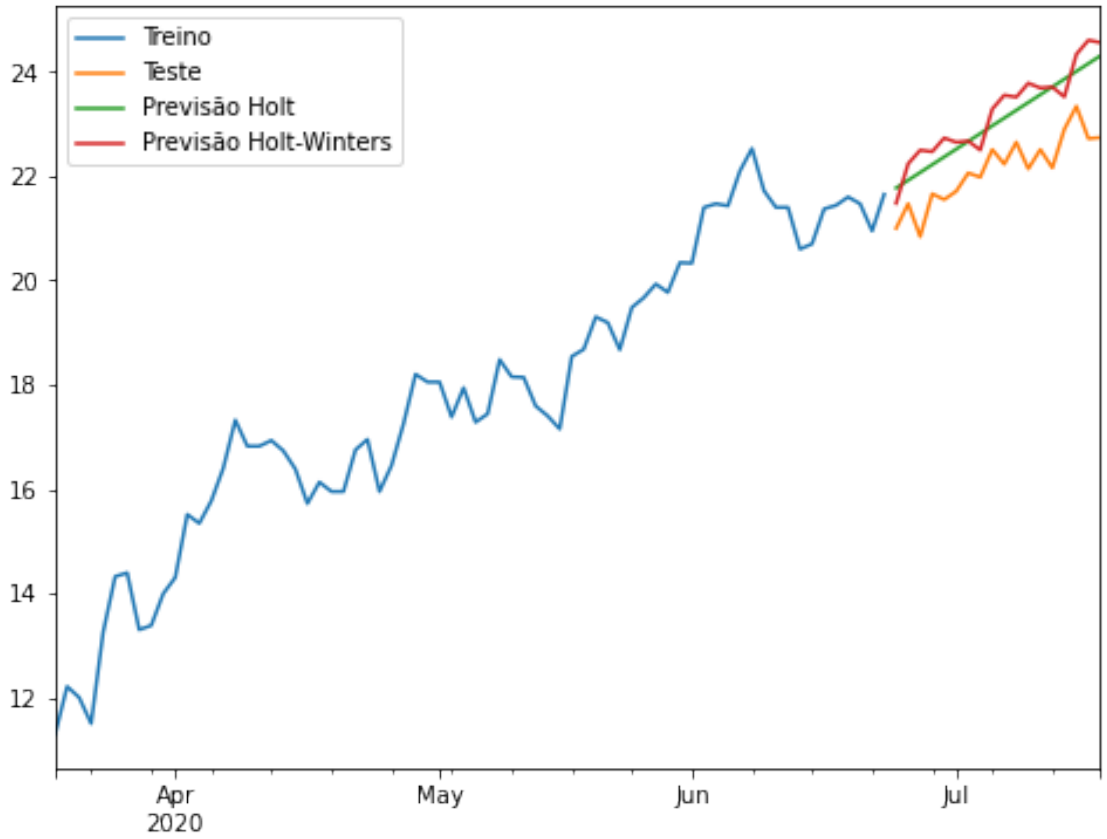
```
packages/statsmodels/tsa/holtwinters.py:744: ConvergenceWarning: Optimization
failed to converge. Check mle_retvals.
  ConvergenceWarning)
```



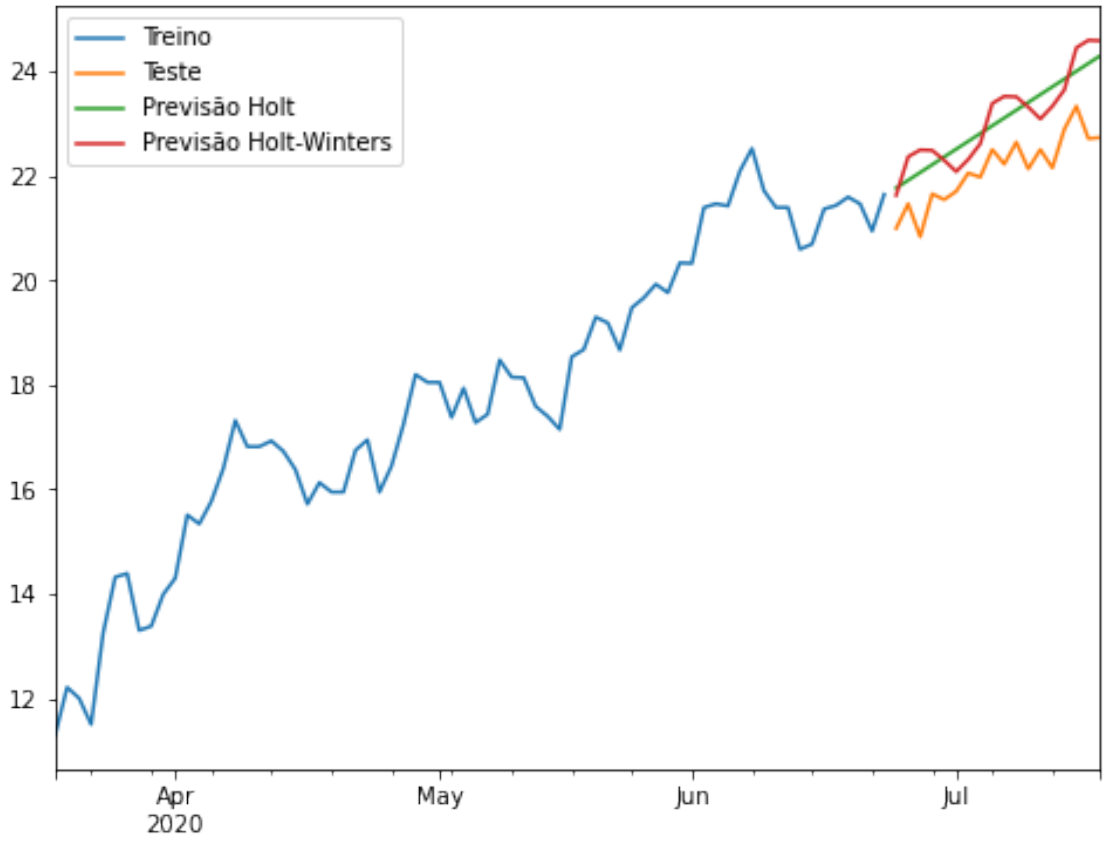
9. É possível melhorar as previsões com o método de Holt-Winters com sazonalidade e/ou tendência multiplicativos?

[37]:
```
ajustado_H = ExponentialSmoothing(dados_treino['Close'],trend='add').fit()
ajustado_HW =␣
 ↪ExponentialSmoothing(dados_treino['Close'],trend='add',seasonal='mul',seasonal_periods=7).
 ↪fit()

predito_H = ajustado_H.forecast(18).rename('Previsão Holt')
predito_HW = ajustado_HW.forecast(18).rename('Previsão Holt')

predito_H.index = PETR4.index[70:]
predito_HW.index = PETR4.index[70:]

dados_treino['Close'].plot(legend=True,label='Treino')
dados_teste['Close'].plot(legend=True,label='Teste',figsize=(8,6))
predito_H.plot(legend=True,label='Previsão Holt');
```

```
predito_HW.plot(legend=True,label='Previsão Holt-Winters');
```



[38]:
```
ajustado_H = ExponentialSmoothing(dados_treino['Close'],trend='add').fit()
ajustado_HW =␣
 ↪ExponentialSmoothing(dados_treino['Close'],trend='mul',seasonal='mul',seasonal_periods=7).
 ↪fit()

predito_H = ajustado_H.forecast(18).rename('Previsão Holt')
predito_HW = ajustado_HW.forecast(18).rename('Previsão Holt')

predito_H.index = PETR4.index[70:]
predito_HW.index = PETR4.index[70:]

dados_treino['Close'].plot(legend=True,label='Treino')
dados_teste['Close'].plot(legend=True,label='Teste',figsize=(8,6))
predito_H.plot(legend=True,label='Previsão Holt');
predito_HW.plot(legend=True,label='Previsão Holt-Winters');
```



10. Utilize os métodos de Holt e Holt-Winters e compare os erros quadráticos médios e erros absolutos médios. (Depois você pode repetir para os dados de COVID-19).

[39]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

[40]:
```
mean_squared_error(dados_teste['Close'],predito_HW)
```

[40]: 1.1681781864242922

[41]:
```
mean_absolute_error(dados_teste['Close'], predito_HW)
```

[41]: 0.9820248490412608

[42]:
```
mean_squared_error(dados_teste['Close'],predito_H)
```

[42]: 0.968769526005945

[43]:
```
mean_absolute_error(dados_teste['Close'], predito_H)
```

[43]: 0.9200445293568733

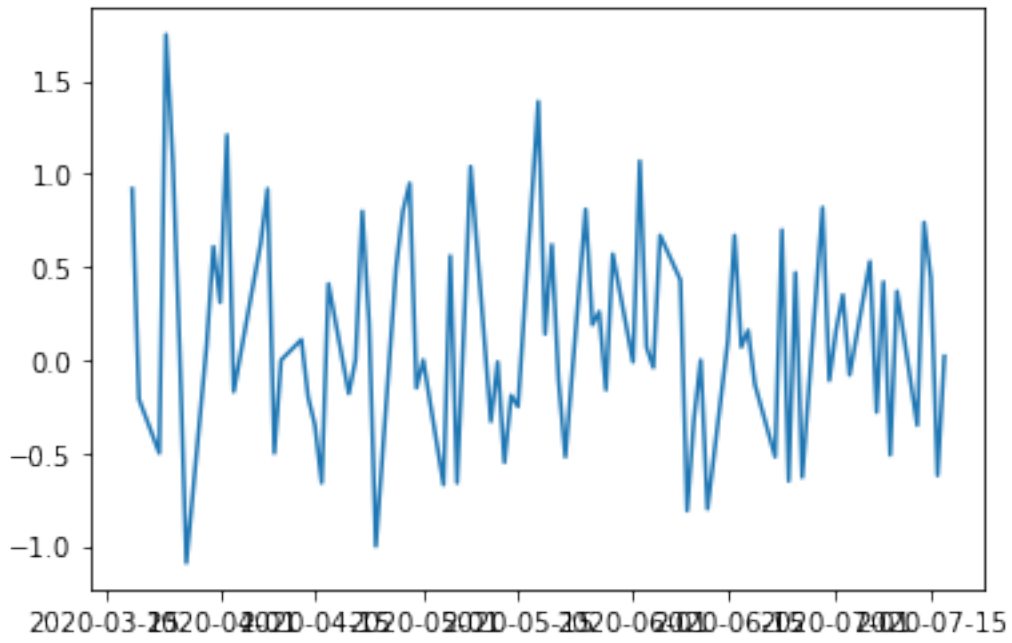
11. Crie séries da média móvel simples e desvio padrão móvel com janelas de 7 dias. A série parece estacionária?

[44]:
```
# Verificando as séries de média e desvio-padrão móvel com janela de 7 dias

PETR4['MMS-7'] = PETR4['Close'].rolling(window=7).mean()
PETR4['DP-7'] = PETR4['Close'].rolling(window=7).std()

PETR4[['Close','MMS-7','DP-7']].plot();
```

A série não parece estacionária.

12. A série das primeiras diferenças parece estacionária?.

[45]:
```
# Primeiras diferenças
y = np.diff(PETR4['Close'])
x = PETR4.iloc[1:].index

plt.plot(x,y);
```

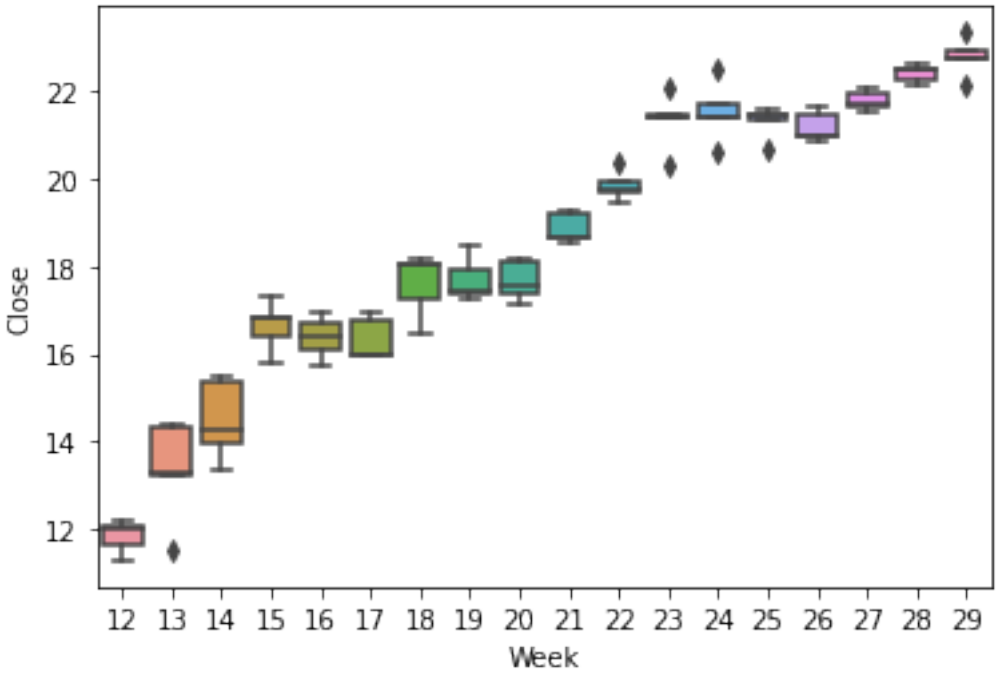


13. Considere boxplots semanais para a série original. Ela parece estacionária?

[46]:
```
import seaborn as sns

PETR4['Week'] = PETR4.index.week

sns.boxplot(x=PETR4['Week'], y=PETR4['Close'])
```

```
/home/cibele/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
FutureWarning: weekofyear and week have been deprecated, please use
DatetimeIndex.isocalendar().week instead, which returns a Series.  To exactly
reproduce the behavior of week and weekofyear and return an Index, you may call
pd.Int64Index(idx.isocalendar().week)
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb326ea3390>
```



14. Utilize o Teste de Dickey-Fuller aumentado. Que evidências ele dá sobre a estacionariedade da série e da primeira diferença?

[47]:
```
from statsmodels.tsa.stattools import adfuller

result = adfuller(PETR4['Close'], autolag='AIC')
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -1.795239
p-value: 0.382828
Critical Values:
        1%: -3.509
        5%: -2.896
        10%: -2.585
```

[48]:
```
from statsmodels.tsa.stattools import adfuller

result = adfuller(np.diff(PETR4['Close']), autolag='AIC')
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
```

```
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -10.506245
p-value: 0.000000
Critical Values:
        1%: -3.509
        5%: -2.896
        10%: -2.585
```

3 Prática 3

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Considere as duas bases de dados a seguir

- Dados de fechamento do papel PETR4 a partir de 18/3 no arquivo PETR4.csv.
- Dados de mortes por COVID-19 no estado de SP.

Nesta prática, aplicaremos os conhecimentos adquiridos na Aula 3 a esses dados, a saber:

- Função de autocorrelação
- Função de autocorrelação parcial
- Repetir a modelagem para os dados de PETR4 visto em aula.
- Desafio para COVID-19: Um modelo ARIMA sazonal.

1. Carregue as bibliotecas

[49]:
```
# Bibliotecas

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # para determinar␣
 ↪(p,q)
```

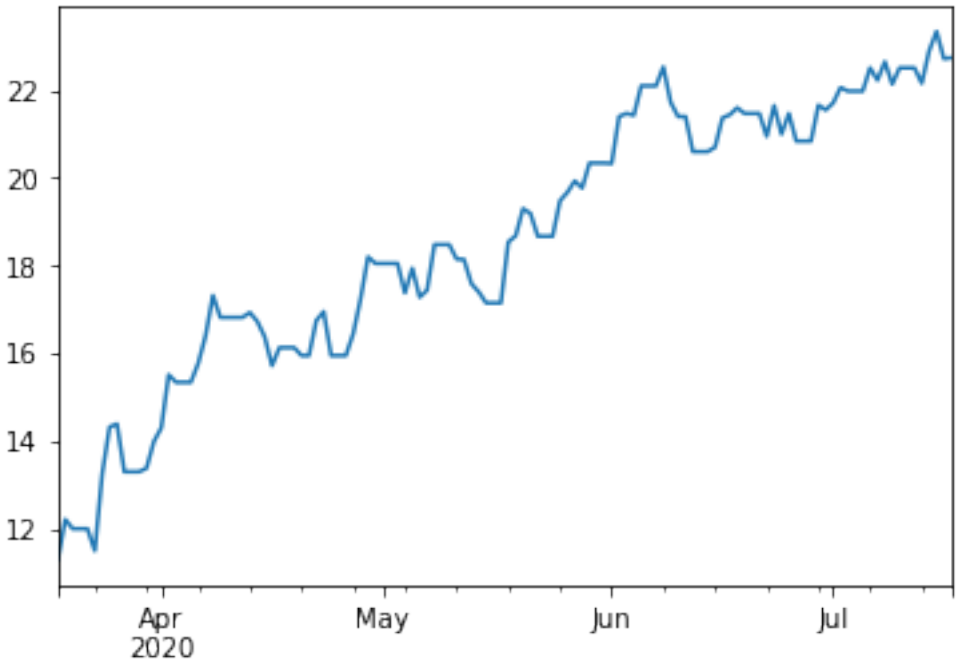
2. Faça a leitura dos dados de PETR4 em df1 e de COVID-19 no estado de SP em df2. Para os dados PETR4 usaremos a variável Close e para os dados de COVID-19 usaremos a variável deaths. Complete as bases de dados usando reindex e fillna.

[50]:
```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

# PETR4 - Leitura dos dados
df1 = pd.read_csv(f'{pkgdir}/PETR4.csv', index_col='Date', parse_dates=True)

idx1 = pd.date_range(start=df1.index.min(), end=df1.index.max(), freq='D')
df1 = df1.reindex(idx1)
df1.fillna(method='ffill', inplace=True)


# COVID-19 no estado de SP
df2 = pd.read_csv(f'{pkgdir}/covidSP.csv', index_col='date', parse_dates=True)

idx2 = pd.date_range(start=df2.index.min(), end=df2.index.max(), freq='D')
df2 = df2.reindex(idx2)
df2.fillna(0,inplace=True)
```

3. Verifique graficamente se os dados foram lidos de forma correta.

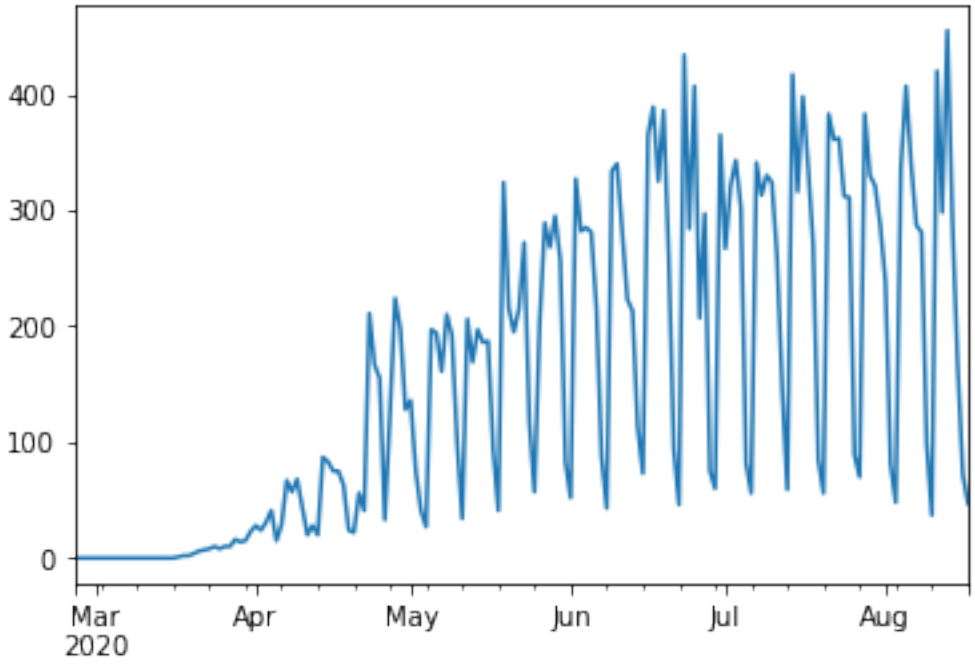
[51]:
```
df1['Close'].plot()
```

[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3328b3610>
```



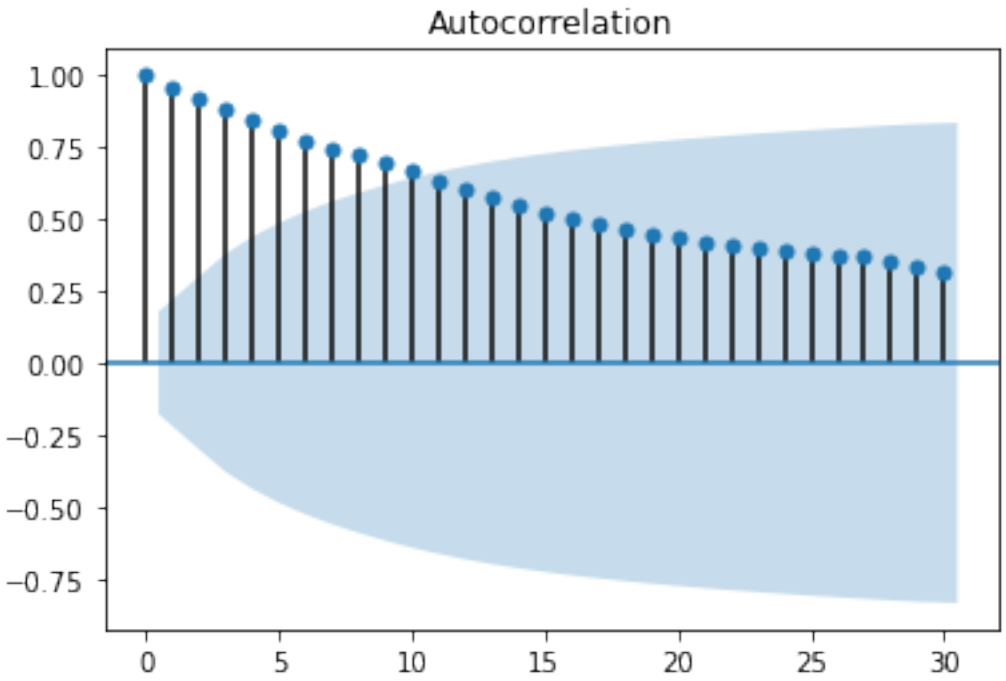
[52]:
```
df2['deaths'].plot()
```

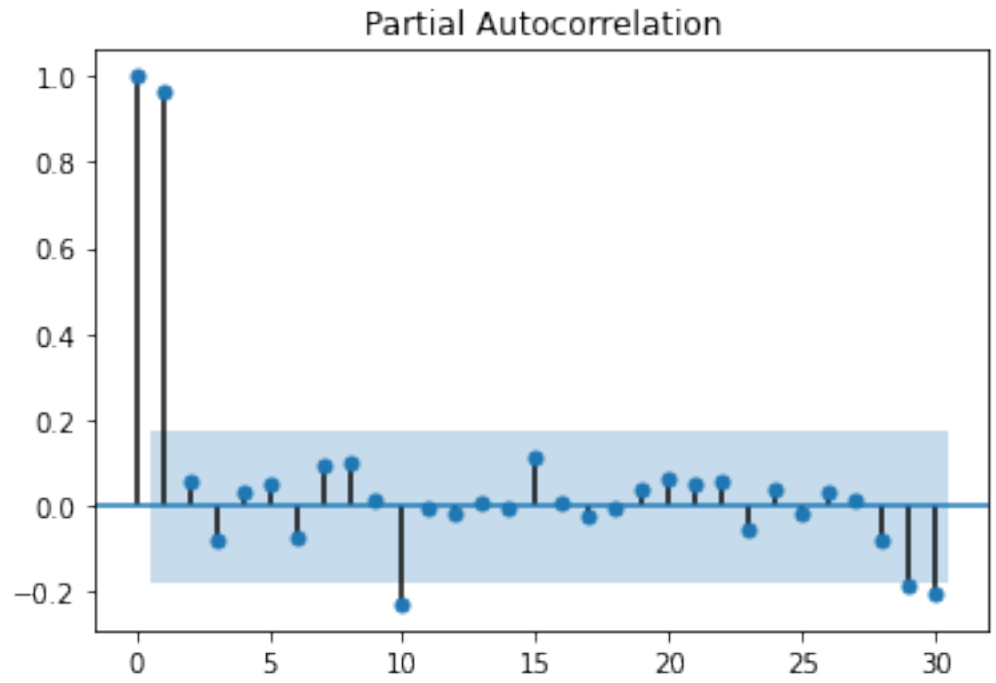
[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3280693d0>



4. Obtenha os gráficos de autocorrelação e autocorrelação parcial para ambas as aplicações. Que características você observa nesses gráficos? Que modelo sugeriria com base nos correlogramas?

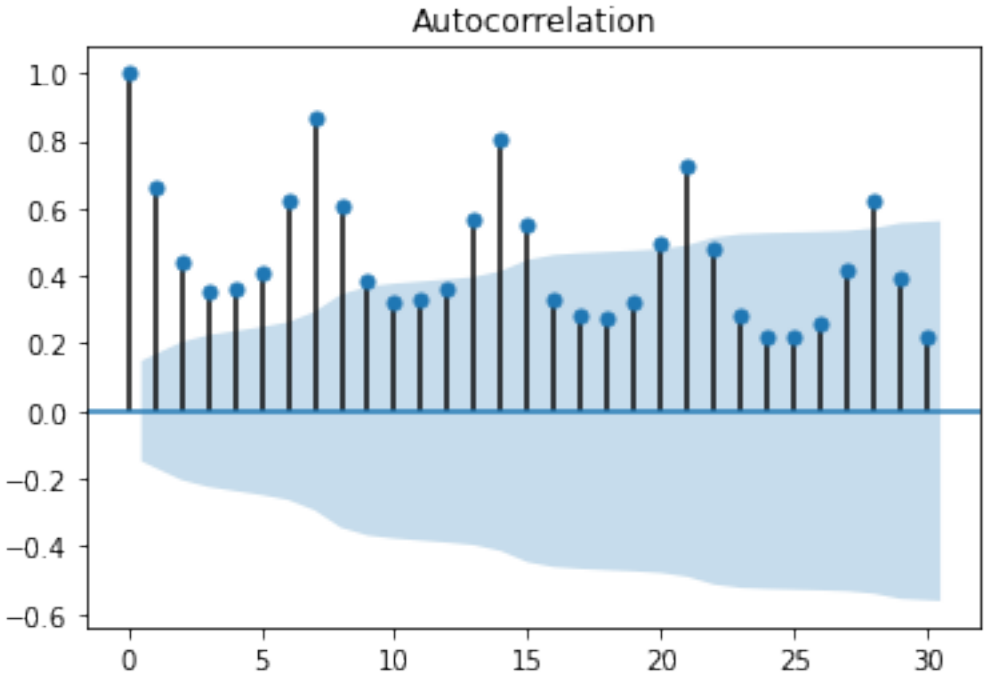
[53]:
```
# Correlograma

plot_acf(df1['Close'], lags=30)
plot_pacf(df1['Close'], lags=30)
plt.show()
```

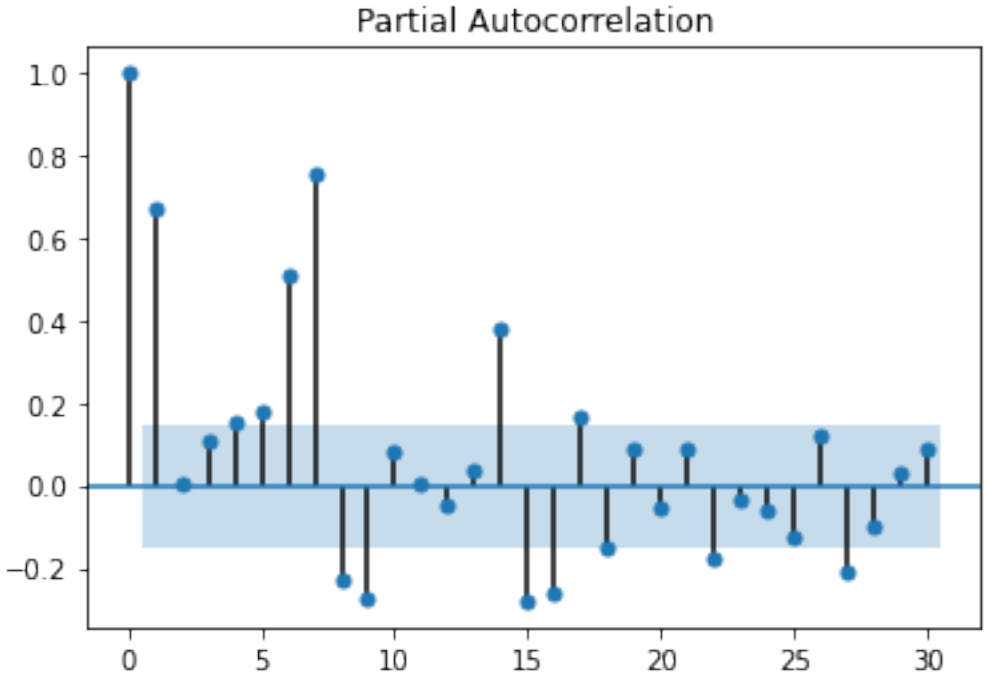




[54]:
```
# Correlograma
```

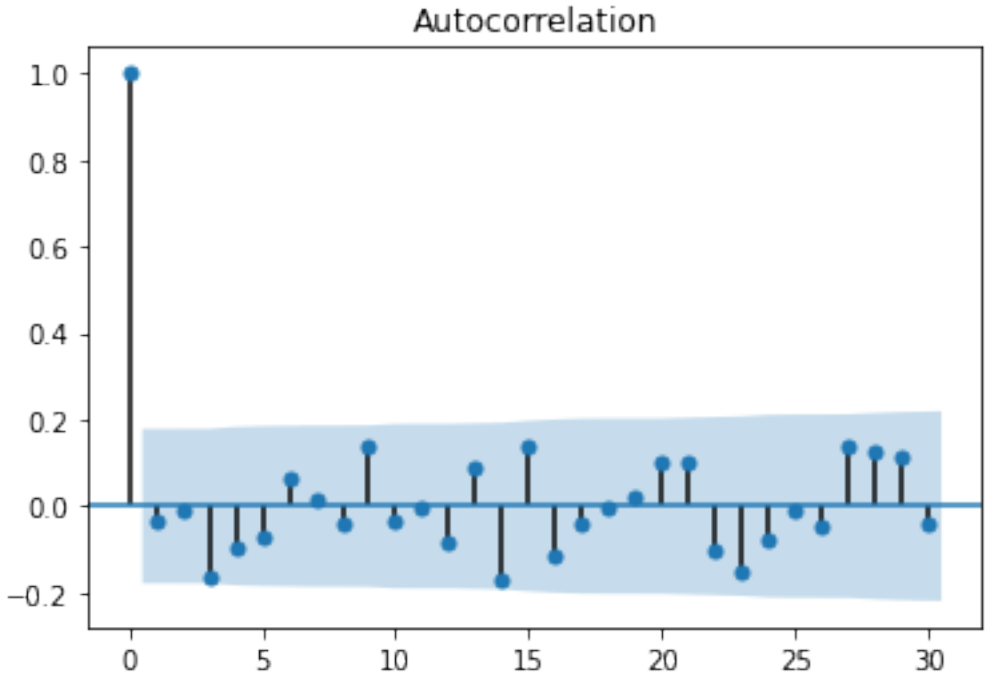
```
plot_acf(df2['deaths'], lags=30)
plot_pacf(df2['deaths'], lags=30)
plt.show()
```

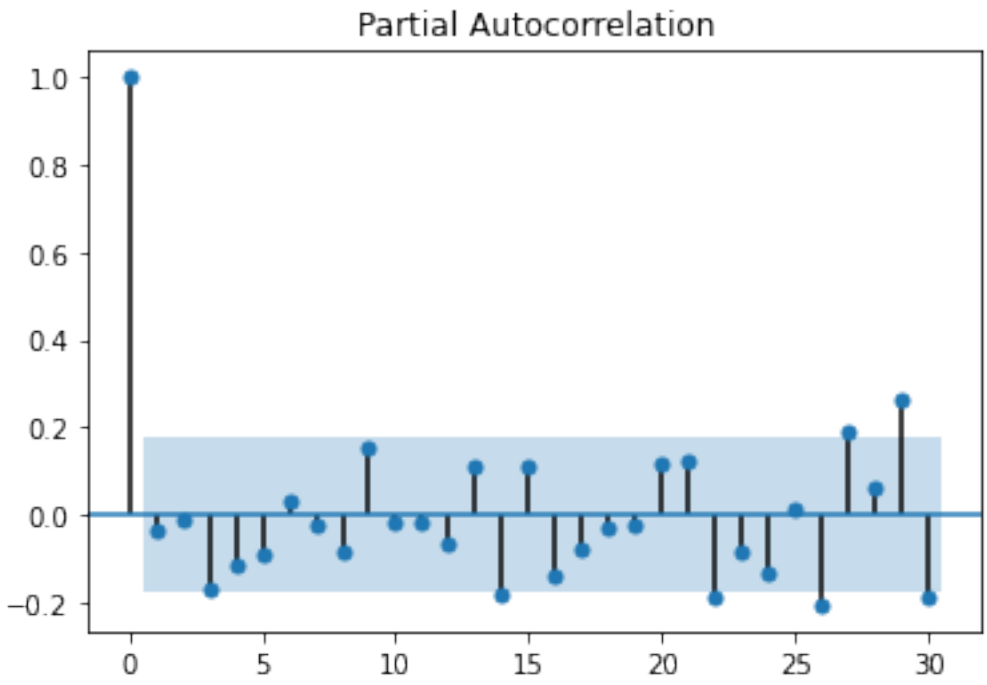




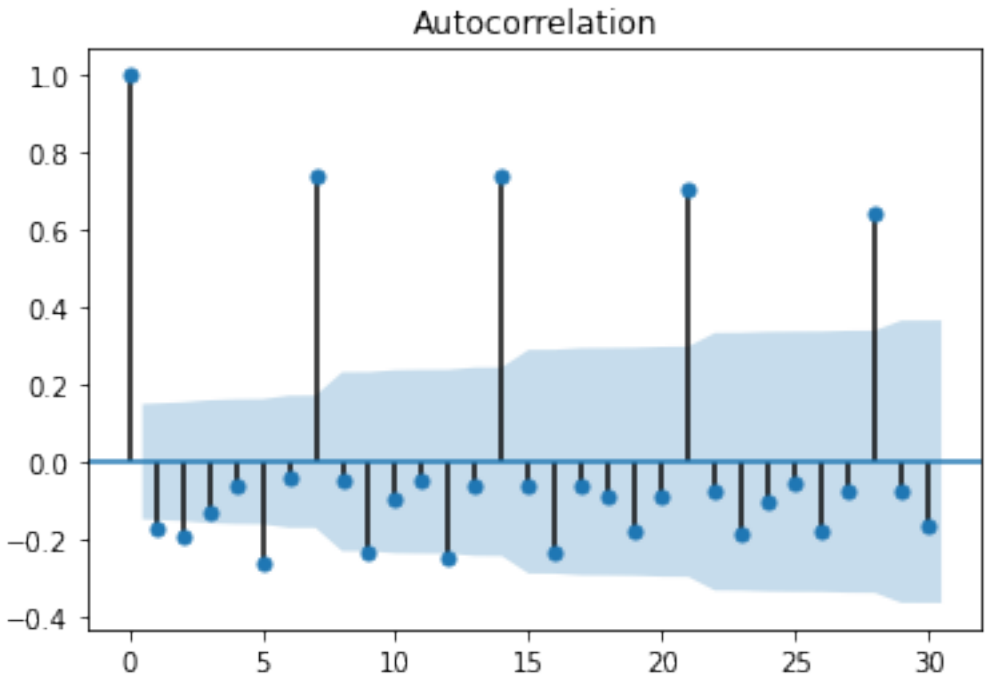
5. Repita os gráficos de autocorrelação e autocorrelação parcial para a primeira diferença de cada aplicação. O que você pode concluir?

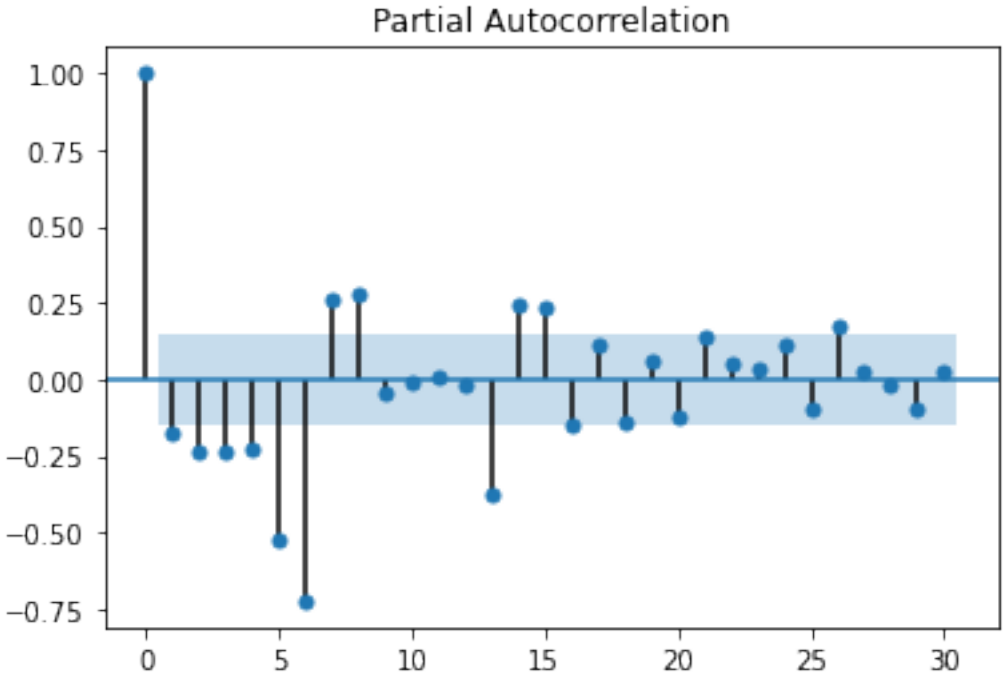
[55]:
```
# Correlograma

plot_acf(np.diff(df1['Close']), lags=30)
plot_pacf(np.diff(df1['Close']), lags=30)
plt.show()
```





[56]:
```
# Correlograma

plot_acf(np.diff(df2['deaths']), lags=30)
plot_pacf(np.diff(df2['deaths']), lags=30)
plt.show()
```





6. Para os dados de PETR4, refaça a modelagem vista em aula.

7. Desafio: Ainda não vimos o modelo SARIMA, mas já sabemos que a série de COVID-19

tem sazonalidade. Como os dados de COVID-19 apresentam sazonalidade, vamos arriscar no tema da próxima aula? Altere seasonal para True e escolha o melhor modelo entre os SARIMA propostos. Discutiremos com detalhes este modelo na Aula 4.

[57]:
```
from pmdarima import auto_arima
```

[58]:
```
stepwise_fit = auto_arima(df2['deaths'], start_p=0, start_q=0,
                          max_p=6, max_q=3, m=7,
                          seasonal=True,
                          d=0, trace=True,
                          error_action='ignore',   # we don't want to know if an␣
 ↪order does not work
                          suppress_warnings=True,  # we don't want convergence␣
 ↪warnings
                          stepwise=True)           # set to stepwise

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
Fit ARIMA(0,0,0)x(1,1,1,7) [intercept=True]; AIC=1754.509, BIC=1766.981,
Time=0.329 seconds
Fit ARIMA(0,0,0)x(0,1,0,7) [intercept=True]; AIC=1796.968, BIC=1803.204,
Time=0.013 seconds
Fit ARIMA(1,0,0)x(1,1,0,7) [intercept=True]; AIC=1756.055, BIC=1768.527,
Time=0.227 seconds
Fit ARIMA(0,0,1)x(0,1,1,7) [intercept=True]; AIC=1757.606, BIC=1770.078,
Time=0.185 seconds
Fit ARIMA(0,0,0)x(0,1,0,7) [intercept=False]; AIC=1801.627, BIC=1804.745,
Time=0.018 seconds
Fit ARIMA(0,0,0)x(0,1,1,7) [intercept=True]; AIC=1758.455, BIC=1767.809,
Time=0.116 seconds
Fit ARIMA(0,0,0)x(1,1,0,7) [intercept=True]; AIC=1755.924, BIC=1765.278,
Time=0.316 seconds
Fit ARIMA(0,0,0)x(2,1,1,7) [intercept=True]; AIC=1754.242, BIC=1769.832,
Time=0.627 seconds
Fit ARIMA(0,0,0)x(2,1,0,7) [intercept=True]; AIC=1752.851, BIC=1765.323,
Time=0.513 seconds
Fit ARIMA(1,0,0)x(2,1,0,7) [intercept=True]; AIC=1750.339, BIC=1765.929,
Time=0.456 seconds
Fit ARIMA(1,0,0)x(2,1,1,7) [intercept=True]; AIC=1752.226, BIC=1770.934,
Time=0.691 seconds
Fit ARIMA(1,0,0)x(1,1,1,7) [intercept=True]; AIC=1752.525, BIC=1768.115,
Time=0.293 seconds
Fit ARIMA(2,0,0)x(2,1,0,7) [intercept=True]; AIC=1745.229, BIC=1763.937,
Time=0.568 seconds
Fit ARIMA(2,0,0)x(1,1,0,7) [intercept=True]; AIC=1749.753, BIC=1765.343,
Time=0.340 seconds
Fit ARIMA(2,0,0)x(2,1,1,7) [intercept=True]; AIC=1746.878, BIC=1768.704,
```

```
Time=1.142 seconds
Fit ARIMA(2,0,0)x(1,1,1,7) [intercept=True]; AIC=1746.977, BIC=1765.685,
Time=0.434 seconds
Fit ARIMA(3,0,0)x(2,1,0,7) [intercept=True]; AIC=1745.615, BIC=1767.440,
Time=0.767 seconds
Fit ARIMA(2,0,1)x(2,1,0,7) [intercept=True]; AIC=1743.980, BIC=1765.806,
Time=1.486 seconds
Fit ARIMA(2,0,1)x(1,1,0,7) [intercept=True]; AIC=1746.788, BIC=1765.496,
Time=0.497 seconds
Fit ARIMA(2,0,1)x(2,1,1,7) [intercept=True]; AIC=1747.926, BIC=1772.870,
Time=1.378 seconds
Fit ARIMA(2,0,1)x(1,1,1,7) [intercept=True]; AIC=1745.183, BIC=1767.009,
Time=0.664 seconds
Fit ARIMA(1,0,1)x(2,1,0,7) [intercept=True]; AIC=1741.868, BIC=1760.576,
Time=0.975 seconds
Fit ARIMA(1,0,1)x(1,1,0,7) [intercept=True]; AIC=1753.036, BIC=1768.626,
Time=0.565 seconds
Fit ARIMA(1,0,1)x(2,1,1,7) [intercept=True]; AIC=1743.852, BIC=1765.677,
Time=1.257 seconds
Fit ARIMA(1,0,1)x(1,1,1,7) [intercept=True]; AIC=1744.100, BIC=1762.808,
Time=0.625 seconds
Fit ARIMA(0,0,1)x(2,1,0,7) [intercept=True]; AIC=1751.838, BIC=1767.428,
Time=0.517 seconds
Fit ARIMA(1,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.499, BIC=1766.325,
Time=1.251 seconds
Fit ARIMA(0,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.811, BIC=1763.519,
Time=0.619 seconds
Fit ARIMA(2,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.103, BIC=1769.047,
Time=1.522 seconds
Total fit time: 18.447 seconds
```

[58]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==========================================================================
==========
Dep. Variable:                                   y   No. Observations:
174
Model:             SARIMAX(1, 0, 1)x(2, 1, [], 7)   Log Likelihood
-864.934
Date:                            Sun, 20 Sep 2020   AIC
1741.868
Time:                                    22:37:49   BIC
1760.576
Sample:                                         0   HQIC
1749.461
                                              - 174
```

```
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.4443      0.928      0.479      0.632      -1.375       2.264
ar.L1          0.9725      0.036     27.280      0.000       0.903       1.042
ma.L1         -0.8866      0.059    -15.020      0.000      -1.002      -0.771
ar.S.L7       -0.7035      0.064    -10.932      0.000      -0.830      -0.577
ar.S.L14      -0.2888      0.083     -3.466      0.001      -0.452      -0.126
sigma2      1805.7756    178.328     10.126      0.000    1456.259    2155.292
==============================================================================
===
Ljung-Box (Q):                       40.45   Jarque-Bera (JB):
5.42
Prob(Q):                              0.45   Prob(JB):
0.07
Heteroskedasticity (H):               2.05   Skew:
0.29
Prob(H) (two-sided):                  0.01   Kurtosis:
3.67
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

4 Prática 4

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Considere a base de dados de mortes por COVID-19 no estado de SP.

Nesta prática, vamos trabalhar com a análise de má-especificação de modelos, ou seja, como fica a análise de diagnóstico se ajustarmos um modelo "correto" e um modelo "errado".

1. Carregue as bibliotecas que serão utilizadas.

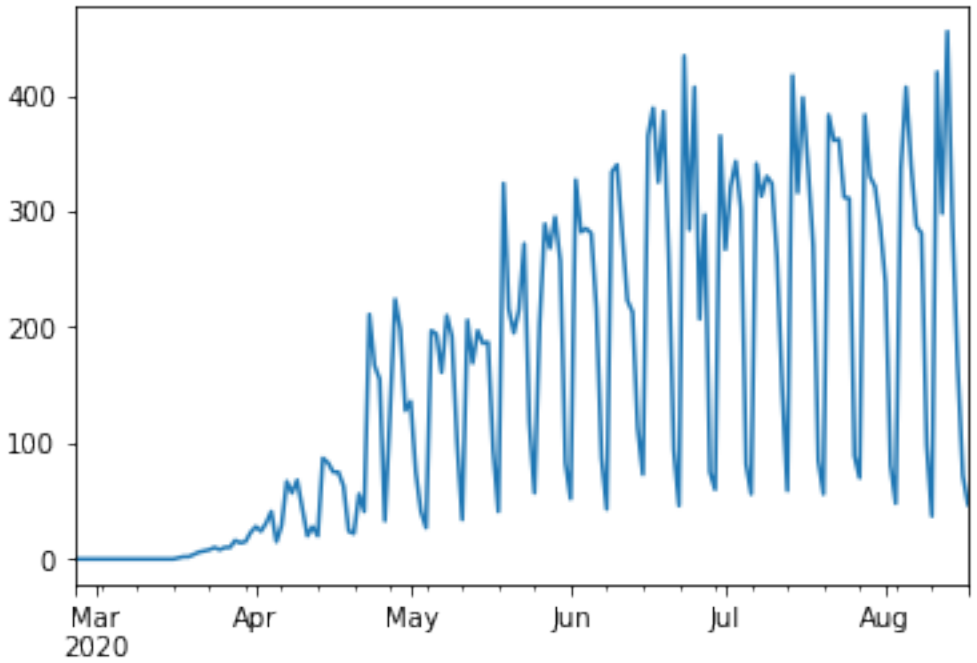
[59]:
```
import pandas as pd
import numpy as np
%matplotlib inline

from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from pmdarima import auto_arima

# Ignorar warnings não prejudiciais
import warnings
warnings.filterwarnings("ignore")

np.random.seed(0)
```

2. Faça a leitura dos dados de COVID-19 do estado de SP e complete os dados faltantes. Verifique se os dados estão Ok.

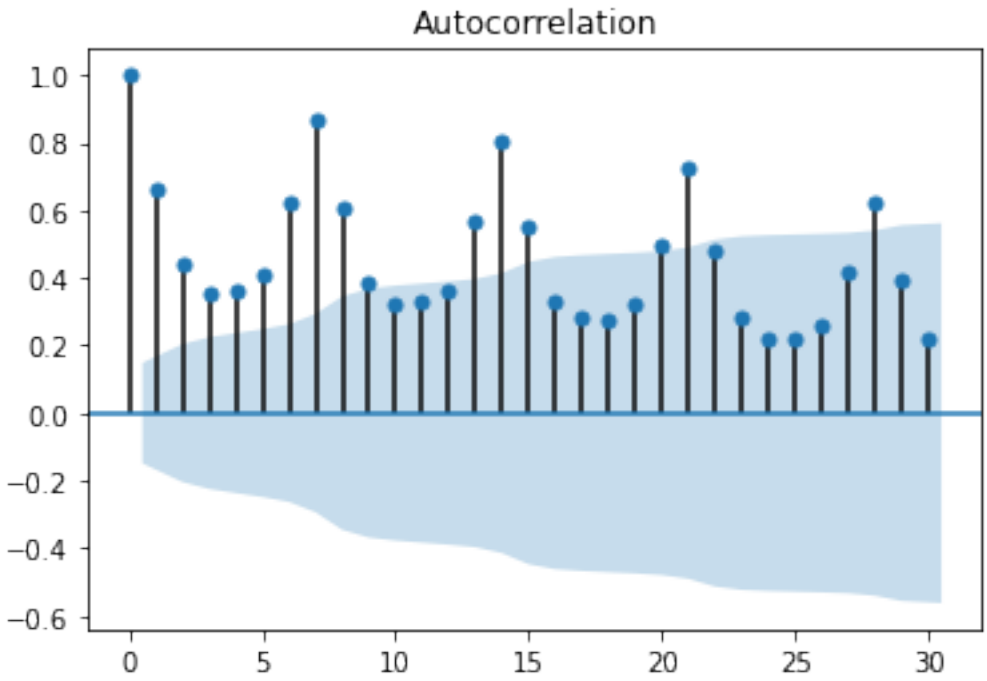
[60]:
```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

# Leitura dos dados de COVID-19 no estado de SP - vamos trabalhar com as mortes
covidSP = pd.read_csv(f'{pkgdir}/covidSP.csv', index_col='date',␣
 ↪parse_dates=True)

idx = pd.date_range(start=covidSP.index.min(), end=covidSP.index.max(), freq='D')
covidSP = covidSP.reindex(idx)
covidSP.fillna(0,inplace=True)

covidSP['deaths'].plot()
```

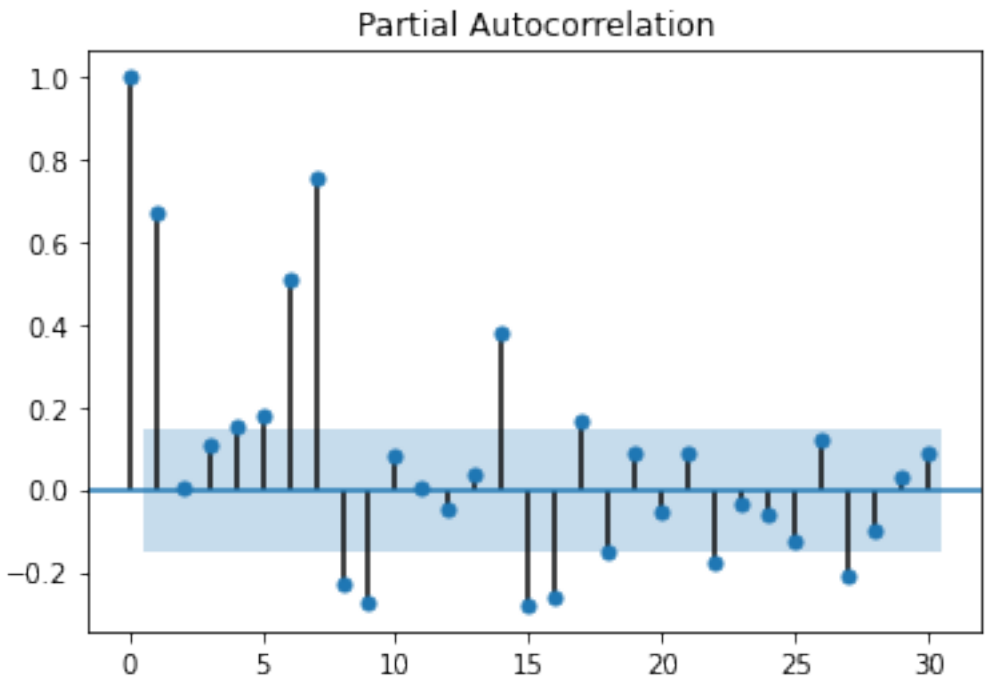
[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb324d25610>
```



3. Repita a análise com gráficos de autocorrelação e autocorrelação parcial feitos em aula. Já é possível identificar características do melhor modelo para esses dados?

[61]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # para determinar␣
 ↪(p,q)

# Correlograma

plot_acf(covidSP['deaths'], lags=30)
plot_pacf(covidSP['deaths'], lags=30)
plt.show()
```





4. Utilize o stepwise_fit para escolher as ordens de um modelo SARIMA (p,d,q)x(P,D,Q)m, como feito em aula

[62]:
```
# Ajuste de modelo SARIMA

auto_arima(covidSP['deaths'],seasonal=True,m=7).summary()


stepwise_fit = auto_arima(covidSP['deaths'], start_p=0, start_q=0,
                          max_p=6, max_q=3, m=7,
                          seasonal=True,
                          trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
Fit ARIMA(0,0,0)x(1,1,1,7) [intercept=True]; AIC=1754.509, BIC=1766.981,
Time=0.296 seconds
Fit ARIMA(0,0,0)x(0,1,0,7) [intercept=True]; AIC=1796.968, BIC=1803.204,
Time=0.013 seconds
Fit ARIMA(1,0,0)x(1,1,0,7) [intercept=True]; AIC=1756.055, BIC=1768.527,
Time=0.223 seconds
```

```
Fit ARIMA(0,0,1)x(0,1,1,7) [intercept=True]; AIC=1757.606, BIC=1770.078,
Time=0.183 seconds
Fit ARIMA(0,0,0)x(0,1,0,7) [intercept=False]; AIC=1801.627, BIC=1804.745,
Time=0.010 seconds
Fit ARIMA(0,0,0)x(0,1,1,7) [intercept=True]; AIC=1758.455, BIC=1767.809,
Time=0.111 seconds
Fit ARIMA(0,0,0)x(1,1,0,7) [intercept=True]; AIC=1755.924, BIC=1765.278,
Time=0.312 seconds
Fit ARIMA(0,0,0)x(2,1,1,7) [intercept=True]; AIC=1754.242, BIC=1769.832,
Time=0.574 seconds
Fit ARIMA(0,0,0)x(2,1,0,7) [intercept=True]; AIC=1752.851, BIC=1765.323,
Time=0.514 seconds
Fit ARIMA(1,0,0)x(2,1,0,7) [intercept=True]; AIC=1750.339, BIC=1765.929,
Time=0.462 seconds
Fit ARIMA(1,0,0)x(2,1,1,7) [intercept=True]; AIC=1752.226, BIC=1770.934,
Time=0.660 seconds
Fit ARIMA(1,0,0)x(1,1,1,7) [intercept=True]; AIC=1752.525, BIC=1768.115,
Time=0.287 seconds
Fit ARIMA(2,0,0)x(2,1,0,7) [intercept=True]; AIC=1745.229, BIC=1763.937,
Time=0.557 seconds
Fit ARIMA(2,0,0)x(1,1,0,7) [intercept=True]; AIC=1749.753, BIC=1765.343,
Time=0.325 seconds
Fit ARIMA(2,0,0)x(2,1,1,7) [intercept=True]; AIC=1746.878, BIC=1768.704,
Time=1.111 seconds
Fit ARIMA(2,0,0)x(1,1,1,7) [intercept=True]; AIC=1746.977, BIC=1765.685,
Time=0.425 seconds
Fit ARIMA(3,0,0)x(2,1,0,7) [intercept=True]; AIC=1745.615, BIC=1767.440,
Time=0.664 seconds
Fit ARIMA(2,0,1)x(2,1,0,7) [intercept=True]; AIC=1743.980, BIC=1765.806,
Time=1.121 seconds
Fit ARIMA(2,0,1)x(1,1,0,7) [intercept=True]; AIC=1746.788, BIC=1765.496,
Time=0.490 seconds
Fit ARIMA(2,0,1)x(2,1,1,7) [intercept=True]; AIC=1747.926, BIC=1772.870,
Time=1.369 seconds
Fit ARIMA(2,0,1)x(1,1,1,7) [intercept=True]; AIC=1745.183, BIC=1767.009,
Time=0.652 seconds
Fit ARIMA(1,0,1)x(2,1,0,7) [intercept=True]; AIC=1741.868, BIC=1760.576,
Time=0.952 seconds
Fit ARIMA(1,0,1)x(1,1,0,7) [intercept=True]; AIC=1753.036, BIC=1768.626,
Time=0.531 seconds
Fit ARIMA(1,0,1)x(2,1,1,7) [intercept=True]; AIC=1743.852, BIC=1765.677,
Time=1.228 seconds
Fit ARIMA(1,0,1)x(1,1,1,7) [intercept=True]; AIC=1744.100, BIC=1762.808,
Time=0.621 seconds
Fit ARIMA(0,0,1)x(2,1,0,7) [intercept=True]; AIC=1751.838, BIC=1767.428,
Time=0.499 seconds
Fit ARIMA(1,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.499, BIC=1766.325,
Time=1.223 seconds
```

```
Fit ARIMA(0,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.811, BIC=1763.519,
Time=0.523 seconds
Fit ARIMA(2,0,2)x(2,1,0,7) [intercept=True]; AIC=1744.103, BIC=1769.047,
Time=1.500 seconds
Total fit time: 17.501 seconds
```

[62]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                     y   No. Observations:
174
Model:             SARIMAX(1, 0, 1)x(2, 1, [], 7)   Log Likelihood
-864.934
Date:                            Sun, 20 Sep 2020   AIC
1741.868
Time:                                    22:38:25   BIC
1760.576
Sample:                                         0   HQIC
1749.461
                                              - 174
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.4443      0.928      0.479      0.632      -1.375       2.264
ar.L1          0.9725      0.036     27.280      0.000       0.903       1.042
ma.L1         -0.8866      0.059    -15.020      0.000      -1.002      -0.771
ar.S.L7       -0.7035      0.064    -10.932      0.000      -0.830      -0.577
ar.S.L14      -0.2888      0.083     -3.466      0.001      -0.452      -0.126
sigma2      1805.7756    178.328     10.126      0.000    1456.259    2155.292
==============================================================================
===
Ljung-Box (Q):                       40.45   Jarque-Bera (JB):
5.42
Prob(Q):                              0.45   Prob(JB):
0.07
Heteroskedasticity (H):               2.05   Skew:
0.29
Prob(H) (two-sided):                  0.01   Kurtosis:
3.67
==========================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
```

```
step).
"""
```

5. Atribua a um objeto modelo1 o modelo escolhido pelo stepwise_fit e previsões1 os valores preditos por ele

[63]:
```
modelo1 = SARIMAX(covidSP['deaths'],order=(4,0,0),seasonal_order=(2,1,0,7))
resultado1 = modelo1.fit()

previsões1 = resultado1.predict(start=covidSP.index.min(), end=covidSP.index.
 ↪max(), dynamic=False, typ='levels')
previsões1.index = covidSP.index
```

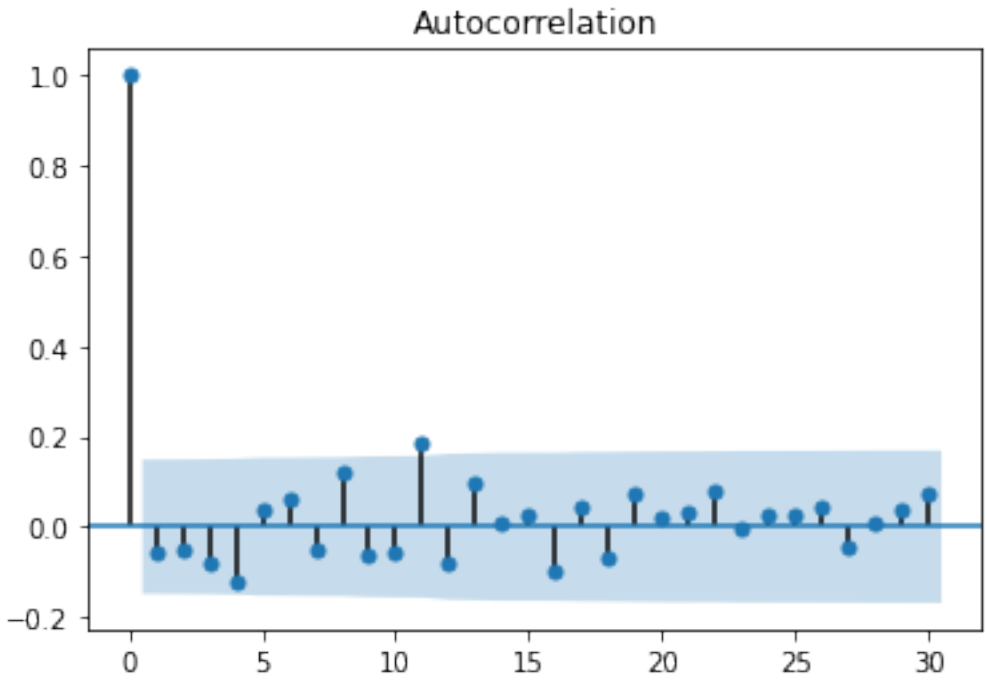
6. Ajuste um modelo "errado" aos dados, por exemplo um SARIMA(0,0,1)x(0,0,2,7), ou seja, com termos de médias móveis e não autorregressivos. Atribua o ajuste e valores preditos aos objetos modelo2 e previsões2.

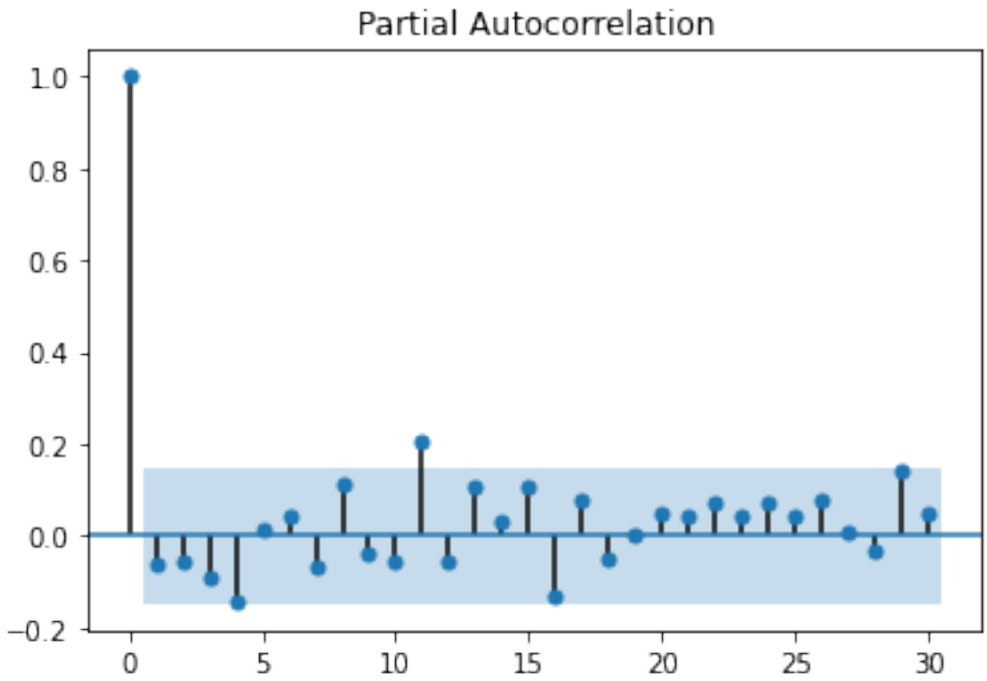
[64]:
```
modelo2 = SARIMAX(covidSP['deaths'],order=(0,0,1),seasonal_order=(0,0,2,7))
resultado2 = modelo2.fit()

previsões2 = resultado2.predict(start=covidSP.index.min(), end=covidSP.index.
 ↪max(), dynamic=False, typ='levels')
previsões2.index = covidSP.index
```

7. Faça uma análise de diagnóstico usando os resíduos obtidos pelo modelo1 e modelo2. Compare, por exemplo, os gráficos de autocorrelação e autocorrelação parcial.

[65]:
```
resíduos1 = resultado1.resid
resíduos2 = resultado2.resid
```

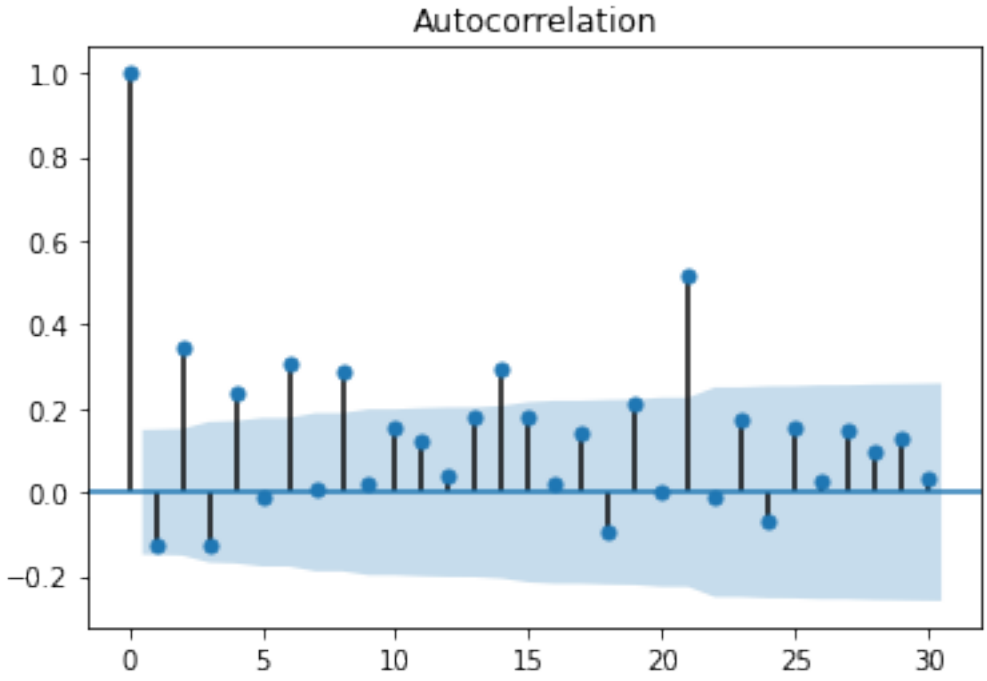
[66]:
```
plot_acf(resíduos1, lags=30)
plot_pacf(resíduos1, lags=30)
plt.show()
```

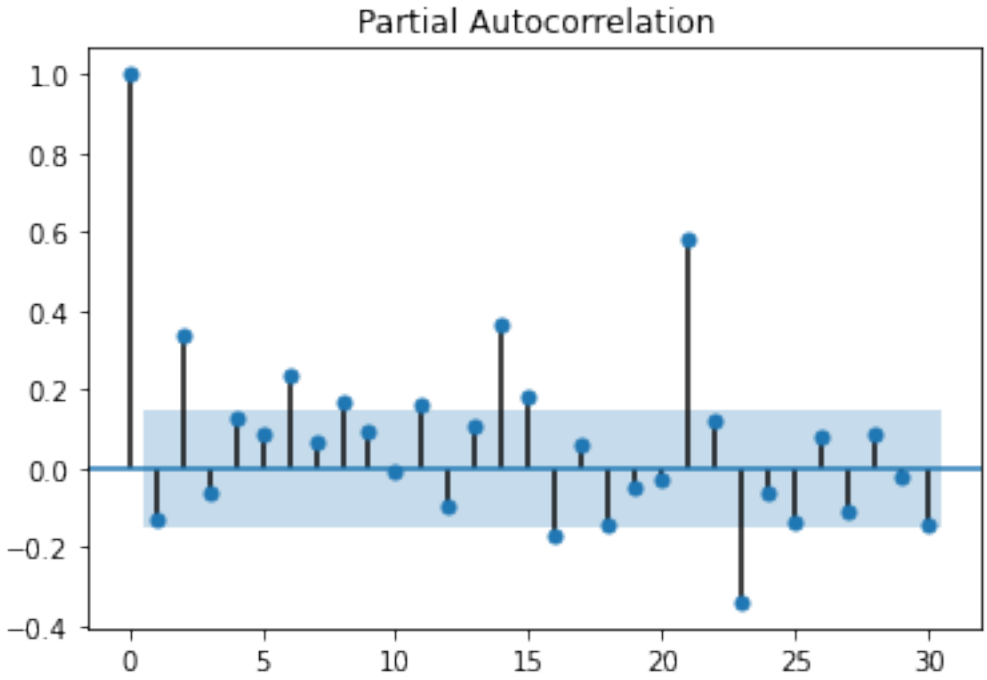




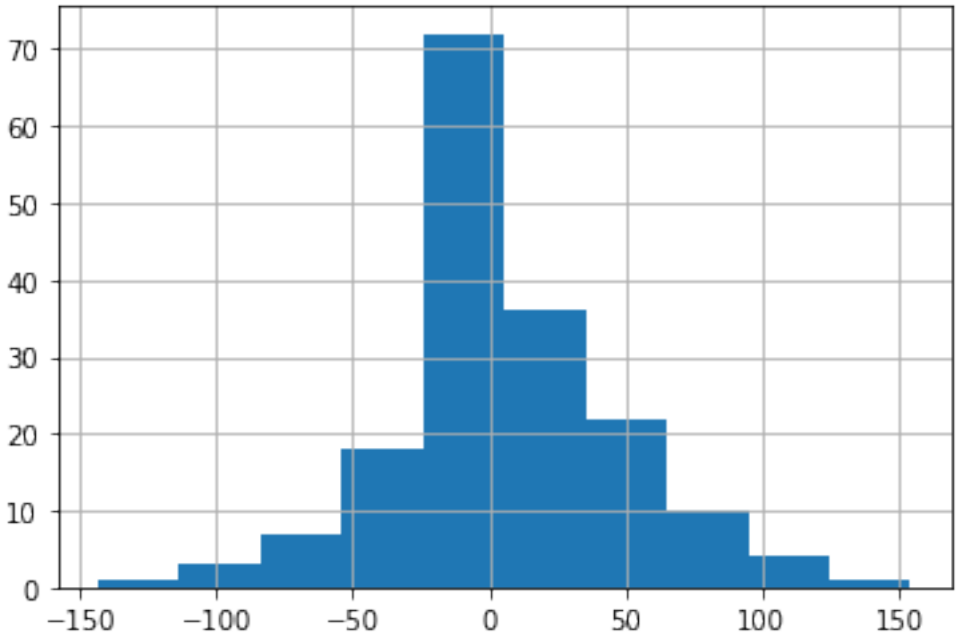
[67]:
```
plot_acf(resíduos2, lags=30)
plot_pacf(resíduos2, lags=30)
```

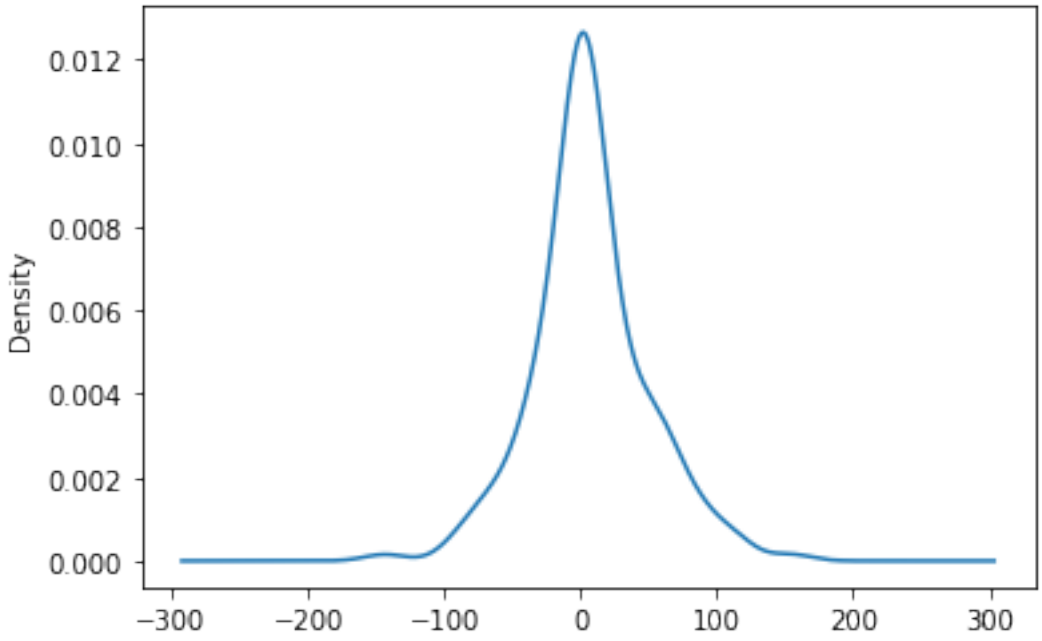
```
plt.show()
```


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
25
30

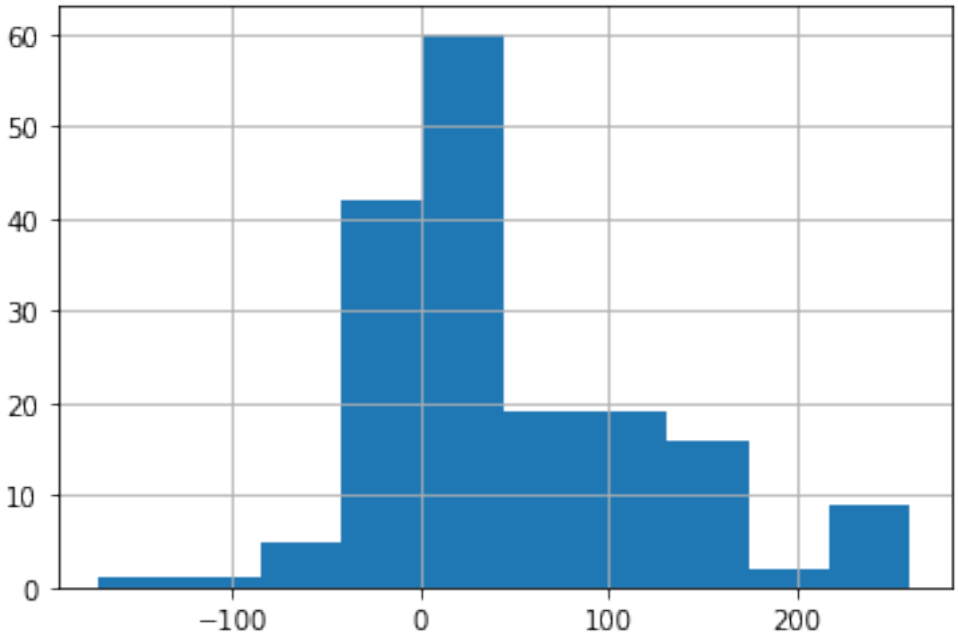

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
5
10
15
20
25
30

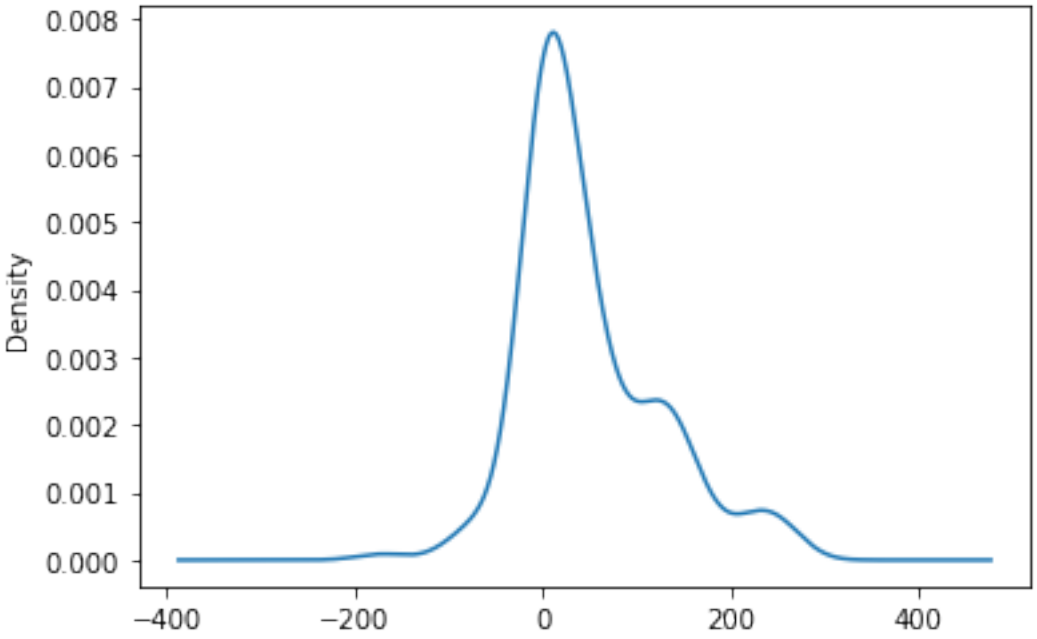
[68]:
```
from matplotlib import pyplot


resíduos1.hist()

pyplot.show()
resíduos1.plot(kind='kde')
pyplot.show()
```





[69]:
```
from matplotlib import pyplot


resíduos2.hist()

pyplot.show()
resíduos2.plot(kind='kde')
pyplot.show()
```





8. Compare as métricas erro quadrático médio e erro absoluto médio. Existe muita diferença entre eles?

[70]:
```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(covidSP['deaths'], previsões1)
print(f'EQM SARIMA(4,0,0)(2,1,0,7): {error:11.10}')
```

```
EQM SARIMA(4,0,0)(2,1,0,7): 1798.007755
```

[71]:
```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(covidSP['deaths'], previsões2)
print(f'EQM SARIMA(0,0,4)(0,1,2,7): {error:11.10}')
```

```
EQM SARIMA(0,0,4)(0,1,2,7): 7746.331189
```

[72]:
```
from statsmodels.tools.eval_measures import rmse

error = rmse(covidSP['deaths'], previsões1)
print(f'REQM SARIMA(4,0,0)(2,1,0,7): {error:11.10}')
```

```
REQM SARIMA(4,0,0)(2,1,0,7): 42.40292154
```

[73]:
```
from statsmodels.tools.eval_measures import rmse

error = rmse(covidSP['deaths'], previsões2)
print(f'REQM SARIMA(0,0,4)(0,1,2,7):  88.0132444')
```

```
REQM SARIMA(0,0,4)(0,1,2,7):  88.0132444
```

9. Utilize outros métodos de diagnóstico se achar necessário.

[ ]:
```

```

5 Prática 5

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Considere os dados RestaurantVisitors.csv, que contém dados de visitantes de restaurantes, baseado em uma competição Kaggle . Os dados consideram o total de visitantes diários de quatro restaurantes localizados nos Estados Unidos, sujeitos aos feriados americanos. Para a variável exógena, utilizaremos os feriados, para verificar como eles afetam o movimento nos restaurantes. O conjunto de dados contém 478 dias de dados de restaurantes, além de 39 dias adicionais de dados de feriados para fins de previsão.

Ajuste um modelo SARIMA com uma variável exógena “holiday” usando o enfoque de modelos de espaço de estado para a variável “total”.

Faça a divisão da base em treino e teste e verifique as previsões obtidas.

Em seguida, faça a previsão para observações futuras com as informações de feriados disponíveis.

1. Carregue as bibliotecas necessárias.

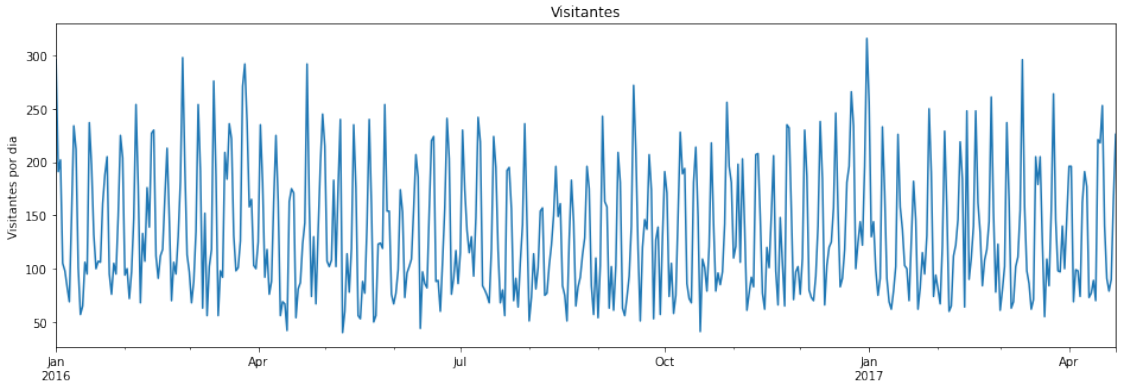
[74]:
```
import numpy as np
import pandas as pd
from scipy.stats import norm
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
import requests
from io import BytesIO

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from pmdarima import auto_arima


# Iniba warnings não prejudiciais
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

2. Faça a leitura dos dados. Exclua as observações faltantes em total, que correspondem às observações extras de feriados.

[75]:
```
# Leitura dos dados

df = pd.read_csv('../Data/RestaurantVisitors.
 ↪csv',index_col='date',parse_dates=True)
df.index.freq = 'D'
df.head()

df1 = df.dropna()
```

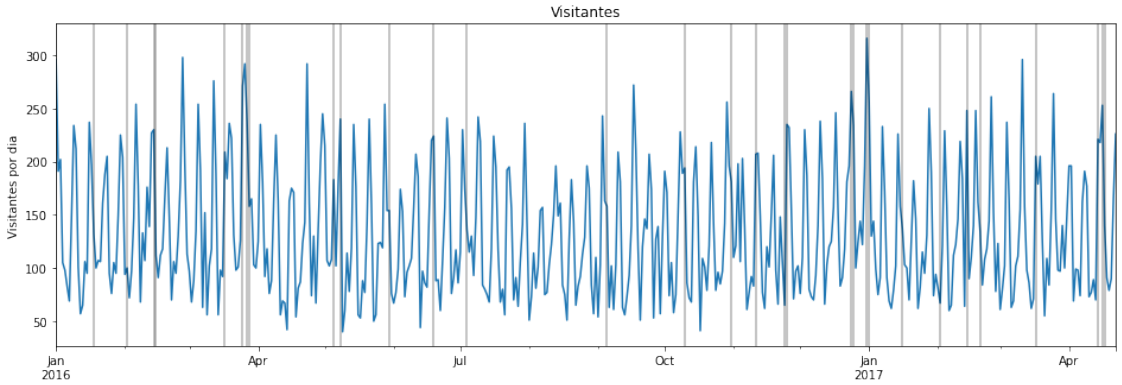
3. Faça a visualização dos dados.

[76]:
```
# Visualização dos dados

title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

ax = df1['total'].plot(figsize=(16,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



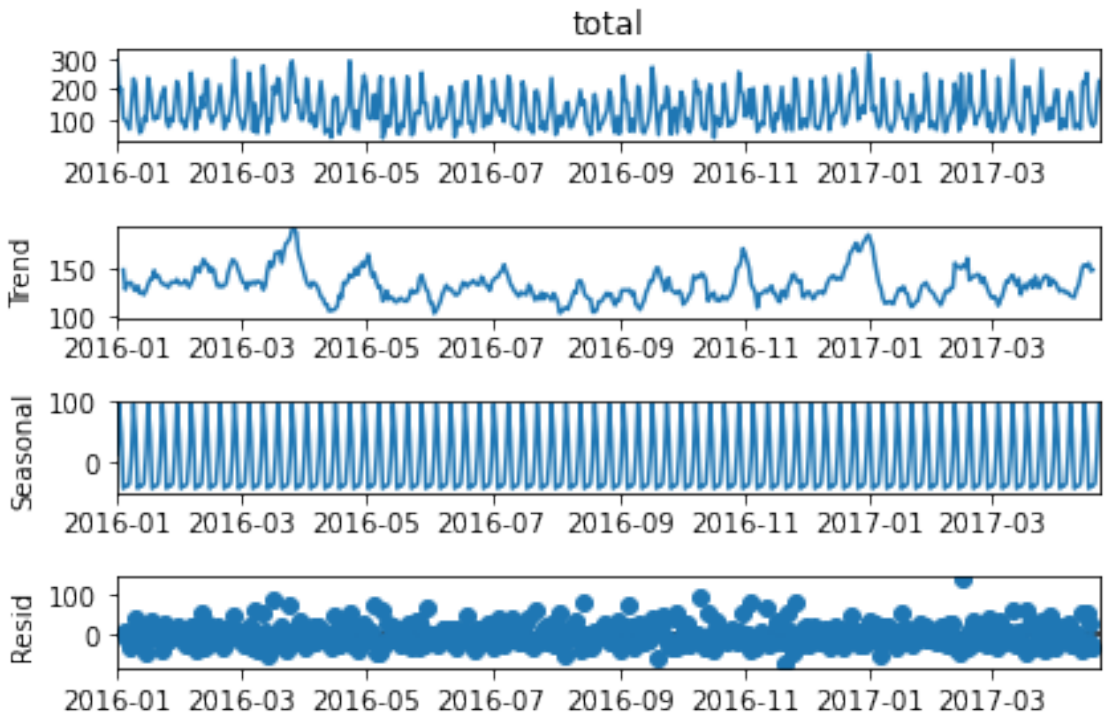
4. Marque os feriados com linhas verticais em cinza.

[77]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

ax = df1['total'].plot(figsize=(16,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in df1.query('holiday==1').index:        # for days where holiday == 1
    ax.axvline(x=x, color='k', alpha = 0.3);   # add a semi-transparent grey line
```



5. Faça uma decomposição da série em tendência e sazonalidade.

[78]:
```
decomposicao = seasonal_decompose(df1['total'])
decomposicao.plot();
```



6. Verifique a estacionariedade da série.

[79]:
```
# Testando estacionariedade
# fonte: https://machinelearningmastery.com/time-series-data-stationary-python/


from statsmodels.tsa.stattools import adfuller

result = adfuller(df1['total'], autolag='AIC')
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -5.592497
p-value: 0.000001
Critical Values:
        1%: -3.445
        5%: -2.868
        10%: -2.570
```

7. Escolha um modelo SARIMA permitindo termos com sazonalidade 7.

[80]:
```
stepwise_fit = auto_arima(df1['total'], start_p=0, start_q=0,
                          max_p=6, max_q=3, m=7,
                          seasonal=True,
                          trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
Fit ARIMA(0,0,0)x(1,0,1,7) [intercept=True]; AIC=4776.028, BIC=4792.706,
Time=0.908 seconds
Fit ARIMA(0,0,0)x(0,0,0,7) [intercept=True]; AIC=5269.484, BIC=5277.823,
Time=0.014 seconds
Fit ARIMA(1,0,0)x(1,0,0,7) [intercept=True]; AIC=4916.749, BIC=4933.428,
Time=0.552 seconds
Fit ARIMA(0,0,1)x(0,0,1,7) [intercept=True]; AIC=5049.644, BIC=5066.322,
Time=0.357 seconds
Fit ARIMA(0,0,0)x(0,0,0,7) [intercept=False]; AIC=6126.084, BIC=6130.254,
Time=0.009 seconds
Fit ARIMA(0,0,0)x(0,0,1,7) [intercept=True]; AIC=5093.130, BIC=5105.639,
Time=0.157 seconds
Fit ARIMA(0,0,0)x(1,0,0,7) [intercept=True]; AIC=4926.360, BIC=4938.869,
Time=0.353 seconds
Fit ARIMA(0,0,0)x(2,0,1,7) [intercept=True]; AIC=nan, BIC=nan, Time=nan seconds
Fit ARIMA(0,0,0)x(1,0,2,7) [intercept=True]; AIC=4961.615, BIC=4982.463,
Time=1.439 seconds
Fit ARIMA(0,0,0)x(0,0,2,7) [intercept=True]; AIC=5010.582, BIC=5027.260,
Time=0.555 seconds
Fit ARIMA(0,0,0)x(2,0,0,7) [intercept=True]; AIC=4859.646, BIC=4876.324,
Time=1.796 seconds
Fit ARIMA(0,0,0)x(2,0,2,7) [intercept=True]; AIC=5276.457, BIC=5301.474,
Time=1.885 seconds
Near non-invertible roots for order (0, 0, 0)(2, 0, 2, 7); setting score to inf
(at least one inverse root too close to the border of the unit circle: 1.000)
Fit ARIMA(1,0,0)x(1,0,1,7) [intercept=True]; AIC=4794.105, BIC=4814.953,
Time=0.947 seconds
Fit ARIMA(0,0,1)x(1,0,1,7) [intercept=True]; AIC=5166.461, BIC=5187.309,
Time=1.153 seconds
Near non-invertible roots for order (0, 0, 1)(1, 0, 1, 7); setting score to inf
(at least one inverse root too close to the border of the unit circle: 1.000)
Fit ARIMA(1,0,1)x(1,0,1,7) [intercept=True]; AIC=4808.217, BIC=4833.235,
Time=1.252 seconds
Near non-invertible roots for order (1, 0, 1)(1, 0, 1, 7); setting score to inf
(at least one inverse root too close to the border of the unit circle: 1.000)
Total fit time: 12.937 seconds
```

[80]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==========================================================================
=
Dep. Variable:                           y   No. Observations:
478
Model:             SARIMAX(1, 0, [1], 7)   Log Likelihood
-2384.014
Date:                   Sun, 20 Sep 2020   AIC
4776.028
Time:                           22:38:42   BIC
4792.706
Sample:                                 0   HQIC
4782.585
                                    - 478
Covariance Type:                      opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      4.6617      1.643      2.837      0.005       1.441       7.883
ar.S.L7        0.9650      0.012     78.149      0.000       0.941       0.989
ma.S.L7       -0.7927      0.046    -17.292      0.000      -0.883      -0.703
sigma2      1195.7389     69.144     17.294      0.000    1060.220    1331.258
==========================================================================
===
Ljung-Box (Q):                       72.99   Jarque-Bera (JB):
67.86
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.84   Skew:
0.78
Prob(H) (two-sided):                  0.26   Kurtosis:
3.98
==========================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

[ ]:

8. Separe uma base de treino e teste. A base de treino pode ter 80% das observações totais.

[81]:
```
train = df1.iloc[:382]
test = df1.iloc[382:]

endog = train['total']
exog = train['holiday']
```

9. Vamos ajustar um modelo SARIMA (0,0,0)x(1,0,1)7 para os dados de treino.

[82]:
```
# Veja https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.
 ↪sarimax.SARIMAX.html

modelo = sm.tsa.statespace.SARIMAX(endog, exog, order=(0,0,0),␣
 ↪seasonal_order=(1,0,1,7))
resultado = modelo.fit(disp=False)
print(resultado.summary())
```

```
                                SARIMAX Results
==========================================================================================
=
Dep. Variable:                          total   No. Observations:
382
Model:               SARIMAX(1, 0, [1], 7)   Log Likelihood
-1842.932
Date:                     Sun, 20 Sep 2020   AIC
3693.863
Time:                             22:38:42   BIC
3709.645
Sample:                         01-01-2016   HQIC
3700.124
                              - 01-16-2017
Covariance Type:                        opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
holiday       70.2791      5.088     13.813      0.000      60.307      80.251
ar.S.L7        0.9999   8.96e-05   1.12e+04      0.000       1.000       1.000
ma.S.L7       -0.9389      0.026    -36.608      0.000      -0.989      -0.889
sigma2       823.7055     54.844     15.019      0.000     716.213     931.198
==============================================================================
===
Ljung-Box (Q):                       63.86   Jarque-Bera (JB):
12.61
Prob(Q):                              0.01   Prob(JB):
0.00
Heteroskedasticity (H):               1.05   Skew:
0.26
Prob(H) (two-sided):                  0.77   Kurtosis:
3.73
```

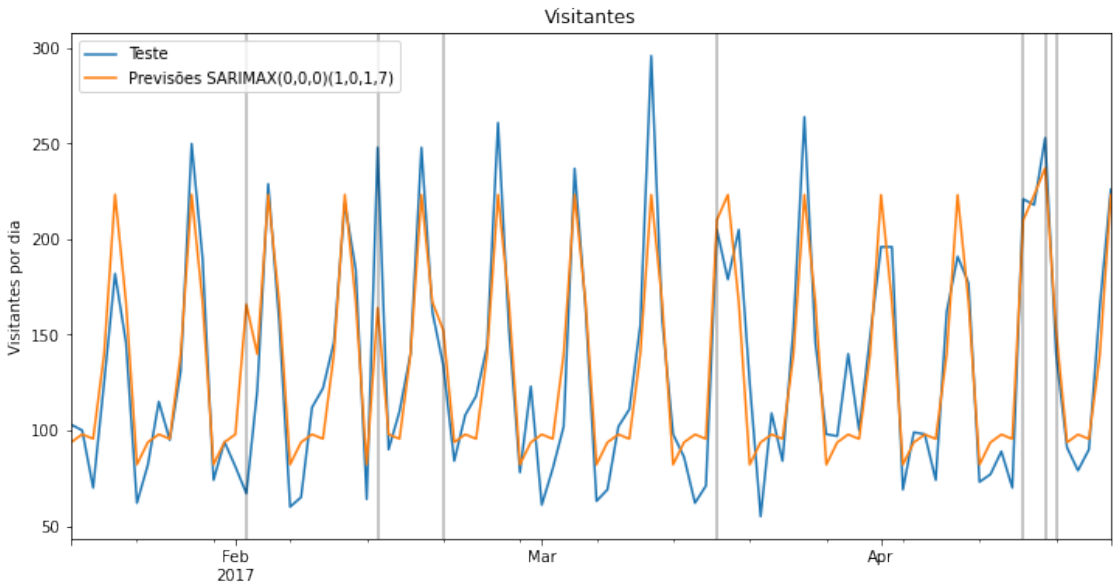
```
==========================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

10. Obtenha os valores preditos. É importante estabelecer em predict a variável exógena.

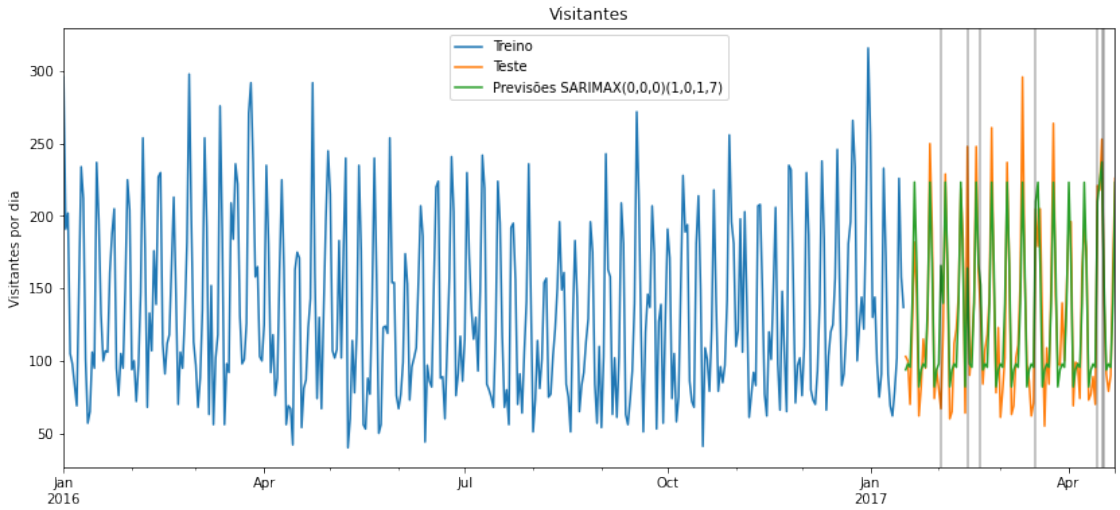
[83]:
```
start=len(train)
end=len(train)+len(test)-1
exog_forecast = test[['holiday']]  # Por que não usar apenas colchetes simples?
previsao = resultado.predict(start=start, end=end, exog=exog_forecast).
 ↪rename('Previsões SARIMAX(0,0,0)(1,0,1,7)')
```

11. Observe as previsões do modelo e compare com a base de teste.

[84]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''


ax = test['total'].plot(legend=True,figsize=(12,6),title=title, label='Teste')
previsao.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)

for x in test.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```



[85]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

train['total'].plot(legend=True,label='Treino')

ax = test['total'].plot(legend=True,figsize=(14,6),title=title, label='Teste')
previsao.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)

for x in test.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```

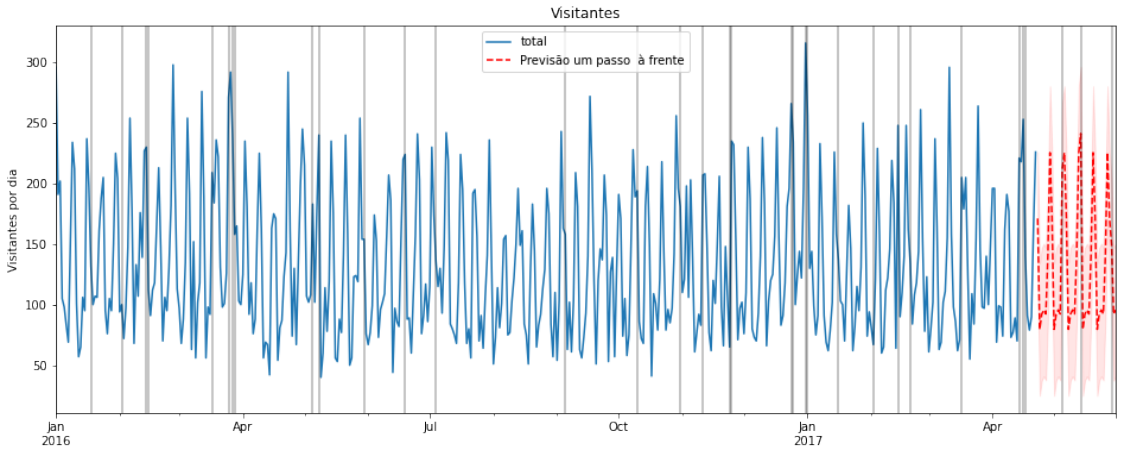


12. Faça as previsões para as próximas observações, com os feriados disponíveis na base. Nesse caso, não fará diferença a previsão passo à frente ou dinâmica.

[86]:
```
modelo = sm.tsa.statespace.
 ↪SARIMAX(df1['total'],exog=df1['holiday'],order=(0,0,0),seasonal_order=(1,0,1,7),enforce_inver
resultado = modelo.fit()
exog_forecast = df[478:][['holiday']]
```

[87]:
```
# Previsão um passo a frente. Sugestão: Use␣
 ↪get_prediction(len(df1),len(df1)+28,exog=exog_forecast)

previsao = resultado.get_prediction(len(df1),len(df1)+38,exog=exog_forecast)
previsao_ip = previsao.conf_int()
```

[88]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

ax = df1['total'].plot(legend=True,figsize=(16,6),title=title)

previsao.predicted_mean.plot(ax=ax, style='r--', label='Previsão um passo  à␣
 ↪frente', legend=True)
ip = previsao_ip.loc['2017-04-23':]
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='r', alpha=0.1)


ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in df.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```

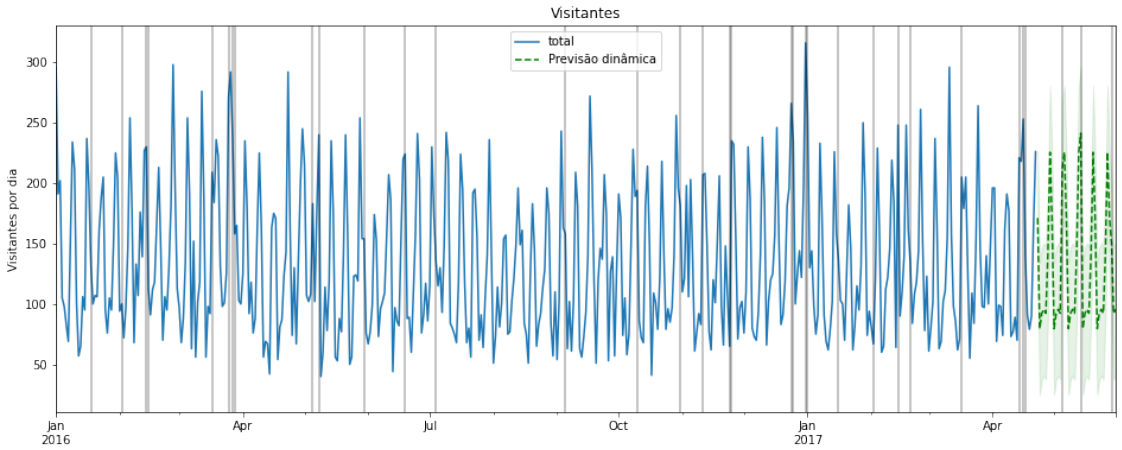


Para essa aplicação, escolher a previsão dinâmica dynamic=data_previsao_dinamica levará aos mesmos resultados na prática.

[89]:
```
# Previsão dinâmica

data_previsao_dinamica = '2017-04-23'

previsao_di = resultado.get_prediction(len(df1),len(df1)+38,␣
 ↪dynamic=data_previsao_dinamica, exog=exog_forecast)
previsao_di_ip = previsao_di.conf_int()
```

[90]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''
```

```
ax = df1['total'].plot(legend=True,figsize=(16,6),title=title)

previsao_di.predicted_mean.plot(ax=ax, style='g--', label='Previsão dinâmica',␣
 ↪legend=True)
ip = previsao_di_ip.loc['2017-04-23':]
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='g', alpha=0.1)

ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in df.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```



Prática extra:

Ajuste e interprete um modelo Bayesiano para a demanda de eletricidade disponível em

https://blog.tensorflow.org/2019/03/structural-time-series-modeling-in.html

[ ]:

6 Prática 6

Aprendizado Dinâmico

por **Cibele Russo** (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Nesta prática vamos comparar o Método Theta simples de Fiorucci et al. (2016) com o Método de Holt, visto na Aula 2, para previsões dos 30 próximos dias para os dados PETR4. Repita com janelas menores como 20 ou 10 dias.

1. Defina a função sThetaF como vista em aula. É ela que usaremos neste exercício.

[91]:
```
## Implementação

# Fonte: https://github.com/MinhDg00/theta/tree/master/src
# Baseado em Fiorucci et. al (2016) https://github.com/cran/forecTheta

# Implement standard Theta method based on Fiorucci et al. (2016)
# Reference: https://github.com/cran/forecTheta

import sys
import numpy as np
import pandas as pd
import statsmodels as sm
import warnings
from scipy.stats import norm
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from sklearn.linear_model import LinearRegression


def sThetaF(y, s_period = 1, h = 10, s = None):
        """
        @param y : array-like time series data
        @param s_period : the no. of observations before seasonal pattern repeats
        @param h : number of period for forcasting
        @s : additive or multiplicative
        """
        fcast = {} # store result
        n = y.index.size
        x = y.copy()
        m = s_period
        time_y = np.array(np.arange(n))/m + 1
        time_fc = time_y[n-1] + np.array(np.arange(1,h+1))/m

        s_type = 'multiplicative'
        if s is not None:
                if s == 'additive':
                        s = True
                        s_type = 'additive'

        # Seasonality Test & Decomposition
        if s is not None and m >= 4:
                r = (acf(x, nlags = m+1))[1:]
                clim = 1.64/sqrt(n) * np.sqrt(np.cumsum([1, 2 * np.square(r)]))
                s = abs(r[m-1]) > clim[m-1]
```

```
        else:
                if not s:
                        s = False


        if s:
                decomp = seasonal_decompose(x, model = s_type)
                if s_type == 'additive' or (s_type -- 'multiplicative' and␣
↪any(decomp < 0.01)):
                        s_type = 'additive'
                        decomp = seasonal_decompose(x, model = 'additive').
↪seasonal
                        x = x - decomp
                else:
                        x = x/decomp


        ## Find Theta Line
        model = LinearRegression().fit(time_y.reshape(-1,1), x)
        fcast['mean'] = model.intercept_ + model.coef_ * time_fc


        return fcast
```

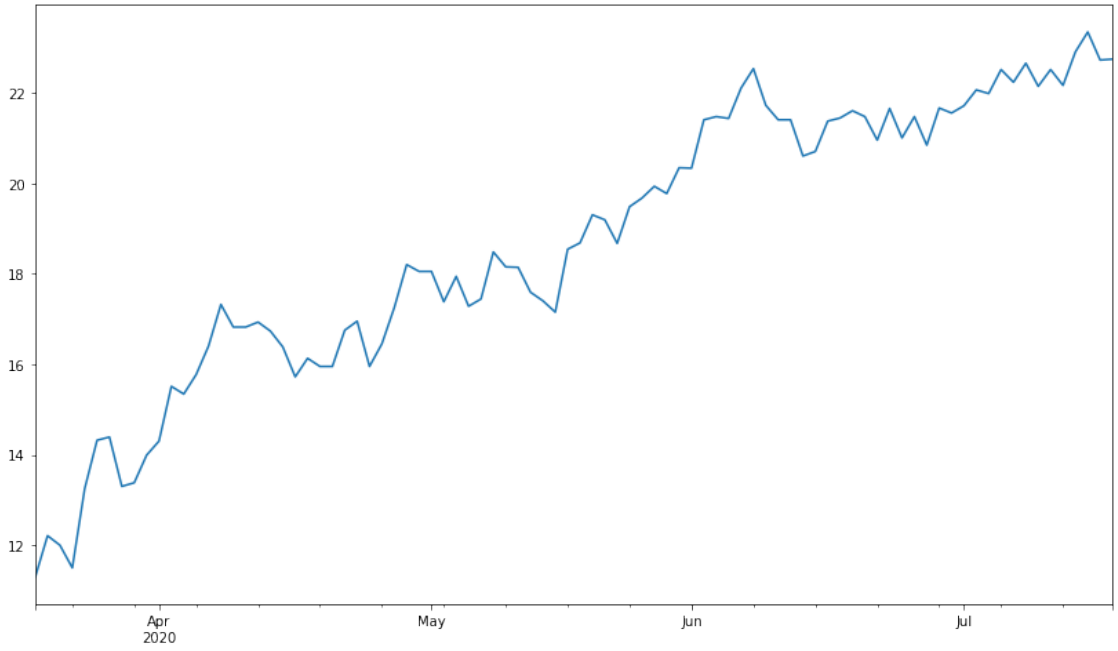
2. Carregue as bibliotecas e faça a leitura dos dados PETR4.

[92]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

# PETR4 - Leitura dos dados
df = pd.read_csv(f'{pkgdir}/PETR4.csv', index_col='Date', parse_dates=True)


idx = pd.date_range(start=df.index.min(), end=df.index.max(), freq='B')
df = df.reindex(idx)
df.fillna(method='ffill', inplace=True)
```

3. Faça um gráfico da série.

[93]:
```
plt.rcParams['figure.figsize'] = [14,8]


df['Close'].plot()
```

[93]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb324a23990>



4. Defina bases de treino e teste, deixando 30 observações para a base de teste.

[94]:
```
treino = df.iloc[:-30]
teste = df.iloc[-30:]
```

5. Prepare os dados para utilizar a função SThetaF.

[95]:
```
data = pd.Series(treino['Close'], df.index)
```

6. Obtenha as previsões pelo Método Theta de Fiorucci (2016) com a função sThetaF.

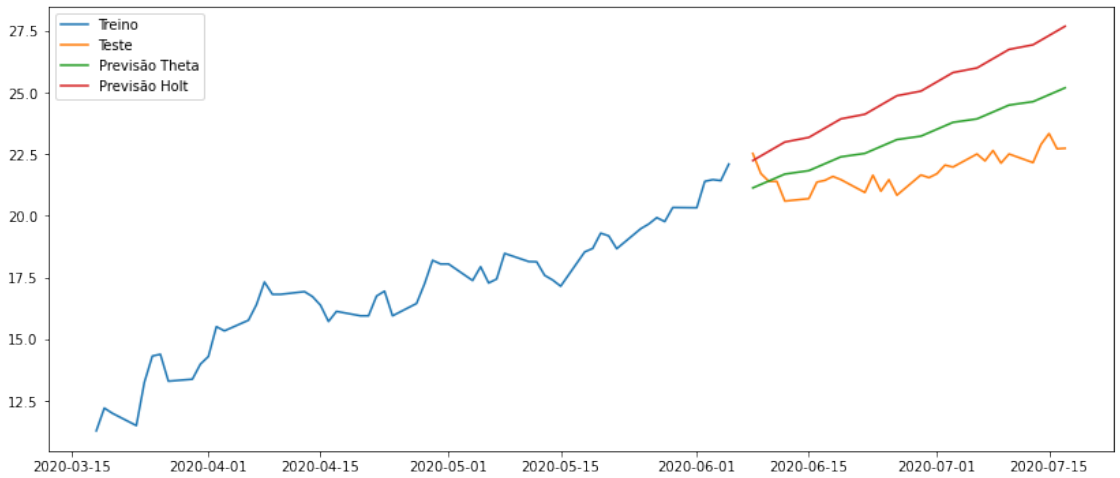
[96]:
```
from datetime import timedelta

data = pd.Series(treino['Close'], treino.index)

modelo = pd.DataFrame()

modelo['mean'] = sThetaF(data, h=30)['mean']

modelo.index = pd.date_range(start=treino.index.
 ↪max()+timedelta(days=1),periods=30,freq='B')
```

7. Aplique o Método de Holt.

[97]:
```
# Método de Holt

from statsmodels.tsa.api import ExponentialSmoothing

modelo_H =  ExponentialSmoothing(treino['Close'], trend='add');

ajustado = modelo_H.fit();

df['Holt'] = ajustado.fittedvalues.shift(-1);

predito_H = ajustado.forecast(30).rename('Previsão Holt')
```

8. Represente graficamente os resultados, com as bases de treino e teste.

[98]:
```
mean = modelo['mean']

plt.figure(figsize = (14,6))

plt.plot(treino['Close'], label='Treino')
plt.plot(teste['Close'], label='Teste')
plt.plot(mean, label='Previsão Theta')
plt.plot(predito_H, label='Previsão Holt')

plt.legend()
plt.show()
```



9. Utilize o EQM para comparar as previsões.

[99]:
```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(teste['Close'], modelo['mean'])
```

```
print(f'EQM Theta: {error:11.10}')


error = mean_squared_error(teste['Close'], predito_H)
print(f'EQM Holt: {error:11.10}')
```

```
EQM Theta: 2.531271192
EQM Holt: 11.34486653
```

10. Mude o tamanho das bases de treino e teste, para por exemplo 10 ou 20 observações na amostra de teste. Os resultados são similares?

11. Repita os procedimentos para obter previsões para dados futuros, ou seja, utilizando todos os dados disponíveis e fazendo previsões para os próximos 10 dias.

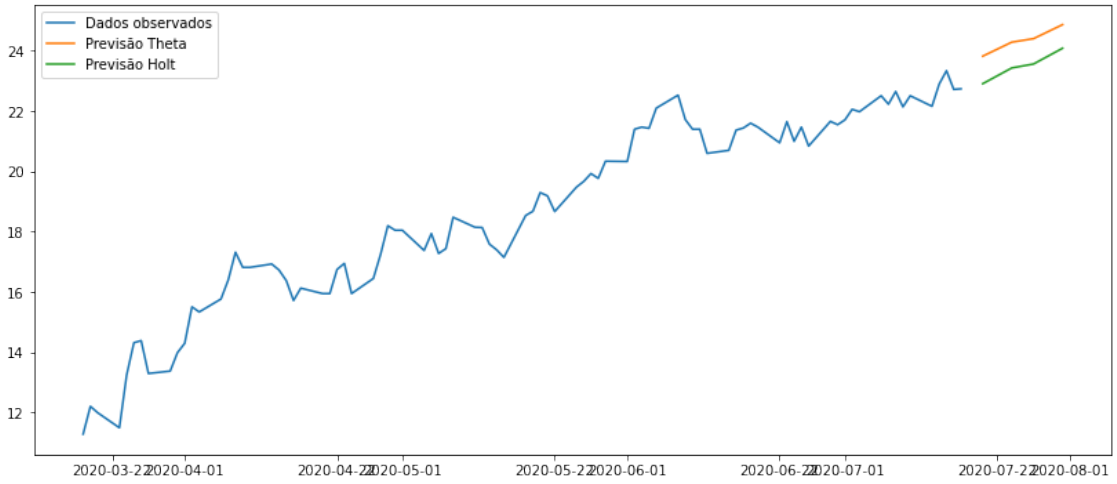
[100]:
```
data = pd.Series(df['Close'], df.index)
```

[101]:
```
from datetime import timedelta

data = pd.Series(df['Close'], df.index)

modelo = pd.DataFrame()

modelo['mean'] = sThetaF(data,  h = 10)['mean']

modelo.index = pd.date_range(start=df.index.
 ↪max()+timedelta(days=1),periods=10,freq='B')
```

[102]:
```
# Método de Holt

from statsmodels.tsa.api import ExponentialSmoothing

modelo_H =  ExponentialSmoothing(df['Close'], trend='add');

ajustado = modelo_H.fit();

df['Holt'] = ajustado.fittedvalues.shift(-1);

predito_H = ajustado.forecast(10).rename('Previsão Holt')
```

[103]:
```
mean = modelo['mean']

plt.figure(figsize = (14,6))

plt.plot( df['Close'], label='Dados observados')
plt.plot( modelo['mean'], label='Previsão Theta')
plt.plot(predito_H, label='Previsão Holt')
```

```
plt.legend()
plt.show()
```



[ ]:

[ ]:

7 Prática 7

Aprendizado Dinâmico

por Cibele Russo (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Nesta prática vamos considerar redes dinâmicas para modelar a temperatura global dos dados em globaltemp.

1.Faça a leitura das bibliotecas.

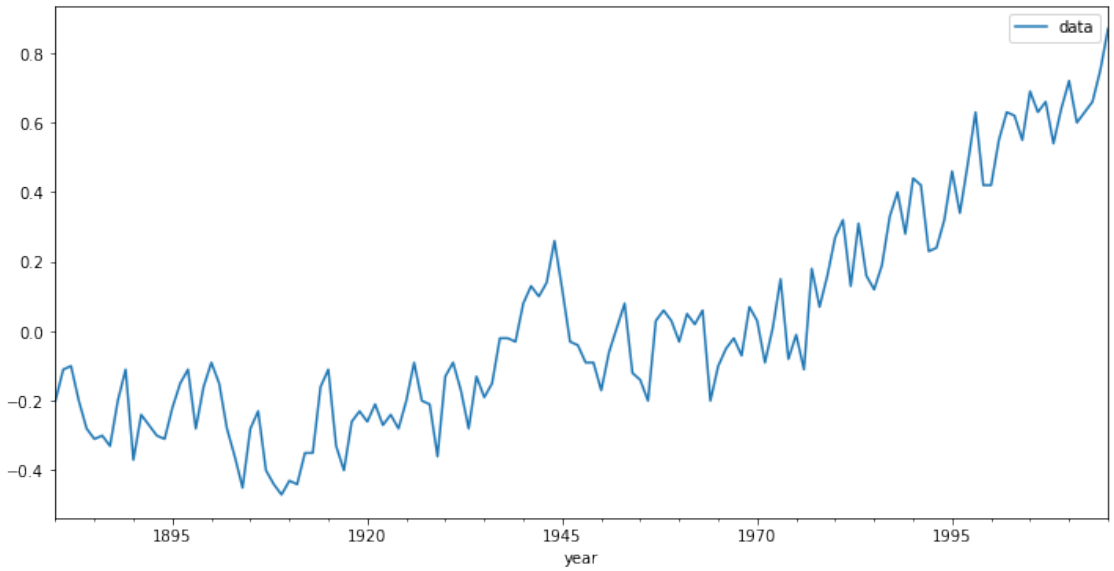
[104]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

2. Leia os dados do arquivo globaltemp.csv.

[105]:
```
# Temperatura global

# Diferenças na média de temperatura global.
# Fonte: https://github.com/mjuez/pytsdatasets/
```

```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'


df = pd.read_csv(f'{pkgdir}/globaltemp.csv', index_col=0,
                         parse_dates=True)
df.index = df.index
df.plot(figsize=(12,6))
```

[105]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb324e76c10>



[106]:
```
df.head()
```

[106]:
```
            data
year
1880-01-01 -0.20
1881-01-01 -0.11
1882-01-01 -0.10
1883-01-01 -0.20
1884-01-01 -0.28
```

7.1

3. Divida a base em treino e teste, deixando 14 dias para a previsão.

[107]:
```
len(df)
```

[107]: 136

[108]:
```
len(df)-14
```

[108]:
```
122
```

[109]:
```
train = df.iloc[:122]
test = df.iloc[122:]
```

[110]:
```
train
```

[110]:
```
            data
year
1880-01-01 -0.20
1881-01-01 -0.11
1882-01-01 -0.10
1883-01-01 -0.20
1884-01-01 -0.28
...          ...
1997-01-01  0.48
1998-01-01  0.63
1999-01-01  0.42
2000-01-01  0.42
2001-01-01  0.55

[122 rows x 1 columns]
```

4. Padronize os dados para a modelagem.

[111]:
```
from sklearn.preprocessing import MinMaxScaler
```

[112]:
```
scaler = MinMaxScaler()
```

[113]:
```
scaler.fit(train)
```

[113]:
```
MinMaxScaler()
```

[114]:
```
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

5. Considere o gerador de séries temporais, com variados valores para os parâmetros length e batch_size.

[115]:
```
from keras.preprocessing.sequence import TimeseriesGenerator
```

[116]:
```
# defina o gerador
n_input = 2
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input,␣
 ↪batch_size=1)
```

[117]:
```
len(scaled_train)
```

[117]: 122

[118]:
```
len(generator) # n_input = 2
```

[118]: 120

[119]:
```
# Qual é a aparência do primeiro lote?
X,y = generator[0]
```

[120]:
```
print(f'Dado o array: \n{X.flatten()}');
print(f'Previsão: \n {y}');
```

```
Dado o array:
[0.24545455 0.32727273]
Previsão:
 [[0.33636364]]
```

6. Carregue as bibliotecas do keras para as redes dinâmicas.

[121]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

7. Defina os lotes pra o processo iterativo.

[122]:
```
# Vamos redefinir lotes de tamanho 21 para o procedimento iterativo
# Veja mais informações sobre o tamanho do lote http://deeplearningbook.com.br/
 ↪o-efeito-do-batch-size-no-treinamento-de-redes-neurais-artificiais/

n_input = 7
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input,␣
 ↪batch_size=1)
```

8. Defina o modelo. Ele pode ter uma camada LSTM e uma camada Dense. Teste alternativas.

[123]:
```
# Defina o modelo
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

[124]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
lstm (LSTM)                  (None, 100)               40800
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

9. Faça o ajuste do modelo e observe a função de perda.

[125]:
```
# Ajuste do modelo

model.fit_generator(generator,epochs=100)
```

```
WARNING:tensorflow:From <ipython-input-125-5e7daf52724a>:3: Model.fit_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0430
Epoch 2/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0119
Epoch 3/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0132
Epoch 4/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0123
Epoch 5/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0119
Epoch 6/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0113
Epoch 7/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0102
Epoch 8/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0114
Epoch 9/100
115/115 [==============================] - 1s 5ms/step - loss: 0.0123
Epoch 10/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0111
Epoch 11/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0115
Epoch 12/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0112
Epoch 13/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0104
Epoch 14/100
```

```
115/115 [==============================] - 0s 4ms/step - loss: 0.0109
Epoch 15/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0107
Epoch 16/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0103
Epoch 17/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0102
Epoch 18/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0094
Epoch 19/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0102
Epoch 20/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 21/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0101
Epoch 22/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0103
Epoch 23/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0102
Epoch 24/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0097
Epoch 25/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0102
Epoch 26/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0105
Epoch 27/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0094
Epoch 28/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0095
Epoch 29/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0099
Epoch 30/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0094
Epoch 31/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0099
Epoch 32/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0091
Epoch 33/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0092
Epoch 34/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0097
Epoch 35/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0098
Epoch 36/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0099
Epoch 37/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 38/100
```

```
115/115 [==============================] - 0s 3ms/step - loss: 0.0094
Epoch 39/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0092
Epoch 40/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0093
Epoch 41/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0093
Epoch 42/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0094
Epoch 43/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0086
Epoch 44/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0090
Epoch 45/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 46/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0087
Epoch 47/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0094
Epoch 48/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 49/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0086
Epoch 50/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 51/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0086
Epoch 52/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 53/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 54/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 55/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0091
Epoch 56/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0085
Epoch 57/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0089
Epoch 58/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0087
Epoch 59/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0086
Epoch 60/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0093
Epoch 61/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 62/100
```

```
115/115 [==============================] - 0s 3ms/step - loss: 0.0089
Epoch 63/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 64/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0083
Epoch 65/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0080
Epoch 66/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 67/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 68/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0086
Epoch 69/100
115/115 [==============================] - 1s 5ms/step - loss: 0.0082
Epoch 70/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0086
Epoch 71/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0082
Epoch 72/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0088
Epoch 73/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 74/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0081
Epoch 75/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 76/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 77/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0081
Epoch 78/100
115/115 [==============================] - 0s 4ms/step - loss: 0.0082
Epoch 79/100
115/115 [==============================] - 1s 5ms/step - loss: 0.0085
Epoch 80/100
115/115 [==============================] - 1s 5ms/step - loss: 0.0088
Epoch 81/100
115/115 [==============================] - 1s 5ms/step - loss: 0.0083
Epoch 82/100
115/115 [==============================] - 1s 4ms/step - loss: 0.0084
Epoch 83/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0080
Epoch 84/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0082
Epoch 85/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0086
Epoch 86/100
```

```
115/115 [==============================] - 0s 3ms/step - loss: 0.0083
Epoch 87/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 88/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 89/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0083
Epoch 90/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 91/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0080
Epoch 92/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0080
Epoch 93/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0081
Epoch 94/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0082
Epoch 95/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0084
Epoch 96/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0079
Epoch 97/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0083
Epoch 98/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0079
Epoch 99/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0082
Epoch 100/100
115/115 [==============================] - 0s 3ms/step - loss: 0.0082
```

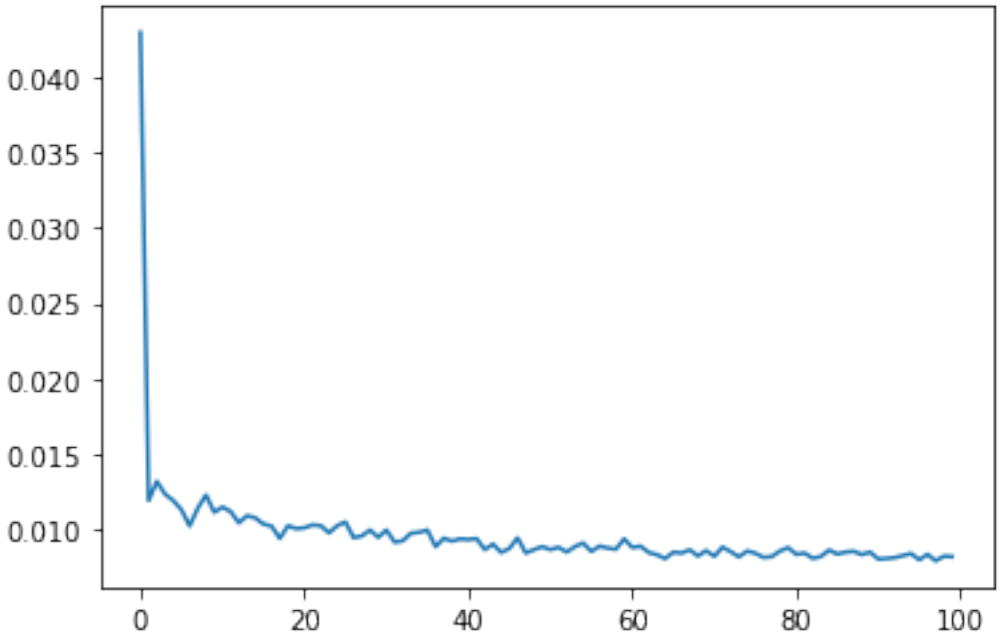
[125]:
```
<tensorflow.python.keras.callbacks.History at 0x7fb2f4468610>
```

[126]:
```
model.history.history.keys()
```

[126]:
```
dict_keys(['loss'])
```

[127]:
```
loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

[127]:
```
[<matplotlib.lines.Line2D at 0x7fb2f42aa450>]
```



10. Faça a previsão.

[128]:
```
# Vejamos passo a passo como é feita a previsão, a princípio para a próxima␣
 ↪observação usando o tamanho do lote igual a 7

first_eval_batch = scaled_train[-7:]
```

[129]:
```
first_eval_batch
```

[129]:
```
array([[0.84545455],
       [0.73636364],
       [0.86363636],
       [1.        ],
       [0.80909091],
       [0.80909091],
       [0.92727273]])
```

[130]:
```
# Agora vamos considerar as previsões para as próximas 21 observações e comparar␣
 ↪com a base de teste

test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(test)):
```

```
    # obter a previsão de tempo 1 antecipadamente ([0] é para pegar apenas o
↪número em vez de [array])
    current_pred = model.predict(current_batch)[0]

    # predição
    test_predictions.append(current_pred)

    # atualize a rodada para agora incluir a previsão e descartar o primeiro
↪valor
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

[131]:
```
test_predictions
```

[131]:
```
[array([1.0574188], dtype=float32),
 array([0.9999189], dtype=float32),
 array([1.1157258], dtype=float32),
 array([1.3270055], dtype=float32),
 array([1.1986059], dtype=float32),
 array([1.1832825], dtype=float32),
 array([1.3401037], dtype=float32),
 array([1.5617845], dtype=float32),
 array([1.5877235], dtype=float32),
 array([1.8407016], dtype=float32),
 array([2.2273872], dtype=float32),
 array([2.192564], dtype=float32),
 array([2.2570548], dtype=float32),
 array([2.6877124], dtype=float32)]
```

[132]:
```
scaled_test
```

[132]:
```
array([[1.        ],
       [0.99090909],
       [0.92727273],
       [1.05454545],
       [1.        ],
       [1.02727273],
       [0.91818182],
       [1.00909091],
       [1.08181818],
       [0.97272727],
       [1.        ],
       [1.02727273],
       [1.10909091],
       [1.21818182]])
```

11. Retorne da padronização.

[133]:
```
true_predictions = scaler.inverse_transform(test_predictions)
```

[134]:
```
true_predictions
```

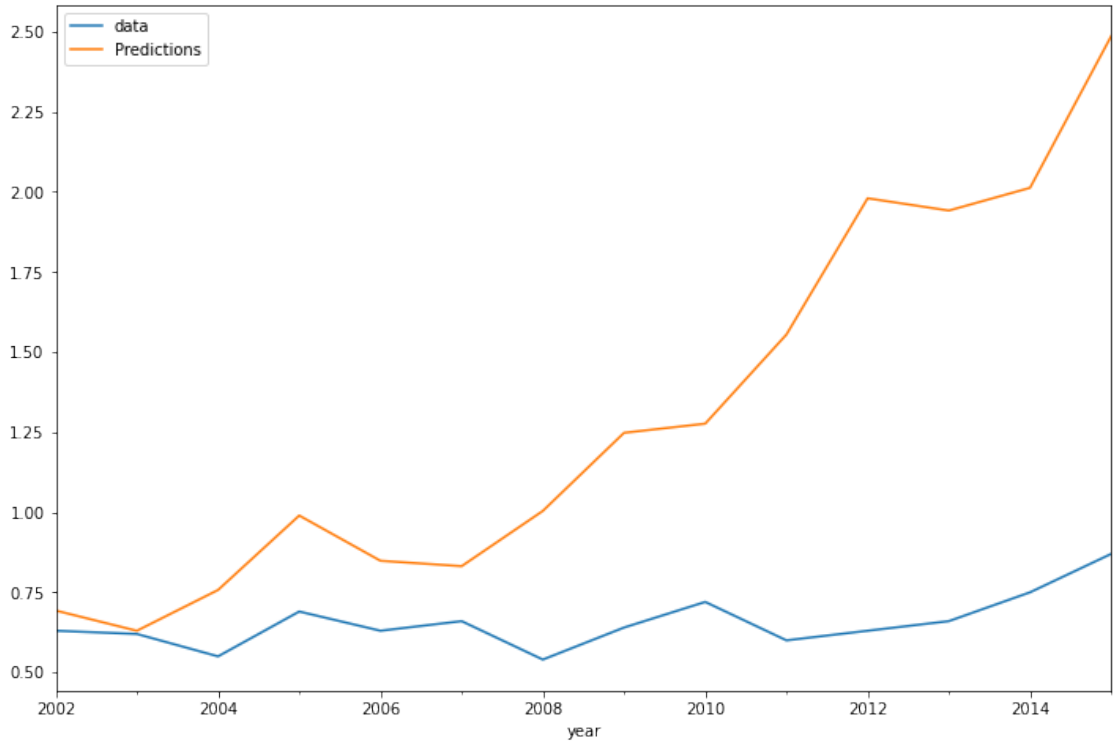
[134]:
```
array([[0.69316071],
       [0.62991077],
       [0.75729833],
       [0.98970606],
       [0.84846648],
       [0.83161074],
       [1.00411412],
       [1.24796296],
       [1.27649584],
       [1.55477174],
       [1.98012591],
       [1.94182041],
       [2.01276029],
       [2.48648367]])
```

[135]:
```
# Possivelmente encontraremos warnings aqui
test['Predictions'] = true_predictions
```

12. Visualize os resultados, comparando as previsões com a base de teste.

[136]:
```
test.plot(figsize=(12,8))
```

[136]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb324d77490>
```



8 Prática 8

Aprendizado Dinâmico

por Cibele Russo (ICMC/USP - São Carlos SP)

MBA em Ciências de Dados

Nesta prática vamos considerar um modelo de sobrevivência com covariáveis, especificamente um modelo de Cox. Fonte: https://www.kaggle.com/bryanb/survival-analysis-with-cox-model-implementation

Deseja-se modelar o tempo até a morte de pacientes de câncer de mama em até 5 anos após uma cirurgia para a retirada de nódulos. Os dados são descritos no arquivo haberman.csv.

Descrição de dados

- Age: Idade do paciente no momento da operação;
- Operation_year: ano de operação do paciente (ano - 19XX);
- Nb_pos_detected: Número de nódulos axilares positivos detectados;
- Surv: Status de sobrevivência (atributo de classe) 1 = o paciente sobreviveu 5 anos ou mais 2 = o paciente morreu dentro de 5 anos;

Desenvolva um estudo com modelo de sobrevivência incluindo como covariáveis o ano da operação e o número de nódulos axilar positivos detectados.

1.Faça a leitura das bibliotecas.

[137]:
```
import numpy as np
import pandas as pd
import pandas_profiling
import matplotlib.pyplot as plt

from lifelines import CoxPHFitter
from lifelines import KaplanMeierFitter

%matplotlib inline
```

2. Leia os dados do arquivo haberman.csv.

[138]:
```
pkgdir = '/home/cibele/CibelePython/AprendizadoDinamico/Data'

data = pd.read_csv(f'{pkgdir}/haberman.csv', names =␣
 ↪['Age','Operation_year','Nb_pos_detected','Surv'])
data.head()
```

[138]:

	Age	Operation_year	Nb_pos_detected	Surv
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

3. Use o pandas_profiling para produzir um relatório de análises exploratórias

Comando:

pandas_profiling.ProfileReport(data)

[139]:
```
pandas_profiling.ProfileReport(data)
```

```
HBox(children=(FloatProgress(value=0.0, description='Summarize dataset', max=18.0, style=Progres
```

```
HBox(children=(FloatProgress(value=0.0, description='Generate report structure', max=1.0, style=
```

```
HBox(children=(FloatProgress(value=0.0, description='Render HTML', max=1.0, style=ProgressStyle(
```

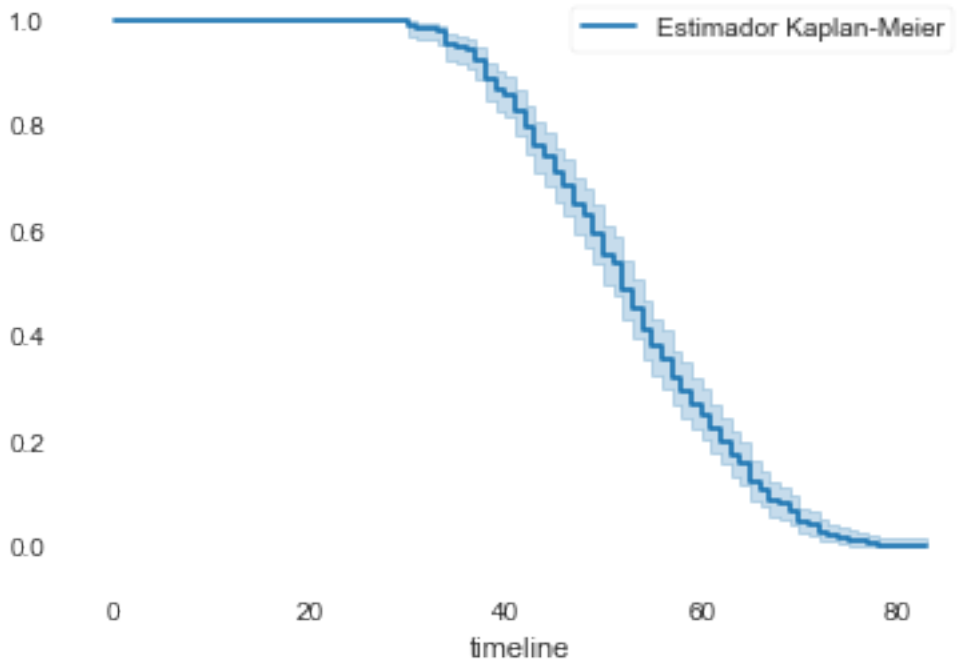
```
<IPython.core.display.HTML object>
```

[139]:

4. Estime a curva de sobrevivência com o estimador de Kaplan-Meier

[140]:
```
T = data.Age
E = data.Surv

km = KaplanMeierFitter()

km.fit(T, E,label='Estimador Kaplan-Meier')

km.plot(ci_show=True)
```

[140]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2959f7650>
```

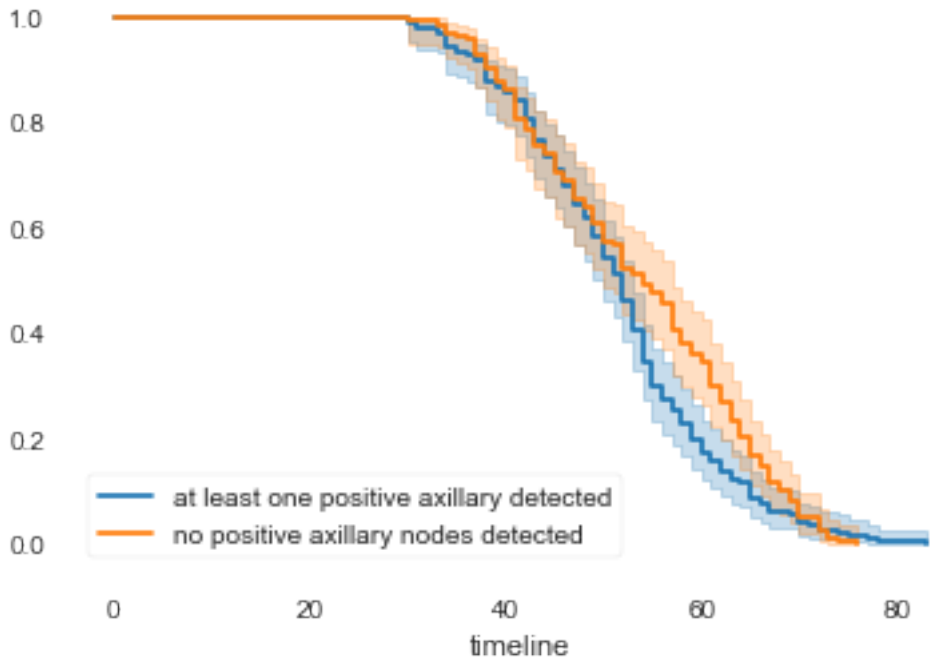


5. Utilize o estimador de Kaplan-Meier com dois grupos, um deles com pelo menos um nódulo axilar.

[141]:
```
km_2 = KaplanMeierFitter()

## Uma possibilidade: crie 2 coortes: com pelo menos um axilar positivo␣
 ↪detectado e um sem nenhum detectado

groups = data['Nb_pos_detected']
i1 = (groups >= 1)
```

```
i2 = (groups < 1)


## ajuste o modelo para a 1ª coorte
km_2.fit(T[i1], E[i1], label='at least one positive axillary detected')
a1 = km_2.plot()

## ajuste o modelo para a 2ª coorte
km_2.fit(T[i2], E[i2], label='no positive axillary nodes detected')
km_2.plot(ax=a1)
```

[141]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb295a79990>



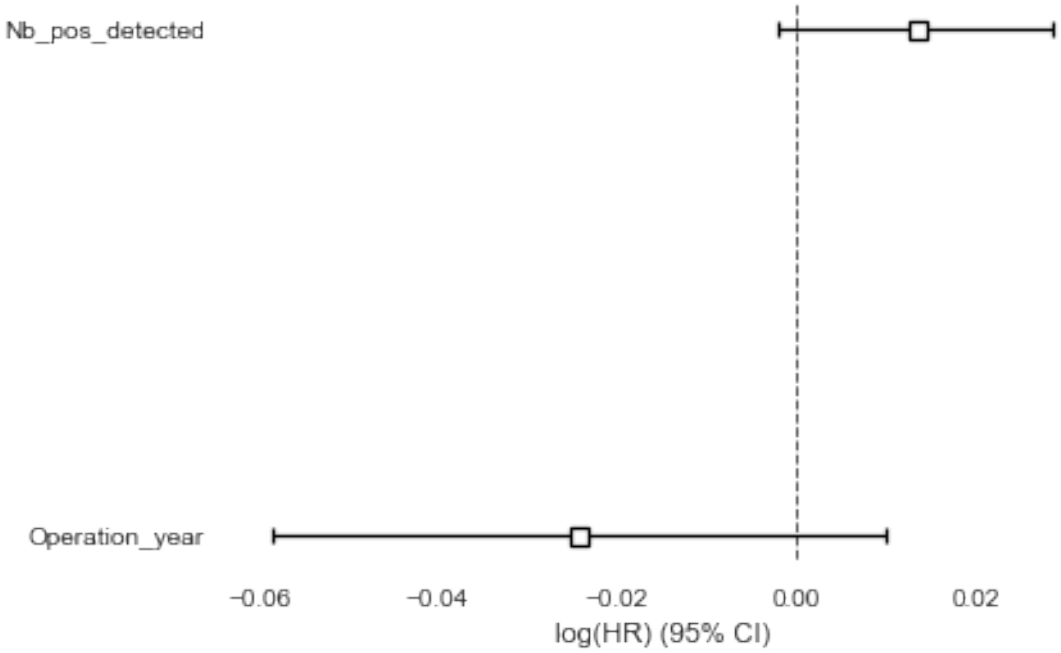
6. Ajuste um modelo de Cox com covariáveis Operatio_year e Nb_pos_detected. Interprete os resultados

[142]:
```
# Crie o modelo
cph = CoxPHFitter()

# Ajuste o modelo
cph.fit(data, 'Age', event_col='Surv')

# Verifique os resultados
cph.print_summary()
```

covariate	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp
Operation_year	-0.02	0.98	0.02	-0.06	0.01	0.94	
Nb_pos_detected	0.01	1.01	0.01	-0.00	0.03	1.00	

[143]:
```
cph.plot()
```

[143]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2959b5dd0>
```



7. Selecione três pacientes de forma aleatória para fazer a previsão da curva de sobrevivência para cada um deles

[144]:
```
patients = [4,125,211]

rows_selected = data.iloc[patients, 1:3]
rows_selected
```

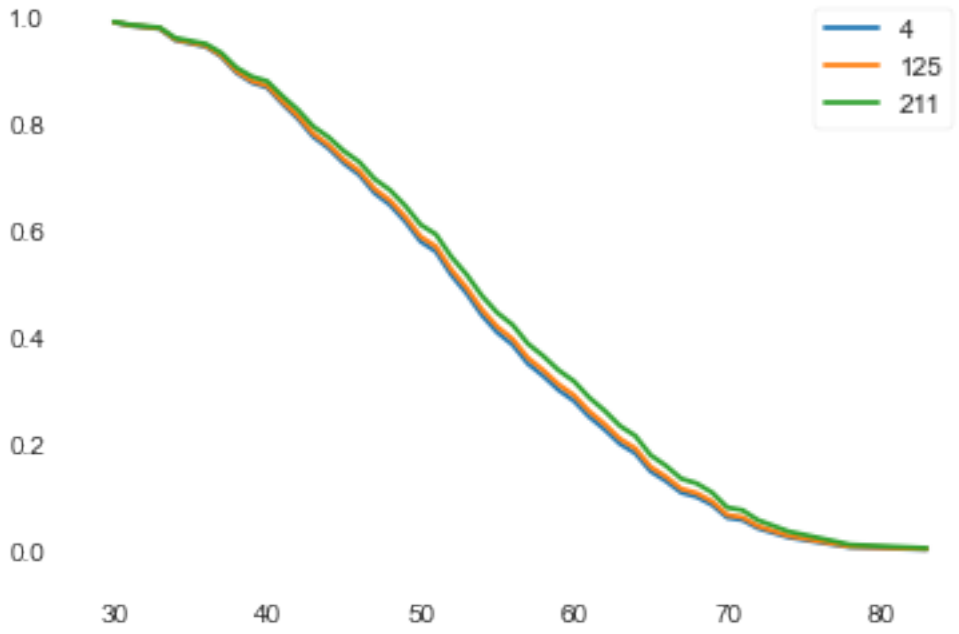
[144]:

	Operation_year	Nb_pos_detected
4	65	4
125	64	0
211	67	0

8. Faça a previsão da curva de sobrevivência para cada um dos pacientes selecionados acima

[145]:
```
cph.predict_survival_function(rows_selected).plot()
```

[145]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb2959b5650>



[ ]: