Aula3_2_Modelos_AR_MA_ARMA_ARIMA

August 1, 2020

1 Modelos AR, MA, ARMA, ARIMA

por **Cibele Russo**

Baseado em

- Moretting, P.A.; Toloi, C.M.C. "Análise de Séries Temporais". Blucher, 2004.
- Ehlers, R.S. (2009) Análise de Séries Temporais, http://www.icmc.usp.br/~ehlers/stemp/stemp.pdf. Acessado em 28/06/2020.

Implementações:

- Brownlee, Jason. Introduction to time series forecasting with python: how to prepare data and develop models to predict the future. Machine Learning Mastery, 2017.

Leituras adicionais:

- Box, G. E., & Jenkins, G. M. (1976). Time series analysis: Forecasting and control San Francisco. Calif: Holden-Day.
- Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). Time series analysis, forecasting and control. Englewood Clifs.
- Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice. OTexts.
- https://otexts.com/fpp2/AR.html
- https://otexts.com/fpp2/MA.html
- https://otexts.com/fpp2/non-seasonal-arima.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.ar_model.AR.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.ar_model.ARResults.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARMA.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARMAResults.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMAResults.html

2 Modelos ARIMA (p,d,q)

Componentes de um modelo ARIMA:

- **AR (p)**: Componentes autorregressivas, utilizam a relação de dependência entre a observação corrente e as observações em um período prévio
- **Integrado (d)**: Diferenças para tornar a série estacionária
- **MA (q)**: Componentes de médias móveis, utilizam a dependência entre uma obervação e um erro residual de um modelo de média móvel aplicado a observações em atraso.

SARIMA: ARIMA com sazonalidade.

Abordagem de Box e Jenkins (1976): Ajustar modelos autorregressivos integrados de médias móveis, ARIMA(p,d,q) a um conjunto de dados. Veja também Box et al. (1994).

Uma estratégia para a construção do modelo será baseada em um ciclo iterativo, na qual a escolha da estrutura do modelo é baseada nos próprios dados:

1. Uma classe geral de modelos é considerada para a análise, no caso modelos ARIMA (***especificação***)
2. Há ***identificação*** do modelo, com base na análise de autocorrelações, autocorrelações parciais e outros critérios
3. ***Estimação*** dos parâmetros do modelo identificado.
4. ***Verificação*** ou ***diagnóstico*** do modelo ajustado, por meio de uma análise de resíduos, para saber se esse modelo é adequado para fazer previsão, por exemplo

Se o modelo não for adequado, as estapas 2, 3 e 4 se repetem até obter um ajuste satisfatório. A etapa mais trabalhosa é a identificação.

No contexto desse curso, utilizaremos a função auto_arima do pacote pmdarima do Python para selecionar a ordem do ARIMA.

Nesta aula veremos uma introdução e aplicações, e continuaremos na próxima aula.

2.1 Operadores úteis

2.1.1 Operador translação (defasagem, backshift)

$BZ_t = Z_{t-1},$

$B^2 Z_t = Z_{t-2},$

$\vdots$

$B^m Z_t = Z_{t-m}.$

2.1.2 Operador translação para o futuro (forward)

$FZ_t = Z_{t+1},$

$F^2 Z_t = Z_{t+2},$

$\vdots$

$F^m Z_t = Z_{t+m}.$

2.1.3 Operador diferença

$\Delta Z_t = Z_t - Z_{t-1} = (1 - B)Z_t$

ou seja $\Delta = (1 - B)$.

2.1.4 Operador soma

$SZ_t = \sum_{j=0}^{\infty} Z_t + Z_{t-1} + Z_{t-2} + \ldots = (1 + B + B^2 + \ldots)Z_t$

do que segue que

$SZ_t = (1 - B)^{-1} Z_t = \Delta^{-1} Z_t$

ou seja $S = \Delta^{-1}$.

2.1.5 Modelos lineares estacionários

Processo linear geral (modelo de filtro linear)

$Z_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \ldots = \mu + \psi(B)a_t$

em que

$\psi(B) = 1 + \psi_1 B + \psi_2 B^2 + \ldots$

é denominada **função de transferência do filtro** e μ é um parâmetro que determina o nível da série.

Assumindo que

$E(a_t) = 0, \forall t,$

$Var(a_t) = \sigma_a^2, \forall t,$

$E(a_t a_s) = 0,\ s \neq t$

e fazendo

$\tilde{Z}_t = Z_t - \mu$, temos que

$\tilde{Z}_t = \psi(B)a_t$

Se a sequência de pesos $\{\psi_j, j \geq 1\}$ for finita ou infinita e convergente, o filtro é estável (somável) e Z_t é estacionária. Nesse caso, μ é a média do processo. Caso contrário, Z_t é não estacionária e μ não tem significado específico.

2.2 Modelos autorregressivos - AR(p)

O termo autoregressão descreve uma regressão da variável contra ela mesma. Uma regressão automática é executada em um conjunto de valores defasados da ordem p.

$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + \phi_2 \tilde{Z}_{t-2} + \ldots + \phi_p \tilde{Z}_{t-p} + a_t$

onde $\tilde{Z}_t = Z_t - \mu$, $\phi_1, \ldots, \phi_p$ **são coeficientes de atraso** até p e a_t é um ruído branco.

Por exemplo, um modelo AR (1) seria dado por

$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + a_t$

considerando que um modelo AR (2) seria dado por

$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + \phi_2 \tilde{Z}_{t-2} + a_t$

e assim por diante.

2.3 Modelos de médias móveis - MA

$\tilde{Z}_t = a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$

sendo $\tilde{Z}_t = Z_t - \mu$, teremos

$\tilde{Z}_t = (1 - \theta_1 B - \ldots - \theta_q B^q) a_t = \theta(B) a_t$

onde $\theta(B) = 1 - \theta_1 B - \ldots - \theta_q B^q$ é o operador de médias móveis de ordem q.

O exemplo mais simples é o MA(1):

$\tilde{Z}_t = a_t - \theta a_{t-1}$ ou

$\tilde{Z}_t = (1 - B) a_t$

com $\theta(B) = (1 - \theta B)$.

Um resultado garante que como $\psi(B) = 1 - \theta B$ é finito, o processo é sempre estacionário.

2.4 Modelo autorregressivo e de médias móveis

Os modelos ARMA(p,q) são dados na forma

$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + \ldots + \phi_p \tilde{Z}_{t-p} + a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$

Exemplo: Um modelo ARMA(1,1) com $p = 1$, $q = 1$, $\phi(B) = 1 - \phi B$, $\theta(B) = 1 - \theta B$, é dado por

$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + a_t - \theta_1 a_{t-1}$

Os modelos ARMA podem ser usado para séries estacionárias se as raízes de $\phi(B) = 0$ caírem todas fora do círculo unitário.

Para séries não-estacionárias com uma componente de tendência, os modelos ARIMA podem ser mais adequados.

2.4.1 Função de autocorrelação (fac) e função de autocorrelação parcial (facp) para processos AR, MA, ARMA

Função de autocorrelação (fac):

1. Um processo AR(p) tem fac que decai de acordo com exponenciais ou senoides amortecidas, infinita em extensão;
2. Um processo MA(q) tem fac finita, no sentido de que ela apresenta um corte após o "lag" q;
3. Um processo ARMA (p,q) tem fac infinita em extensão, a qual decai de acordo com exponenciais e/ou senoides amortecidas após o "laq" q-p.

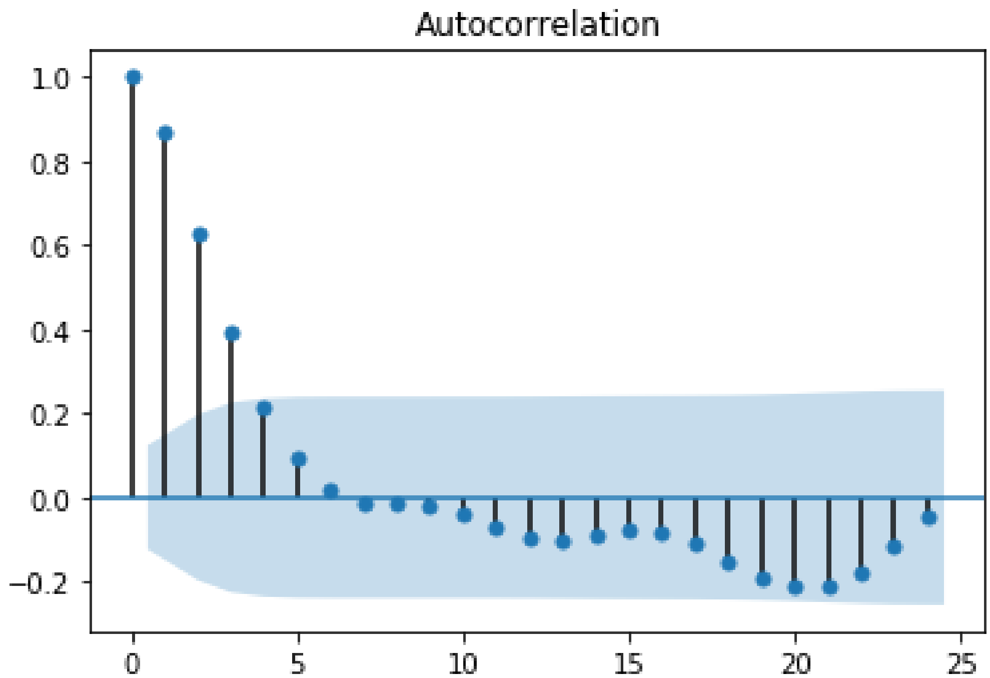
A função de autocorrelação parcial também pode auxiliar na identificação do modelo. Entre outras características,

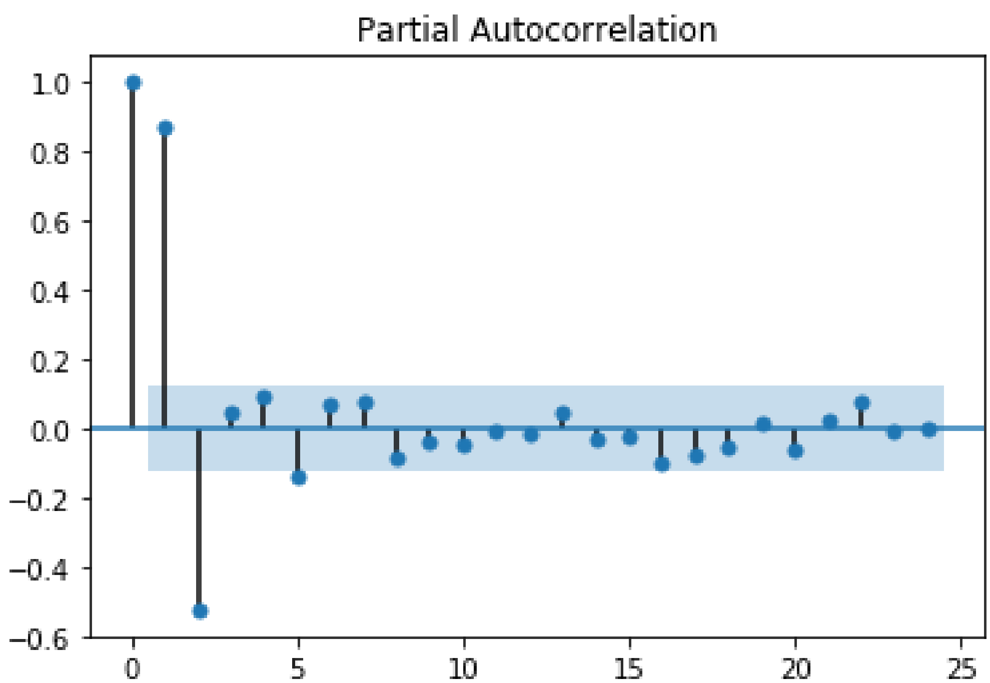
1. Um processo MA(q) tem facp que se comporta de maneira similar à fac de um processo AR(p), com decaimento exponencial e/ou senoides amortecidas;
2. Um processo ARMA(p,q) tem facp que se comporta como a facp de um processo MA puro.

Para mais informações, veja - Moretting, P.A.; Toloi, C.M.C. "Análise de Séries Temporais". Blucher, 2004. - Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). Time series analysis, forecasting and control. Englewood Clifs.

2.4.2 Correlogramas de dados simulados

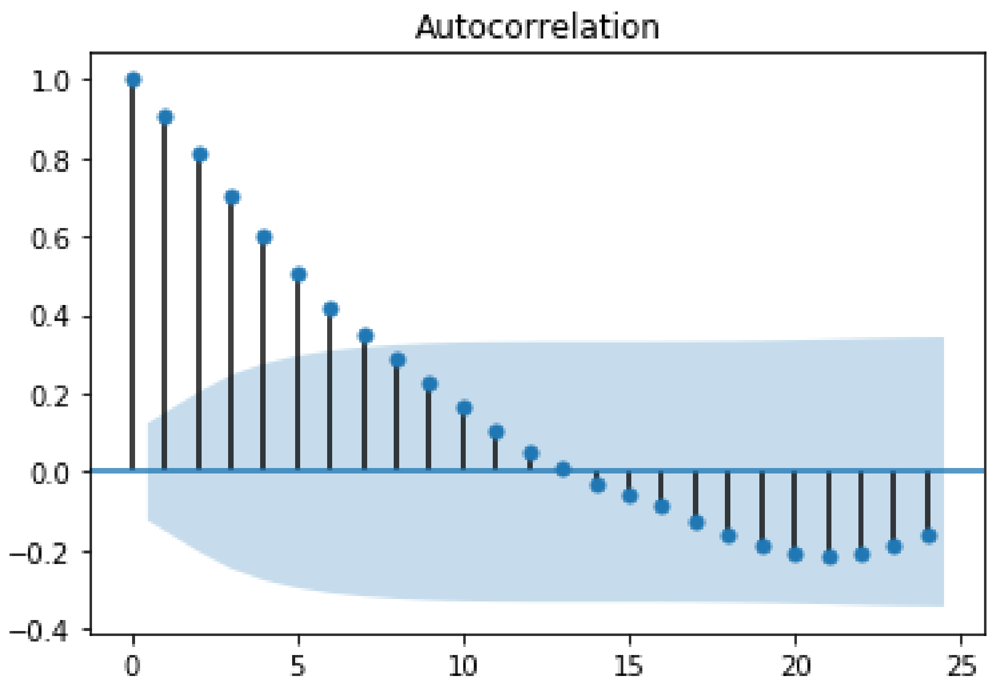
[1]:
```
# ARMA(1,2)
# Fonte: https://www.statsmodels.org/stable/generated/statsmodels.tsa.
↪arima_process.arma_generate_sample.html

import numpy as np
import statsmodels as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.tsa.arima_model import ARMA
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

np.random.seed(12345)
arparams = np.array([.6])
maparams = np.array([.65, .35])
ar = np.r_[1, -arparams] # add zero-lag and negate
ma = np.r_[1, maparams] # add zero-lag
y = arma_generate_sample(ar, ma, 250)
model = ARMA(y, (1,2)).fit(trend='nc', disp=0) # Exercício: Olhar o ajuste

plot_acf(y)
plot_pacf(y)
plt.show()
```

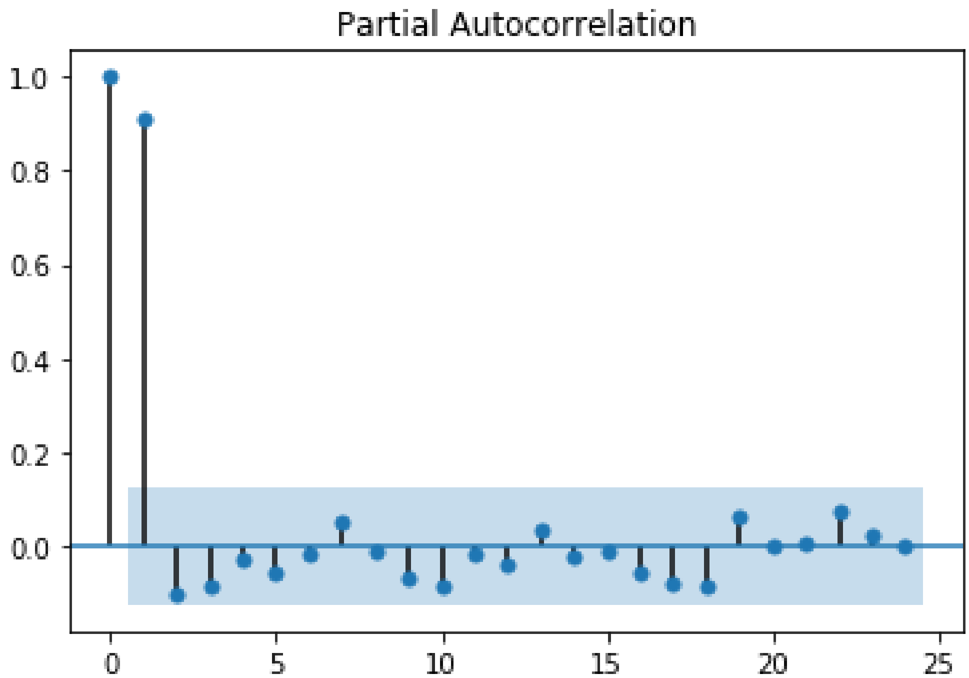




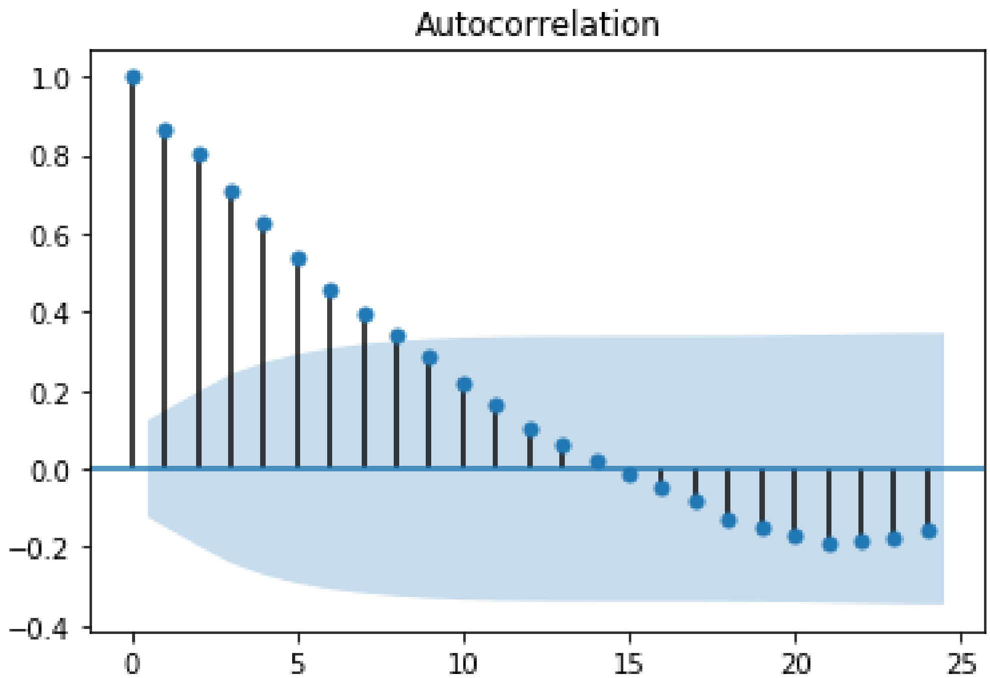

```
[2]: # ARMA(1,0) ou AR(1)
```

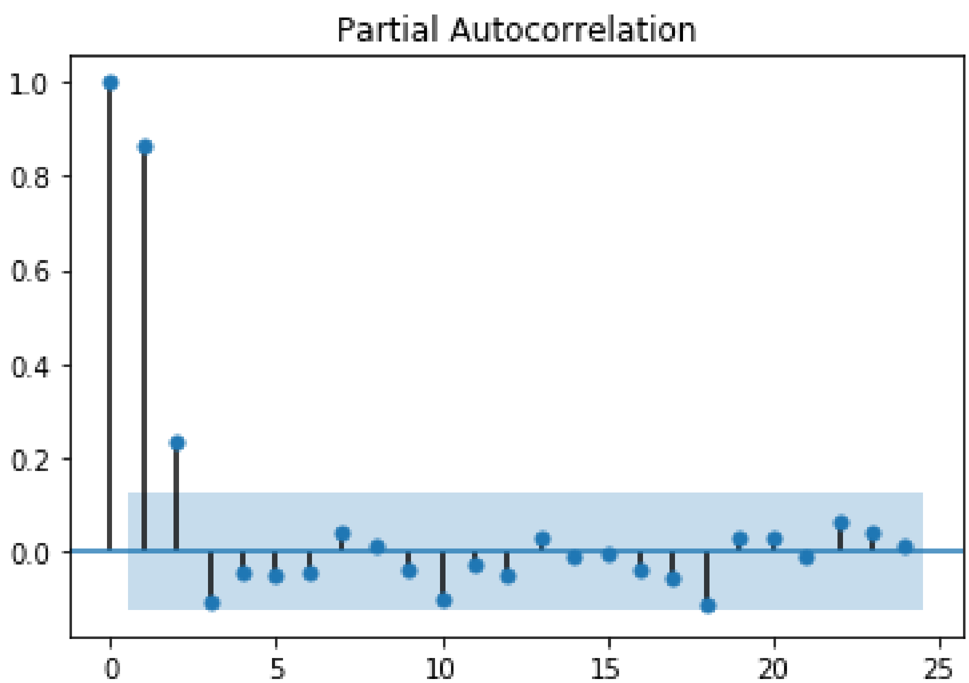
```
np.random.seed(12345)
arparams = np.array([.9])
maparams = np.array([.65, .35])
ar = np.r_[1, -arparams]
ma = np.r_[1]
y = arma_generate_sample(ar, ma, 250)
model = ARMA(y, (1,0)).fit(trend='nc', disp=0)
model.params

plot_acf(y)
plot_pacf(y)
plt.show()
```





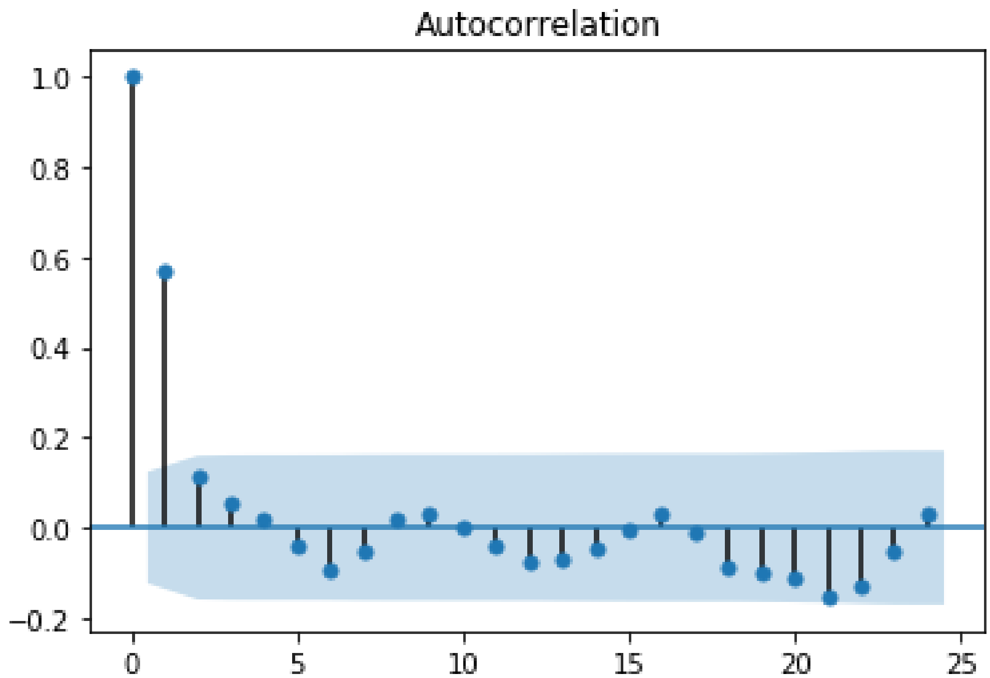
[3]:
```
# ARMA(2,0) ou AR(2)
# Fonte: https://www.statsmodels.org/stable/generated/statsmodels.tsa.
↪arima_process.arma_generate_sample.html

np.random.seed(12345)
arparams = np.array([.6, 0.3])
maparams = np.array([.8, .35])
ar = np.r_[1, -arparams]
ma = np.r_[1]
y = arma_generate_sample(ar, ma, 250)
model = ARMA(y, (2,0)).fit(trend='nc', disp=0)
model.params

plot_acf(y)
plot_pacf(y)
plt.show()
```

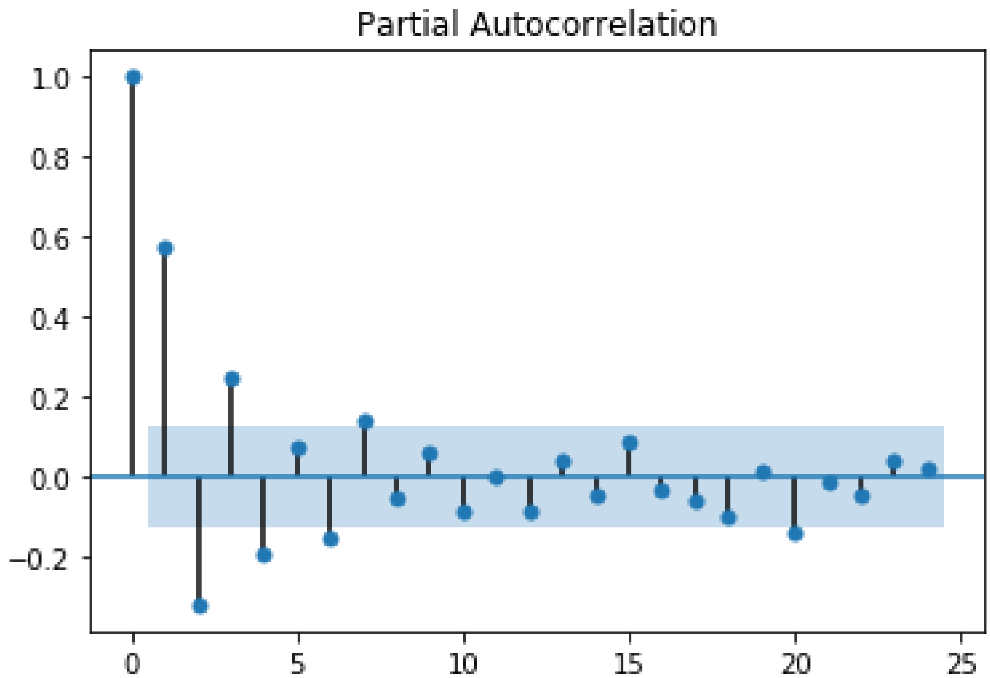




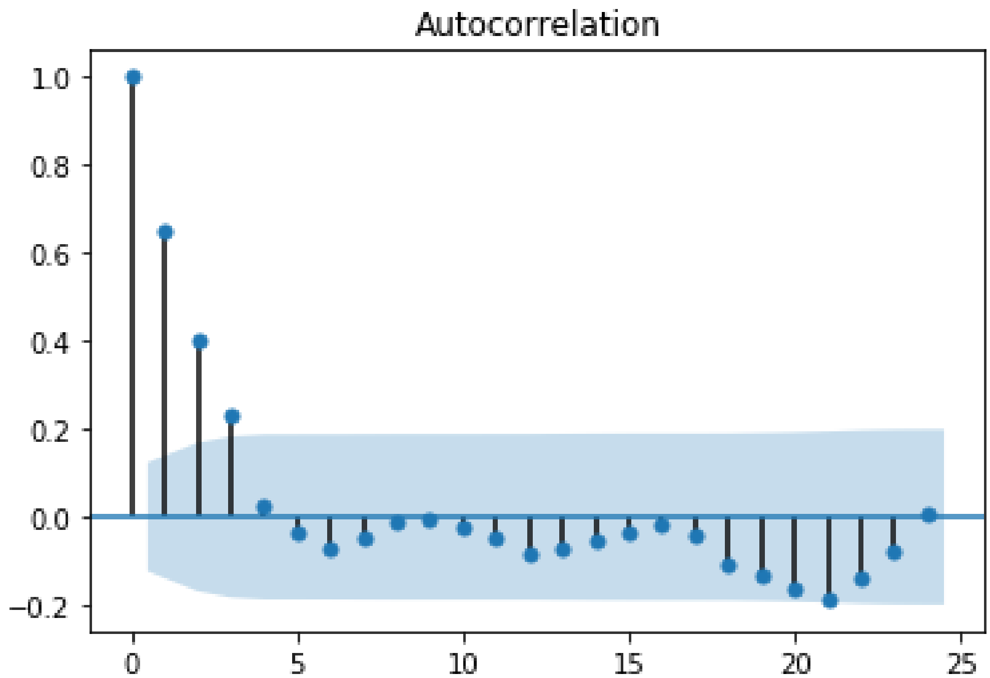
[4]:
```
# ARMA(0,1) ou MA(1)
```

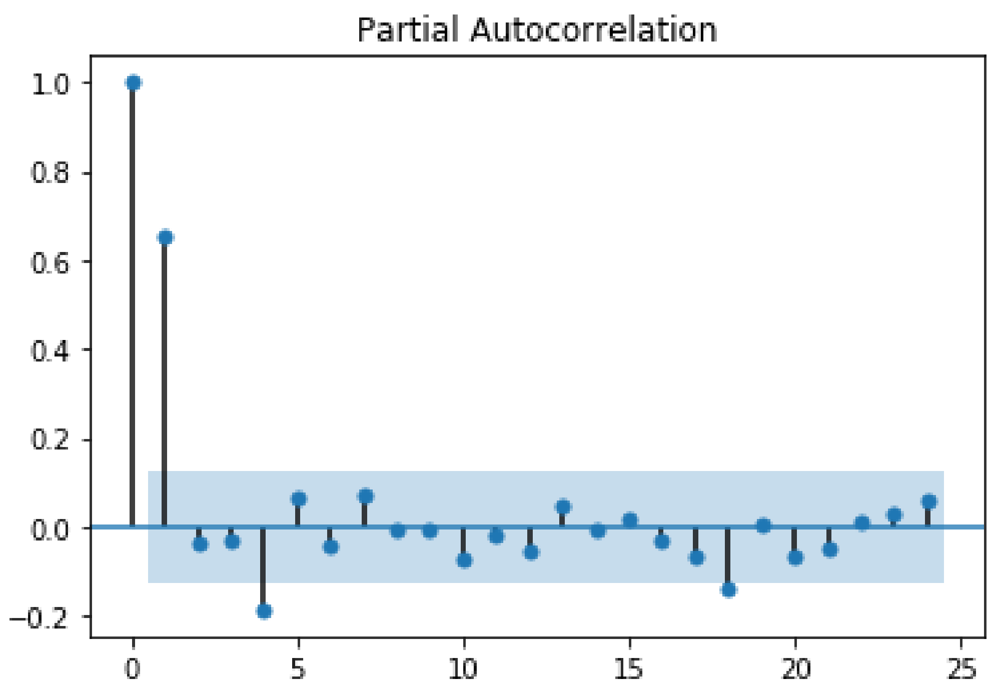
```
np.random.seed(12345)
arparams = np.array([.75])
maparams = np.array([.85])
ar = np.r_[1]
ma = np.r_[1, maparams]
y = arma_generate_sample(ar, ma, 250)
model = ARMA(y, (0, 1)).fit(trend='nc', disp=0)

plot_acf(y)
plot_pacf(y)
plt.show()
```





[5]:
```
# ARMA(0,3) ou MA(3)
# Fonte: https://www.statsmodels.org/stable/generated/statsmodels.tsa.
↪arima_process.arma_generate_sample.html

np.random.seed(12345)
arparams = np.array([.75])
maparams = np.array([.6, 0.3, 0.3])
ar = np.r_[1]
ma = np.r_[1, maparams]
y = arma_generate_sample(ar, ma, 250)
model = ARMA(y, (0, 3)).fit(trend='nc', disp=0)

plot_acf(y)
plot_pacf(y)
plt.show()
```


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


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

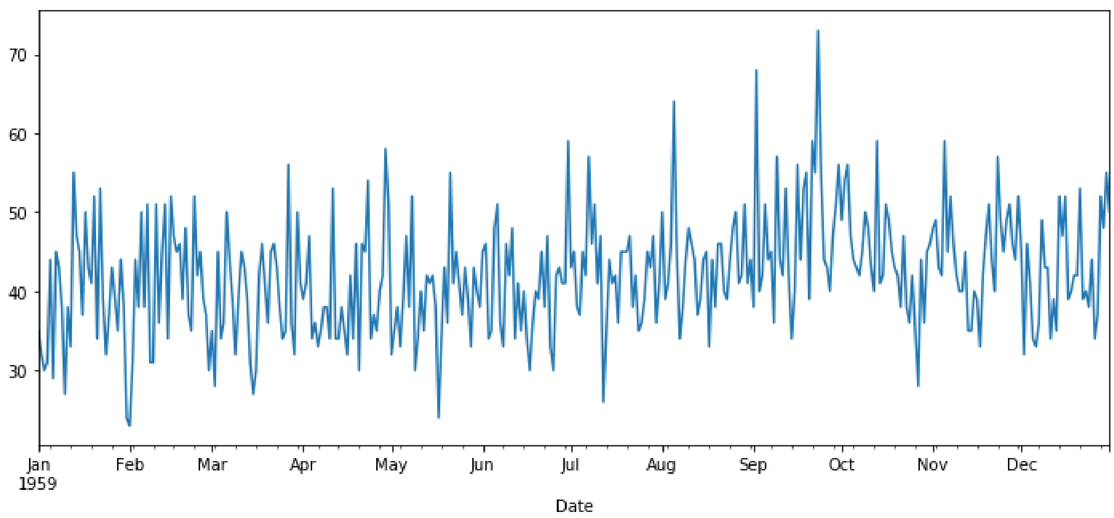
2.5 Aplicações

[6]:
```
import pandas as pd
import numpy as np
%matplotlib inline
from statsmodels.tsa.stattools import adfuller

# Funções específicas para a modelagem e previsão
from statsmodels.tsa.arima_model import ARMA,ARMAResults,ARIMA,ARIMAResults
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # para determinar␣
↪(p,q)
from pmdarima import auto_arima # Para determinar a ordem do ARIMA

# Carregue as bases de dados
# Trabalharemos com uma série estacionária, de nascimentos de mulheres, e uma␣
↪série não estacionária, PETR4

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

# Dados de nascimentos diários de mulheres
df1 = pd.read_csv(f'{pkgdir}/DailyTotalFemaleBirths.
↪csv',index_col='Date',parse_dates=True)
df1.index.freq = 'D'

# Dados de fechamento das ações da PETR4
df2 = pd.read_csv(f'{pkgdir}/PETR4.csv',index_col='Date',parse_dates=True)
idx = pd.date_range(start=df2.index.min(), end=df2.index.max(), freq='B')
df2 = df2.reindex(idx)
df2.fillna(method='ffill', inplace=True)
```

2.6 Média móvel autoregressiva - ARMA (p, q)

Olhemos a princípio uma série estacionária e determinaremos (p,q) em um modelo ARMA.

[7]:
```
df1['Births'].plot(figsize=(12,5));
```



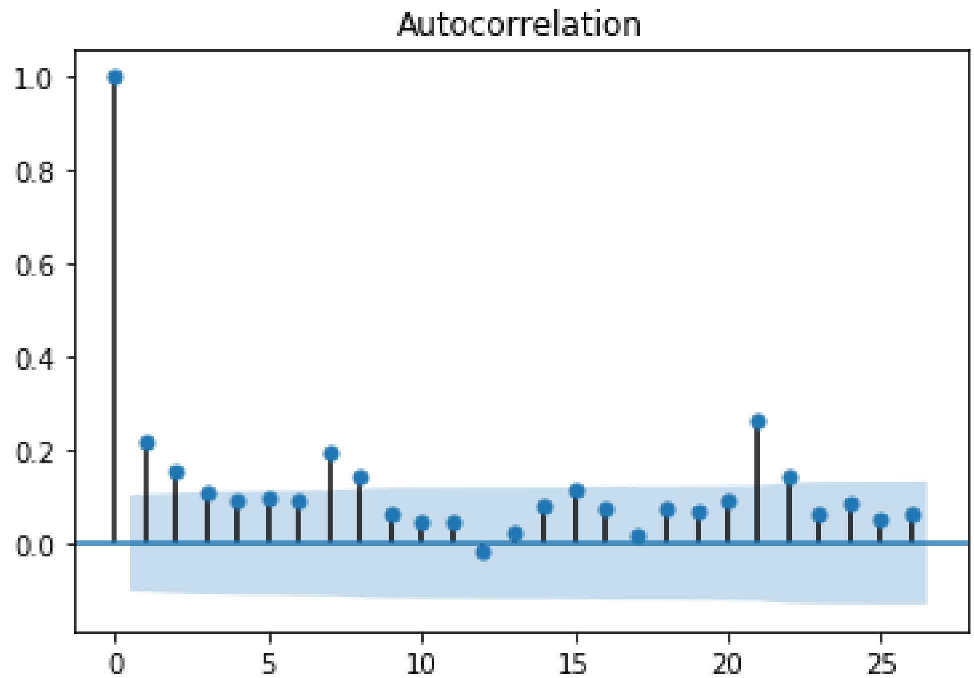
2.6.1 Execute o teste Dickey-Fuller aumentado para confirmar a estacionariedade

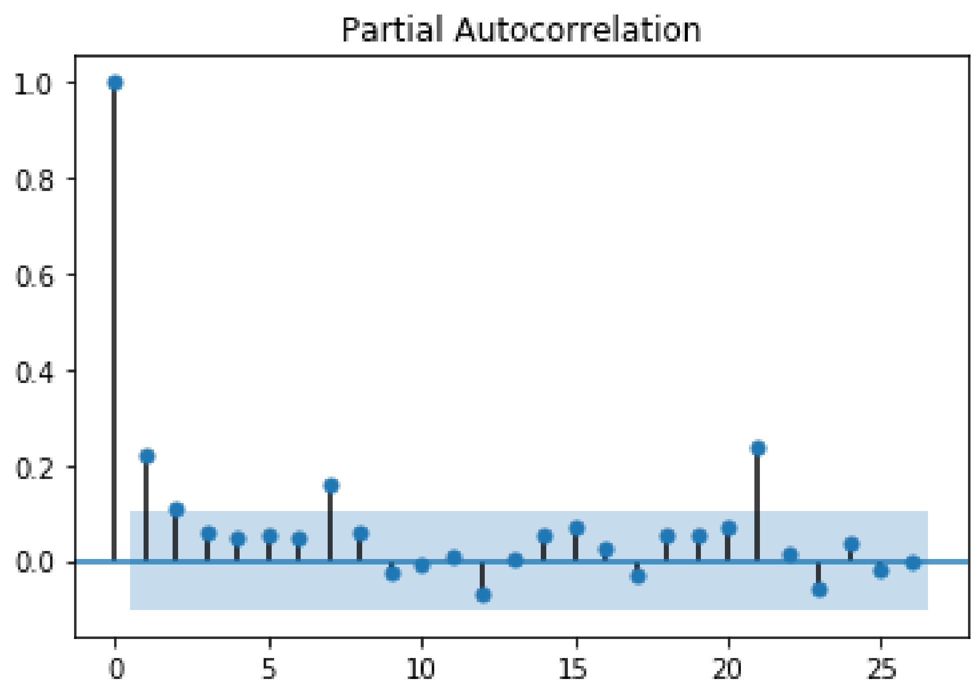
[8]:
```
# Teste de Dickey-Fuller aumentado
# fonte: https://machinelearningmastery.com/time-series-data-stationary-python/

result = adfuller(df1['Births'], 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: -4.808291
p-value: 0.000052
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
```

Há fortes evidências de que a série seja estacionária ou tendência-estacionária.

[9]:
```
# Correlograma

plot_acf(df1['Births'])
plot_pacf(df1['Births'])
plt.show()
```





[10]:
```
# Vamos considerar um modelo ARMA(p,q) e a função auto_arima
```

[11]:
```
stepwise_fit = auto_arima(df1['Births'], start_p=0, start_q=0,
                          max_p=6, max_q=3, m=12,
                          seasonal=False,
                          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(0,0,0,0) [intercept=True]; AIC=2494.782, BIC=2502.582,
Time=0.066 seconds
Fit ARIMA(1,0,0)x(0,0,0,0) [intercept=True]; AIC=2479.081, BIC=2490.780,
Time=0.094 seconds
Fit ARIMA(0,0,1)x(0,0,0,0) [intercept=True]; AIC=2482.539, BIC=2494.239,
Time=0.241 seconds
Fit ARIMA(0,0,0)x(0,0,0,0) [intercept=False]; AIC=3776.976, BIC=3780.876,
Time=0.016 seconds
Fit ARIMA(2,0,0)x(0,0,0,0) [intercept=True]; AIC=2476.368, BIC=2491.968,
Time=0.261 seconds
Fit ARIMA(3,0,0)x(0,0,0,0) [intercept=True]; AIC=2477.027, BIC=2496.526,
Time=0.385 seconds
Fit ARIMA(2,0,1)x(0,0,0,0) [intercept=True]; AIC=2474.844, BIC=2494.344,
Time=1.286 seconds
Fit ARIMA(1,0,1)x(0,0,0,0) [intercept=True]; AIC=2471.625, BIC=2487.224,
Time=0.962 seconds
Fit ARIMA(1,0,2)x(0,0,0,0) [intercept=True]; AIC=2472.318, BIC=2491.818,
Time=1.305 seconds
Fit ARIMA(0,0,2)x(0,0,0,0) [intercept=True]; AIC=2479.132, BIC=2494.732,
Time=0.371 seconds
Fit ARIMA(2,0,2)x(0,0,0,0) [intercept=True]; AIC=2475.453, BIC=2498.853,
Time=1.538 seconds
Total fit time: 6.535 seconds
```

[11]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  365
Model:               SARIMAX(1, 0, 1)   Log Likelihood               -1231.812
Date:                Tue, 28 Jul 2020   AIC                           2471.625
Time:                        00:34:18   BIC                           2487.224
Sample:                             0   HQIC                          2477.824
                                - 365
```

```
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      9.0934      4.540      2.003      0.045       0.195      17.992
ar.L1          0.7832      0.108      7.226      0.000       0.571       0.996
ma.L1         -0.6176      0.140     -4.411      0.000      -0.892      -0.343
sigma2        48.9425      3.433     14.254      0.000      42.213      55.672
================================================================================
===
Ljung-Box (Q):                       52.57   Jarque-Bera (JB):
18.52
Prob(Q):                              0.09   Prob(JB):
0.00
Heteroskedasticity (H):               0.94   Skew:
0.48
Prob(H) (two-sided):                  0.74   Kurtosis:
3.53
================================================================================
===

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

2.6.2 Considere agora bases de treino e teste

[12]:
```python
len(df1)
```

[12]: 365

[13]:
```python
treino = df1.iloc[:290]
teste = df1.iloc[290:]
```

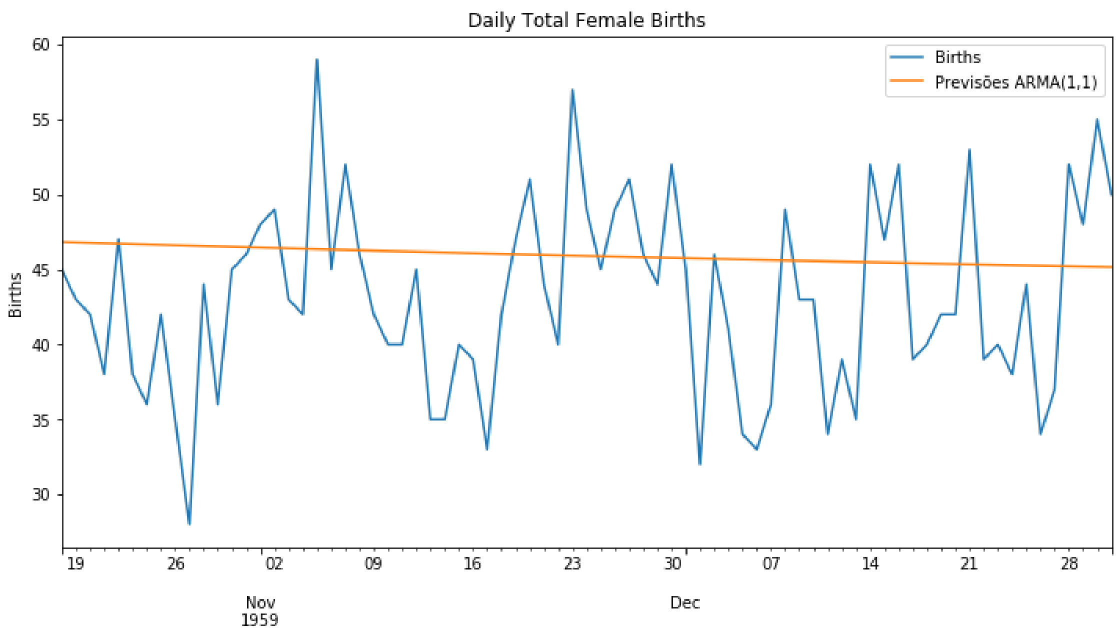
[14]:
```python
modelo = ARMA(treino['Births'],order=(1,1))
resultados = modelo.fit()
resultados.summary()
```

[14]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Births   No. Observations:                  290
Model:                     ARMA(1, 1)   Log Likelihood                -982.268
Method:                       css-mle   S.D. of innovations              7.145
```

```
Date:                Tue, 28 Jul 2020   AIC                          1972.537
Time:                        00:34:18   BIC                          1987.216
Sample:                    01-01-1959   HQIC                         1978.418
                         - 10-17-1959
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const           42.3460      2.673     15.843      0.000      37.107      47.585
ar.L1.Births     0.9937      0.009    113.910      0.000       0.977       1.011
ma.L1.Births    -0.9452      0.024    -39.192      0.000      -0.992      -0.898
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0063           +0.0000j            1.0063            0.0000
MA.1            1.0580           +0.0000j            1.0580            0.0000
-----------------------------------------------------------------------------
"""
```

[15]:
```
# Olhando as previsões

inicio=len(treino)
fim=len(treino)+len(teste)-1
previsoes = resultados.predict(start=inicio, end=fim).rename('Previsões␣
 ↪ARMA(1,1)')
```

[16]:
```
title = 'Daily Total Female Births'
ylabel='Births'
xlabel='' # we don't really need a label here

ax = teste['Births'].plot(legend=True,figsize=(12,6),title=title)
previsoes.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



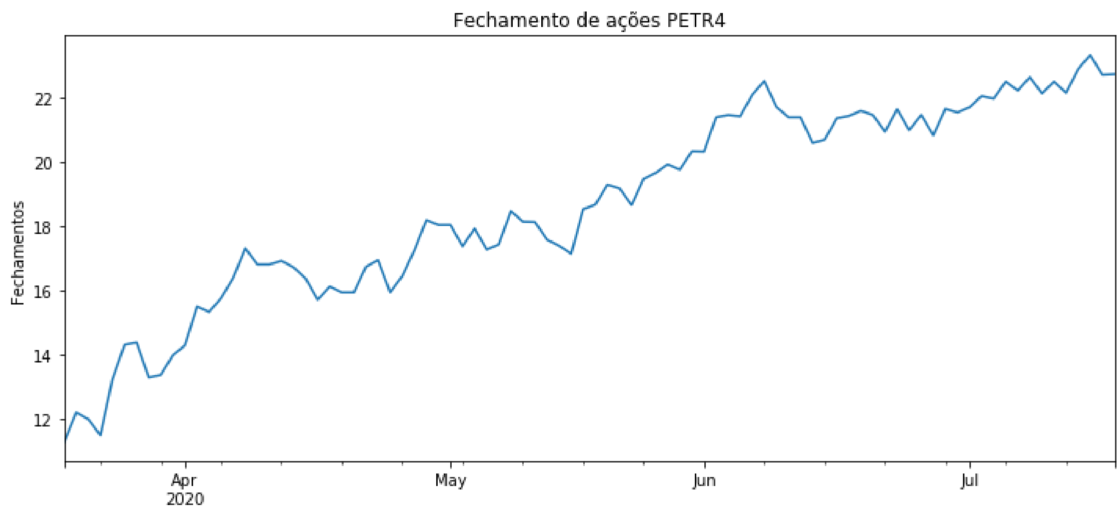
Como nosso conjunto de dados inicial não exibiu tendência ou componente sazonal, essa previsão faz sentido. Na próxima seção, tomaremos medidas adicionais para avaliar o desempenho de nossas previsões e fazer previsões para o futuro.

2.7 Média móvel integrada autoregressiva - ARIMA (p, d, q)

As etapas são as mesmas que para o ARMA (p, q), exceto que aplicaremos um componente diferencial para tornar o conjunto de dados estacionário. Primeiro, vamos dar uma olhada no conjunto de dados PETR4. ### Plotar os dados de origem

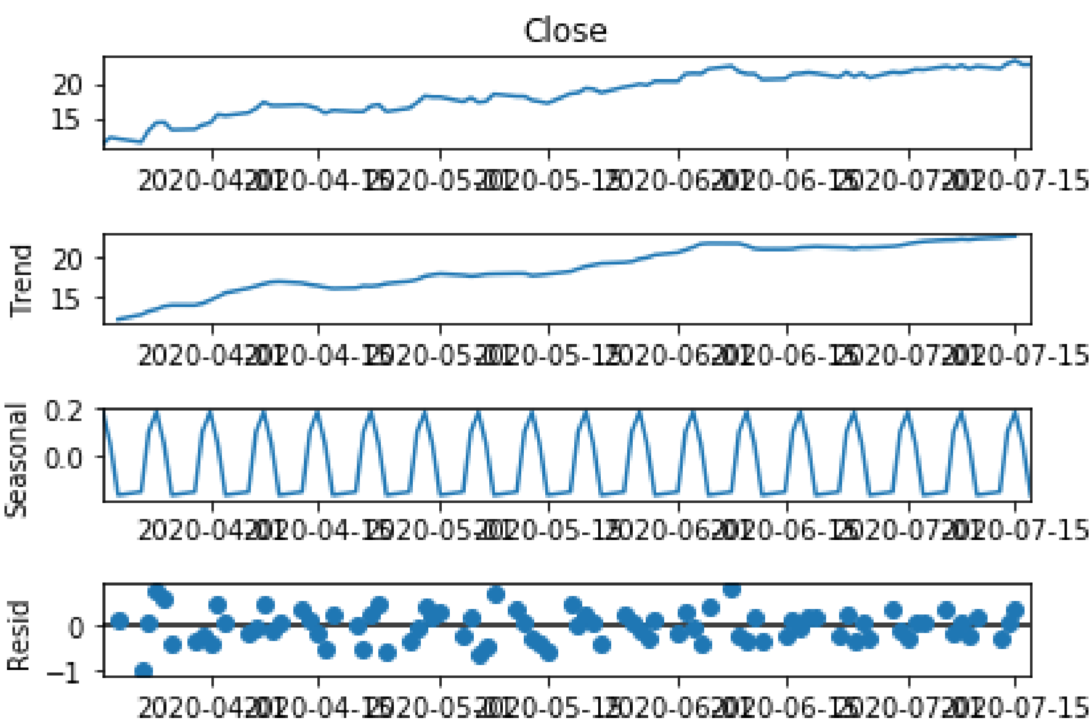
[17]:
```
title = 'Fechamento de ações PETR4'
ylabel='Fechamentos'

ax = df2['Close'].plot(figsize=(12,5),title=title);
ax.set(xlabel=xlabel, ylabel=ylabel)
```

[17]:
```
[Text(0, 0.5, 'Fechamentos'), Text(0.5, 0, '')]
```



2.7.1 Executar uma decomposição em tendência e sazonalidade

[18]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df2['Close'], model='additive' )  # model='add' also works
result.plot();
```



Possivelmente existe sazonalidade nesses dados. A princípio vamos considerar um modelo ARIMA não sazonal.

Vamos usar o pdmarima.auto_arima para determinar o modelo ARIMA.

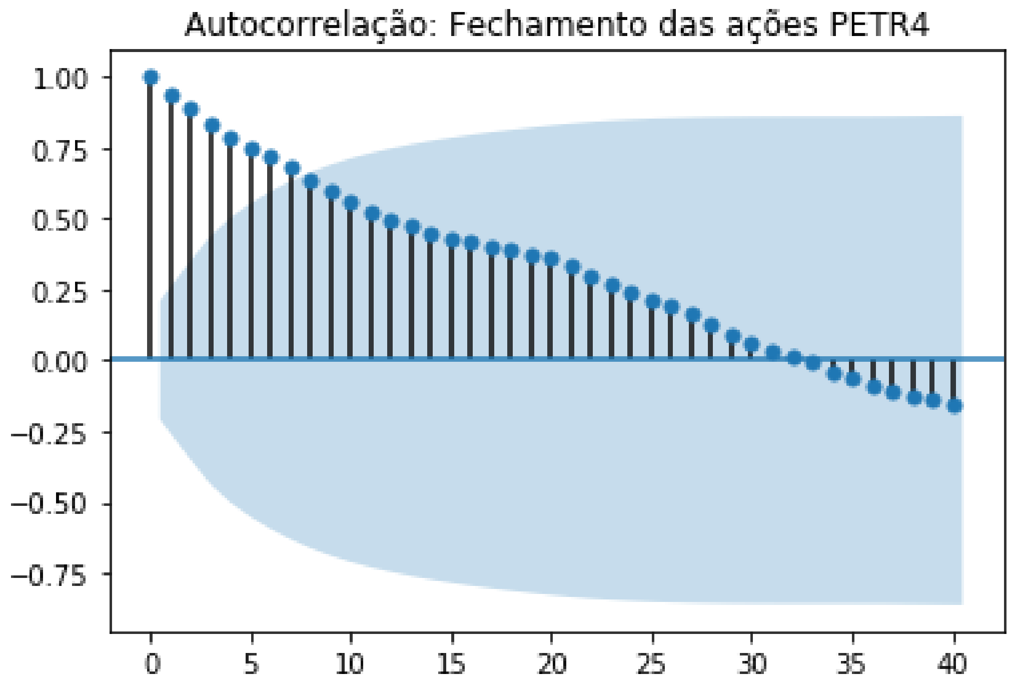
2.7.2 Execute o teste Dickey-Fuller aumentado na primeira diferença

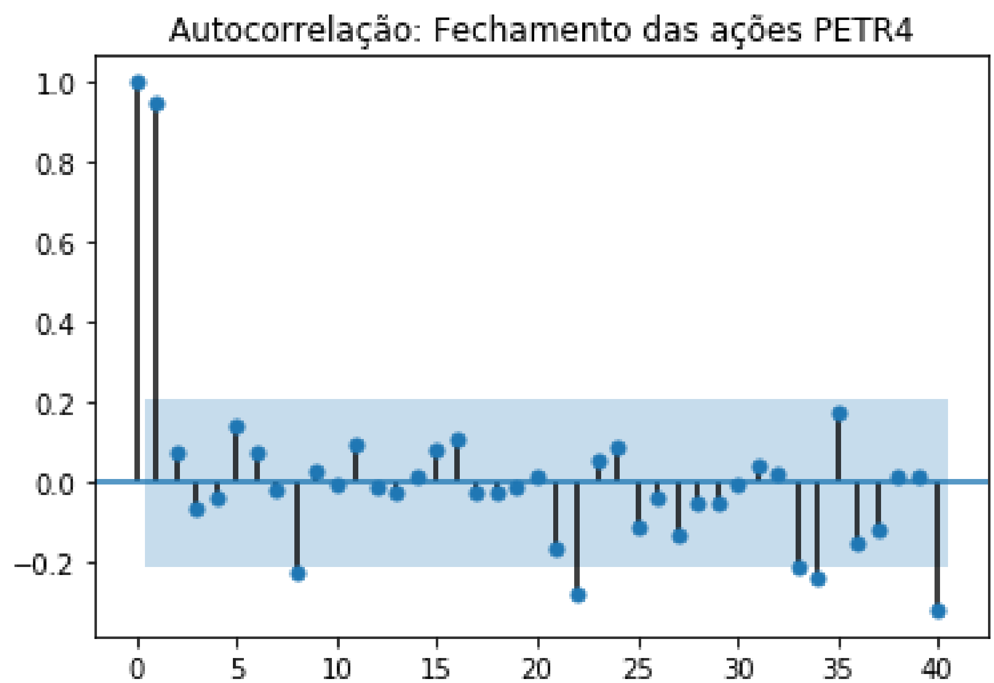
[19]:
```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.tools import diff

result = adfuller(diff(df2['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
```

Isso confirma que atingimos a estacionariedade após a primeira diferença. ### Construa os gráficos ACF e PACF

[20]:
```
title = 'Autocorrelação: Fechamento das ações PETR4'
lags = 40
plot_acf(df2['Close'],title=title,lags=lags);
plot_pacf(df2['Close'],title=title,lags=lags);
plt.show();
```





Vamos dar uma olhada no pmdarima.auto_arima feito com o stepwise_fit para ver se ter os termos p e q o mesmo ainda faz sentido:

[21]:
```
stepwise_fit = auto_arima(df2['Close'], start_p=0, start_q=0,
                          max_p=2, max_q=2, m=7,
                          seasonal=False,
                          d=None, 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(0,1,0)x(0,0,0,0) [intercept=True]; AIC=152.390, BIC=157.322,
Time=0.034 seconds
Fit ARIMA(1,1,0)x(0,0,0,0) [intercept=True]; AIC=153.013, BIC=160.410,
Time=0.060 seconds
Fit ARIMA(0,1,1)x(0,0,0,0) [intercept=True]; AIC=152.598, BIC=159.996,
Time=0.070 seconds
Fit ARIMA(0,1,0)x(0,0,0,0) [intercept=False]; AIC=154.944, BIC=157.410,
Time=0.017 seconds
Fit ARIMA(1,1,1)x(0,0,0,0) [intercept=True]; AIC=152.746, BIC=162.610,
Time=0.259 seconds
Total fit time: 0.448 seconds
```

[21]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   88
Model:               SARIMAX(0, 1, 0)   Log Likelihood                 -74.195
Date:                Tue, 28 Jul 2020   AIC                            152.390
Time:                        00:34:22   BIC                            157.322
Sample:                             0   HQIC                           154.376
                                 - 88
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.1316      0.062      2.124      0.034       0.010       0.253
sigma2         0.3223      0.053      6.046      0.000       0.218       0.427
================================================================================
===
Ljung-Box (Q):                       68.54   Jarque-Bera (JB):
1.14
Prob(Q):                              0.00   Prob(JB):
0.57
```

```
Heteroskedasticity (H):               0.57   Skew:
0.25
Prob(H) (two-sided):                  0.14   Kurtosis:
2.74
================================================================================
===

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

Vamos avaliar o modelo ARIMA (0,1,0) em bases de treino e teste

2.7.3 Divida os dados em conjuntos de treino / teste

[22]:
```python
len(df2)
```

[22]: 88

[23]:
```python
# Defina as bases de treino e teste
treino = df2.iloc[:70]
teste = df2.iloc[70:]
```

2.7.4 Ajuste um modelo ARIMA (0,1,0)

[24]:
```python
modelo = ARIMA(treino['Close'],order=(0,1,0))
resultados = modelo.fit()
resultados.summary()
```

[24]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.Close   No. Observations:                   69
Model:                 ARIMA(0, 1, 0)   Log Likelihood                 -61.375
Method:                           css   S.D. of innovations              0.589
Date:                Tue, 28 Jul 2020   AIC                            126.749
Time:                        00:34:23   BIC                            131.217
Sample:                    03-19-2020   HQIC                           128.522
                         - 06-23-2020
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1501      0.071      2.118      0.034       0.011       0.289
==============================================================================
```

"""

[25]:
```
# Obtain predicted values
inicio=len(treino)
fim=len(treino)+len(teste)-1
previsoes = resultados.predict(start=inicio, end=fim, dynamic=False,␣
 ↪typ='levels').rename('Previsões ARIMA (0,1,0)')
```

Passar dynamic = False significa que as previsões em cada ponto são geradas usando o histórico completo até aquele ponto (todos os valores defasados).

Passar typ = 'levels' prevê os níveis das variáveis endógenas originais. Se tivéssemos usado o padrão typ = 'linear' , teríamos visto previsões lineares em termos de variáveis endógenas diferenciadas.

Para obter mais informações sobre esses argumentos, visite https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMAResults.predict.html

[26]:
```
treino['Close'].plot(legend=True, label='Treino')
teste['Close'].plot(legend=True, label='Teste')
previsoes.plot(legend=True, figsize=(8,6))
```

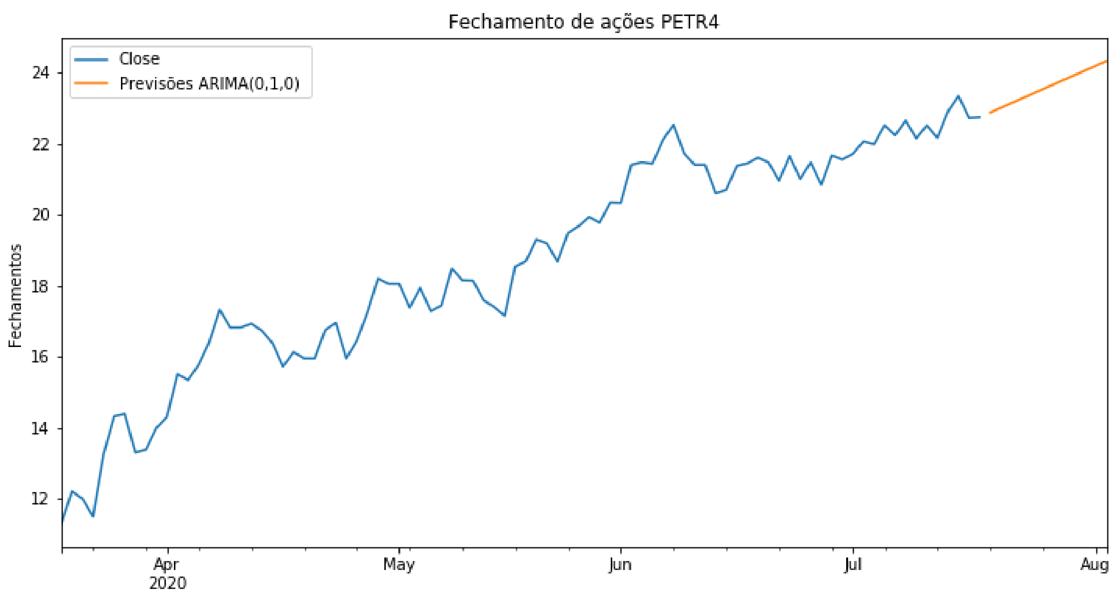
[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f54136e2ed0>

2.7.5 Treine novamente o modelo com os dados completos e preveja o futuro

[27]:
```
modelo = ARIMA(df2['Close'],order=(0,1,0))
resultados = modelo.fit()
fcast = resultados.predict(len(df2),len(df2)+11,typ='levels').rename('Previsões␣
↪ARIMA(0,1,0) ')
```

[28]:
```
# Previsão
title = 'Fechamento de ações PETR4'
ylabel='Fechamentos'
xlabel=''

ax = df2['Close'].plot(legend=True,figsize=(12,6),title=title)
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
```

[28]:
```
[Text(0, 0.5, 'Fechamentos'), Text(0.5, 0, '')]
```



Continuaremos na próxima aula!