Prática5_Respostas

August 13, 2020

1 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.

[1]:
```
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")
```

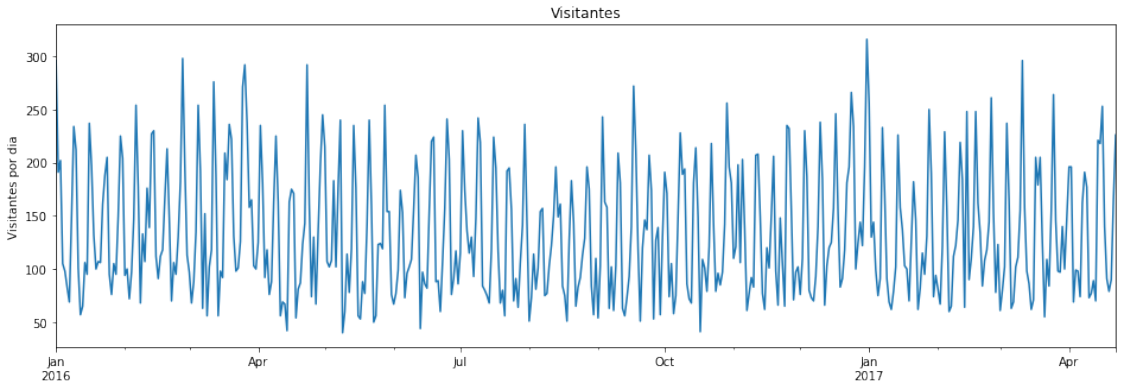
```
%matplotlib inline
```

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

[2]:
```
# 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.

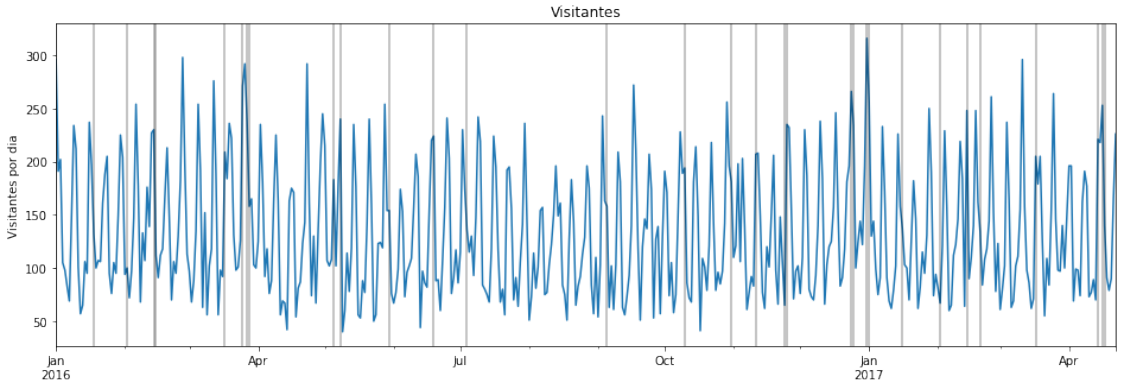
[3]:
```
# Visualização dos dados

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.

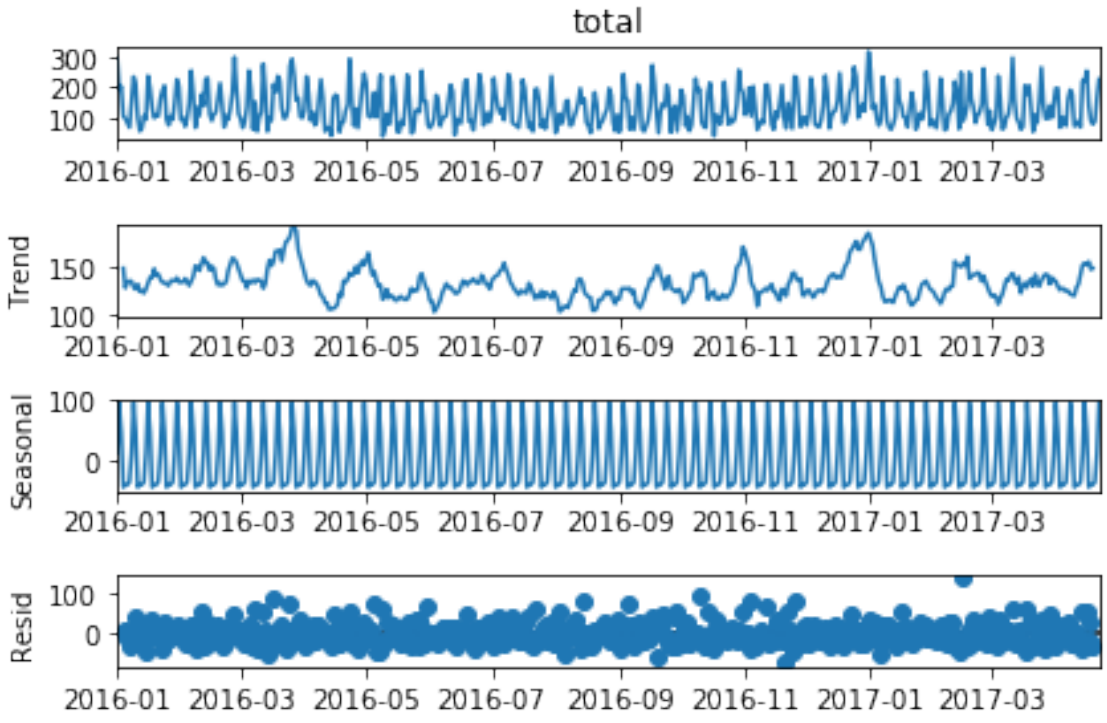
[4]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

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.

[5]:
```
decomposicao = seasonal_decompose(df1['total'])
decomposicao.plot();
```



6. Verifique a estacionariedade da série.

[6]:
```
# 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.

[7]:
```
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=4773.585, BIC=4790.263,
Time=0.855 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=0.532 seconds
Fit ARIMA(0,0,1)x(0,0,1,7) [intercept=True]; AIC=5049.644, BIC=5066.322,
Time=0.360 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=0.152 seconds
Fit ARIMA(0,0,0)x(1,0,0,7) [intercept=True]; AIC=4926.360, BIC=4938.869,
```

```
Time=0.347 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=4908.769, BIC=4929.617,
Time=1.426 seconds
Fit ARIMA(0,0,0)x(0,0,2,7) [intercept=True]; AIC=5010.582, BIC=5027.260,
Time=0.586 seconds
Fit ARIMA(0,0,0)x(2,0,0,7) [intercept=True]; AIC=4859.639, BIC=4876.318,
Time=2.035 seconds
Fit ARIMA(0,0,0)x(2,0,2,7) [intercept=True]; AIC=5275.991, BIC=5301.009,
Time=1.794 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=4830.264, BIC=4851.112,
Time=0.958 seconds
Fit ARIMA(0,0,1)x(1,0,1,7) [intercept=True]; AIC=5166.454, BIC=5187.302,
Time=1.017 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=4789.036, BIC=4814.054,
Time=1.295 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: 12.962 seconds
```

[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                SARIMAX Results
==========================================================================
=
Dep. Variable:                           y   No. Observations:
478
Model:             SARIMAX(1, 0, [1], 7)   Log Likelihood
-2382.792
Date:                   Thu, 13 Aug 2020   AIC
4773.585
Time:                           00:05:06   BIC
4790.263
Sample:                                 0   HQIC
4780.142
                                    - 478
Covariance Type:                      opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
intercept      4.6078      1.729      2.665      0.008       1.219       7.997
ar.S.L7        0.9648      0.013     77.093      0.000       0.940       0.989
ma.S.L7       -0.7134      0.052    -13.741      0.000      -0.815      -0.612
```

```
sigma2      1274.9763     78.549     16.232      0.000    1121.024    1428.929
==============================================================================
===
Ljung-Box (Q):                       68.31   Jarque-Bera (JB):
56.29
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.87   Skew:
0.69
Prob(H) (two-sided):                  0.39   Kurtosis:
3.97
==============================================================================
===

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.

[8]:
```
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.

[9]:
```
# 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:                   Thu, 13 Aug 2020   AIC
3693.863
```

```
Time:                           00:05:06   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       70.2815      5.088     13.814      0.000      60.310      80.253
ar.S.L7        0.9999   8.96e-05   1.12e+04      0.000       1.000       1.000
ma.S.L7       -0.9389      0.026    -36.605      0.000      -0.989      -0.889
sigma2       823.6867     54.841     15.019      0.000     716.200     931.174
===================================================================================
===
Ljung-Box (Q):                       63.86   Jarque-Bera (JB):
12.62
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).
```

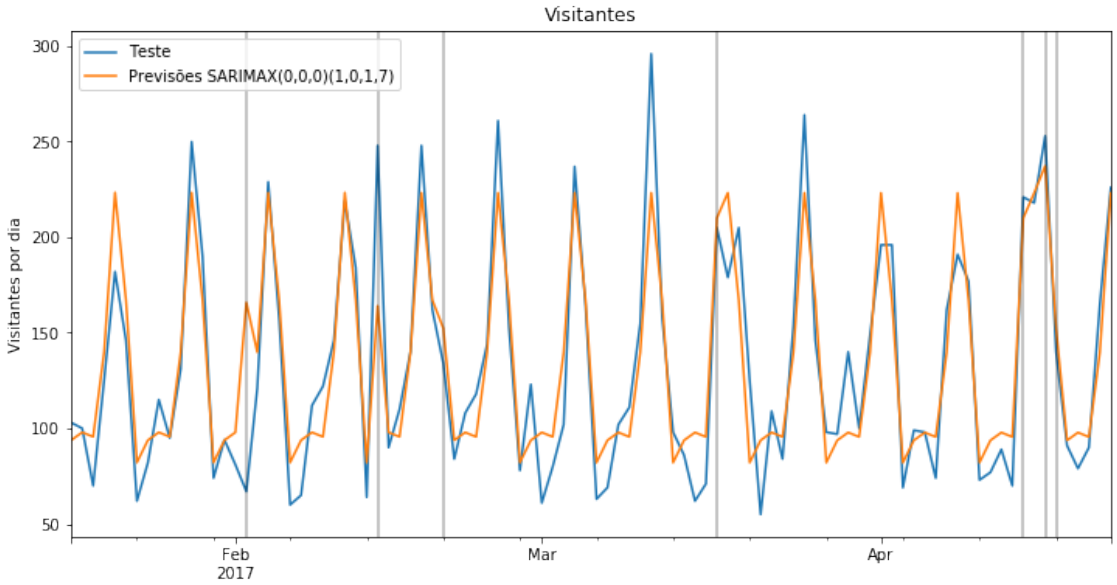
10. Obtenha os valores preditos. É importante estabelecer em predict a variável exógena.

[10]:
```
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)')
```

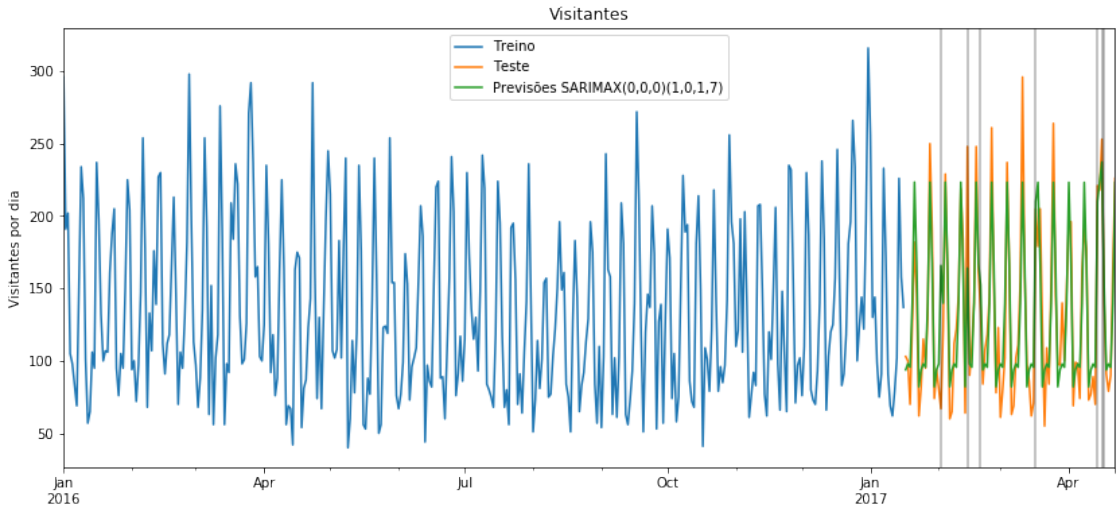
11. Observe as previsões do modelo e compare com a base de teste.

[11]:
```
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);
```



[14]:
```
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);
```

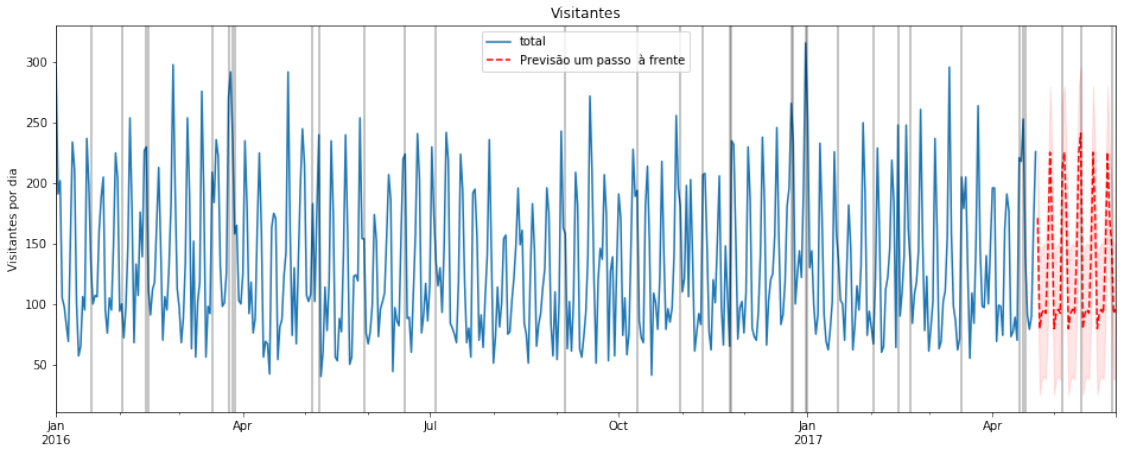


12. Faça as previsões para as próximas observações, com os feriados disponíveis na base. Nesse caso, não fará diferença a previsão passo à frente ou dinâmica.

[15]:
```
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']]
```

[16]:
```
# 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()
```

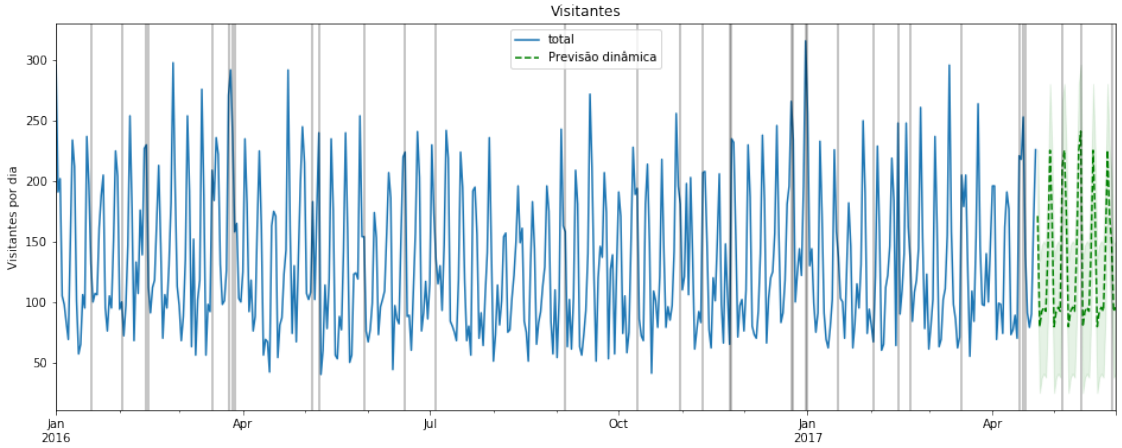
[17]:
```
title='Visitantes'
ylabel='Visitantes por dia'
xlabel=''

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.

[18]:
```
# 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()
```

[19]:
```
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

[ ]: