tp_06 (1)

December 7, 2020

1 Lab work 06 - S1 2020-2021

Notions to be mastered at the end of this class: * manipulation of time series * notion of stationarity * ARIMA modelling --

```
In [37]: #Imports
         %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
         from scipy import stats
         import statsmodels.api as sm
         from statsmodels.tsa.arima_model import ARIMA
         from statsmodels.tsa.stattools import adfuller, kpss
         from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

1.1 A Stationarity analysis

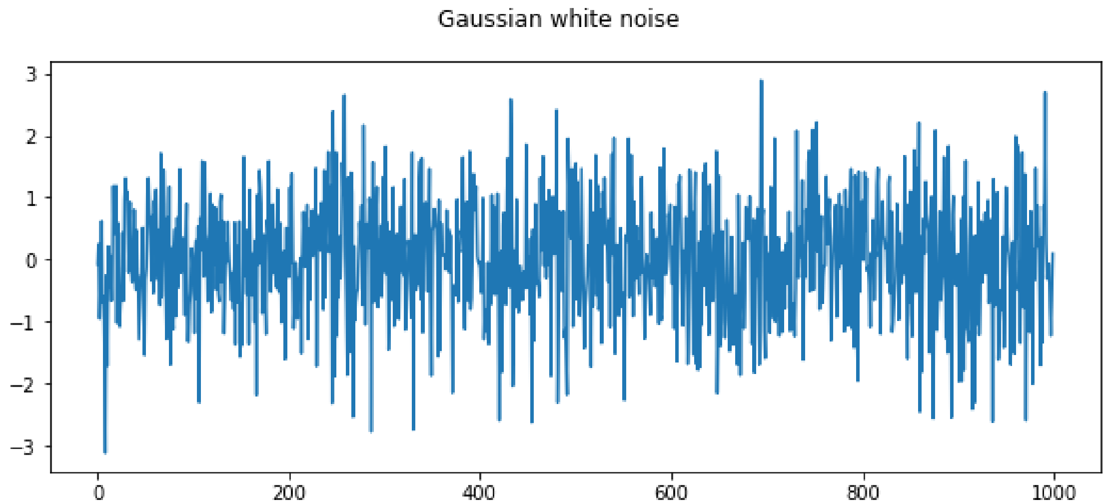
1.1.1 A.1 Stationarity analysis of gaussian white noise

Question 1

```
In [38]: # Generate a white noise signal containing 1000 samples
         nb_samples = 1000
         x = np.random.normal(0, 1, size=nb_samples)
```

Question 2

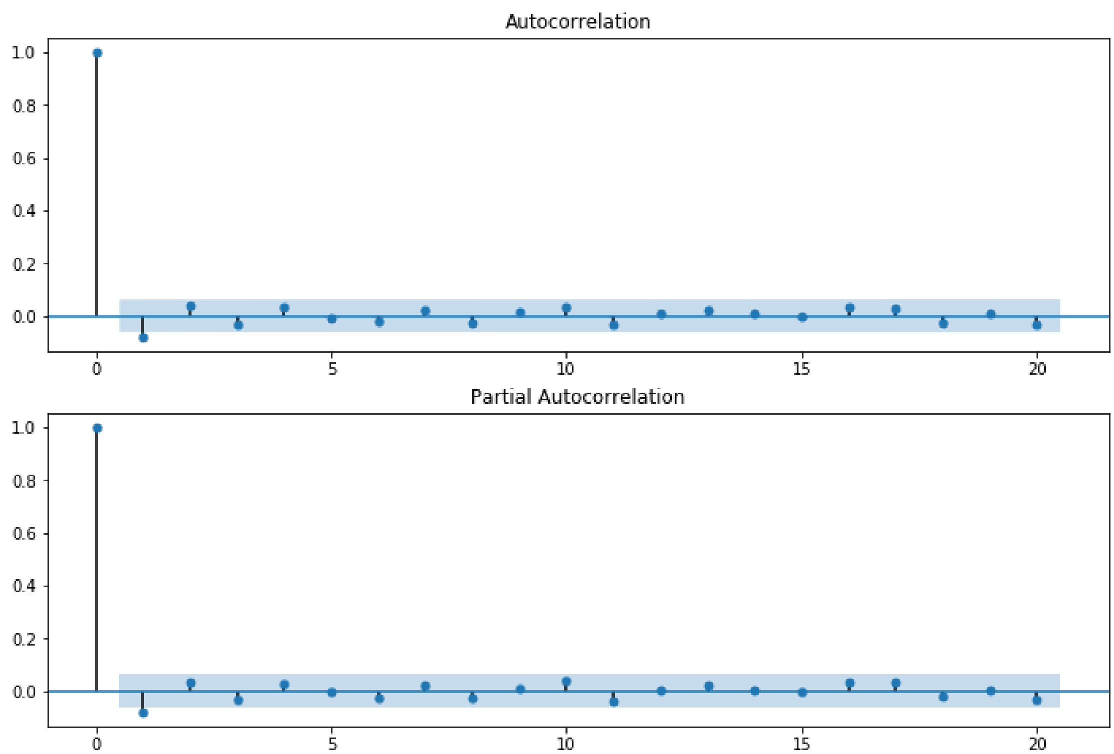
```
In [39]: # Plot the white noise signal
         fig, ax = plt.subplots(figsize=(10, 4))
         ax.plot(x);
         fig.suptitle('Gaussian white noise');
```



Comments * Series values centered around 0 * The series looks stationary

Question 3

```
In [40]: fig = plt.figure(figsize=(12,8))
         ax1 = fig.add_subplot(211)
         fig = sm.graphics.tsa.plot_acf(x, lags=20, ax=ax1)
         ax2 = fig.add_subplot(212)
         fig = sm.graphics.tsa.plot_pacf(x, lags=20, ax=ax2)
```



Comments * The ACF drops to zero relatively quickly (exponential decay) * Only the first ACF/PACF value is out of the confidence interval (blue area). * These plots suggest that the series is stationary

Question 4

```
In [41]: #Create a function to check stationarity according to ADF and KPSS tests
         def stationarity_tests(arr):

             #Check the stationarity of the the input series using the ADF test
             p_val = adfuller(arr)[1]

             #Threshold th ep-value
             if p_val <= 0.05:
                 sign = '<='
                 adf_stat = True
             else:
                 sign = '>'
                 adf_stat = False
             print('ADF: p-value = {0} {1} 0.05'.format(p_val, sign));

             #Check the stationarity of the input series using the KPSS test
             p_val = kpss(arr, nlags='auto')[1]

             #Threshold th ep-value
             if p_val <= 0.05:
                 sign = '<='
                 kpss_stat = False
             else:
                 sign = '>'
                 kpss_stat = True
             print('KPSS: p-value = {0} {1} 0.05'.format(p_val, sign));

             #Conclusion on stationarity
             if adf_stat and kpss_stat:
                 stat_status = ''
             if not adf_stat and not kpss_stat:
                 stat_status = 'non'
             if adf_stat and not kpss_stat:
                 stat_status = 'difference'
             if not adf_stat and kpss_stat:
                 stat_status = 'trend'

             print('The series is {0} stationary.'.format(stat_status))

             return stat_status
```

In [42]:
```
#Evaluate the gaussian white noise stationarity
stat_status = stationarity_tests(x)
```

```
ADF: p-value = 0.0 <= 0.05
KPSS: p-value = 0.1 > 0.05
The series is  stationary.


c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\stattoo
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

- ADF: p value <= 0.05 - there is evidence for rejecting H0 - gaussian white noise is stationary.
- KPSS: p value > 0.05 - H0 cannot be rejected - gaussian white noise is trend stationary.
- Both tests conclude that gaussian white noise is stationary

  references: https://www.statsmodels.org/stable/examples/notebooks/generated/stationarity_detrending_

1.1.2 B.1 Stationarity of a real case series

Chiffre d'affaires en supermarchés et hypermarchés par type de produits
Source : Insee - Enquête mensuelle sur l'activité des grandes surfaces alimentaires (Emagsa)
https://www.insee.fr/fr/statistiques/4923139#graphique-ca-gsa-g2-fr

Question 1

In [43]:
```
# Load the file
df = pd.read_csv('sales.csv', index_col=0)
nb_samples, nb_features = df.shape
print('{0} samples and {1} features/labels'.format(nb_samples, nb_features))
print(df.head())
```

```
57 samples and 3 features/labels
           Food   Fuel  Notfood
2016-01   99.43  93.31    99.27
2016-02   99.88  93.85    99.90
2016-03  101.21  92.83    99.11
2016-04   96.78  92.33   100.12
2016-05   98.45  99.12    98.09
```

Question 2

In [44]:
```
#Save the food and fuel sales between 2016 and 2019 into an array
food_sales = df.loc['2016-01':'2019-12']['Food'].values
fuel_sales = df.loc['2016-01':'2019-12']['Fuel'].values
```

In [45]:
```
#Apply a Box-Pierce tests on the series
for arr in [food_sales, fuel_sales]:
    _, a, bpvalue, bppvalue = sm.stats.acorr_ljungbox(arr, lags=1,
                                                      boxpierce=True);
    if bppvalue[0] <= 0.05:
        sign = '<='
    else:
        sign = '>'
    print('p-value = {0} {1} 0.05'.format(bppvalue[0], sign));
```

```
p-value = 0.414583300142219 > 0.05
p-value = 6.407540379506912e-10 <= 0.05
```

Comments * H0: the series is white noise (= null autocorrelation) * Food sales : p value > 0.05 - H0 can not be rejected * Fuel sales : p values <= 0.5 - there is evidence for rejecting H0 * --> The food sales follow a white noise distribution * --> The fuel sales does not follow a white noise distribution

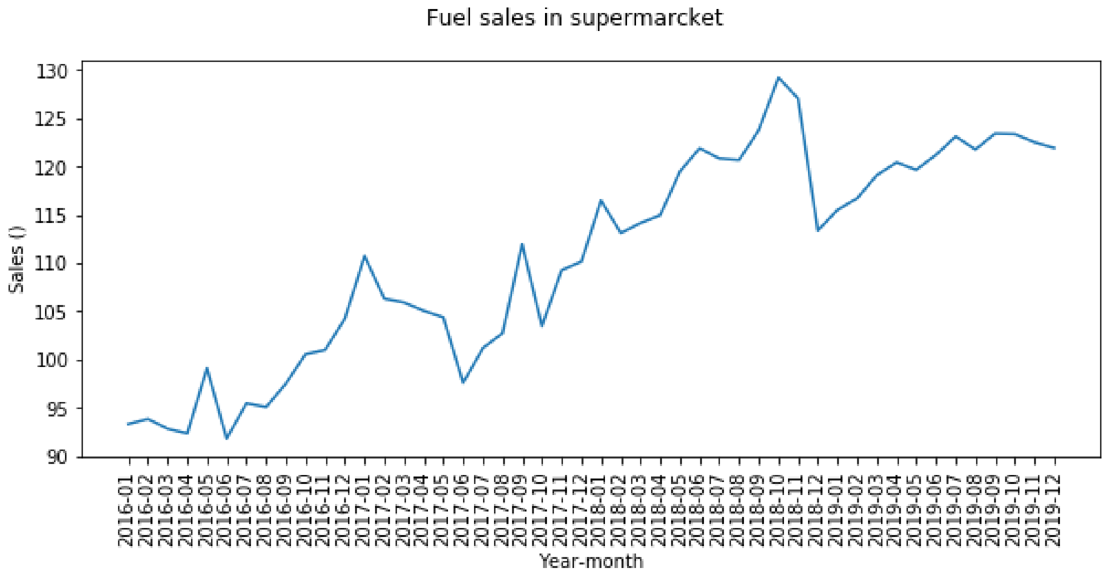
In [46]:
```
#Apply a Shapiro Wilk test on both series
for arr in [food_sales, fuel_sales]:
    _, spval = stats.shapiro(arr)
    if spval <= 0.05:
        sign2 = '<='
    else:
        sign2 = '>'
    spval = round(spval * 1000) / 1000
    print('p-value = {0} {1} 0.05'.format(spval, sign2));
```

```
p-value = 0.243 > 0.05
p-value = 0.01 <= 0.05
```

- H0: the data are drawn from a normal distribution.
- Food sales : p value > 0.05 - H0 can not be rejected
- Fuel sales : p values <= 0.5 - there is evidence for rejecting H0
- The food sales follows a gausian distribution
- The fuel sales does not follow a gaussian distribution

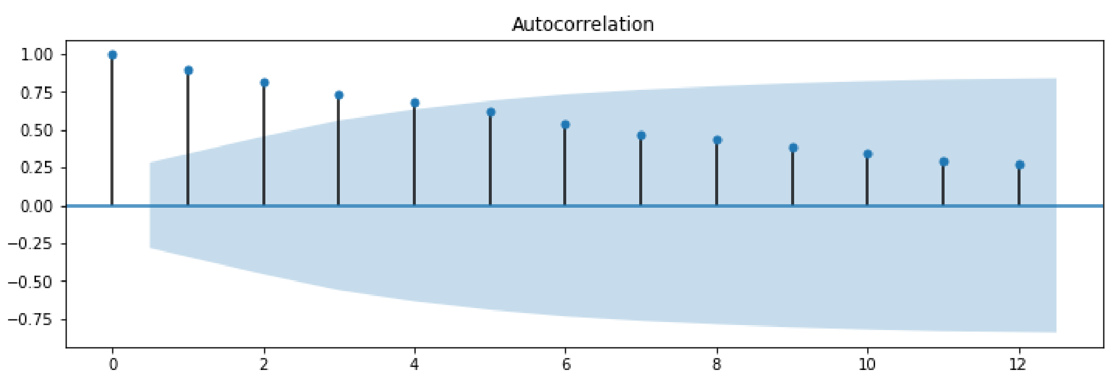
- The food sales can be represented by a gaussian white noise --> stationarity
- The fuel sales cannot be represented by a gaussian white noise --> redo questions 2 to 4 !

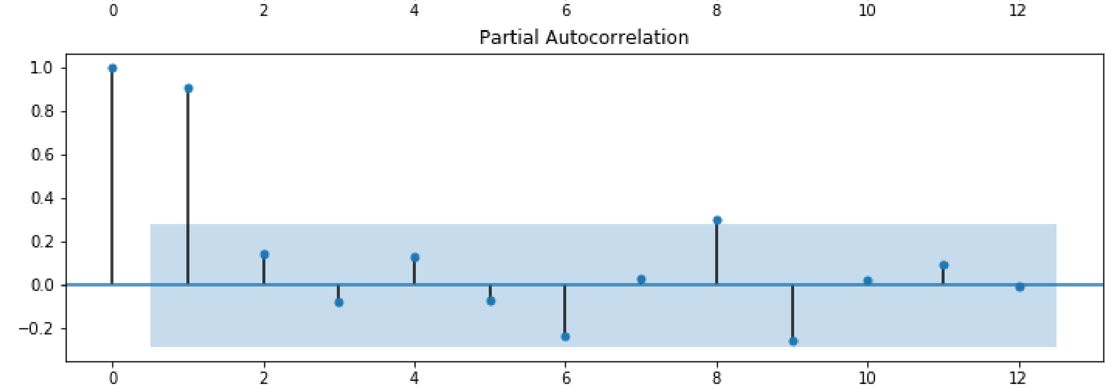
In [47]:
```
## Plot the potentially non stationary series
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(df.loc['2016-01':'2019-12'].index, fuel_sales);
plt.setp(ax.get_xticklabels(), rotation=90)
ax.set_ylabel('Sales ()')
ax.set_xlabel('Year-month')
fig.suptitle('Fuel sales in supermarcket');
```



Comments * The series values are time increasing * The series does not looks stationary

In [48]:
```
#ACF and PACF of fuel sales
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(fuel_sales, lags=12, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(fuel_sales, lags=12, ax=ax2)
```





Comments * The ACF of a non-stationary time series decreases slowly. * The ACF of a non-stationary time series shows many values out of the confidence interval. * The first value of an ACF of a non-stationary time series is often large and positive. * Thus, this time series does not look stationary according to its ACF/PACF plots

```
In [49]: stat_status = stationarity_tests(fuel_sales)
```

```
ADF: p-value = 0.4920682734442802 > 0.05
KPSS: p-value = 0.01 <= 0.05
The series is non stationary.


c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\stattoo
  warn("p-value is smaller than the indicated p-value", InterpolationWarning)
```

- ADF: p value > 0.05 - H0 can not be rejected - the fuel sales may be non-stationary.
- KPSS: p value <= 0.05 - there is evidence for rejecting H0 - the fuel sales are not stationary

- Both tests conclude that fuel sales are not stationary

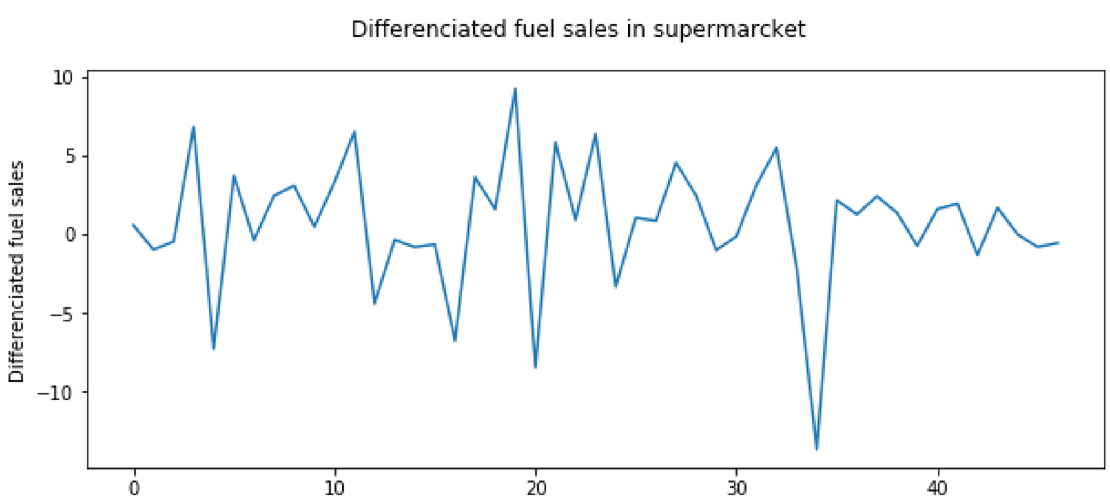
1.1.3 B.2 Influence of differencing on the stationarity

Question 1

```
In [50]: #Differenciate the series
         fuel_sales_d = fuel_sales[1:] - fuel_sales[:len(fuel_sales) - 1]
```

Question 2

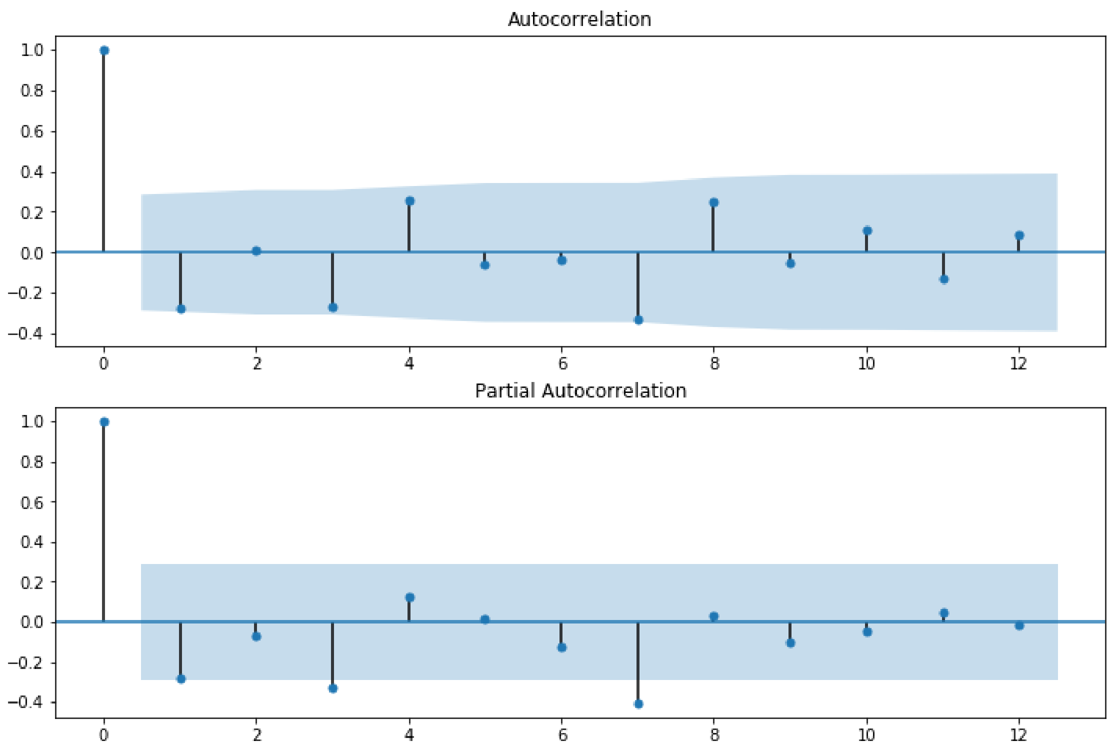
```
In [51]: #Plot the differenciated series
         fig, ax = plt.subplots(figsize=(10, 4))
         ax.plot(fuel_sales_d)
         ax.set_ylabel('Differenciated fuel sales')
         fig.suptitle('Differenciated fuel sales in supermarcket');
```



Comments * Series values seem to be centered * The series looks stationary

Question 3

```
In [52]: #Display the associated ACF and PACF
         fig = plt.figure(figsize=(12,8))
         ax1 = fig.add_subplot(211)
         fig = sm.graphics.tsa.plot_acf(fuel_sales_d, lags=12, ax=ax1)
         ax2 = fig.add_subplot(212)
         fig = sm.graphics.tsa.plot_pacf(fuel_sales_d, lags=12, ax=ax2)
```



Comments * Most ACF/PACF values are in of the confidence interval (blue area). * These plots suggest that the differentiated series is stationary

Question 4

```
In [53]: #Check the stationarity
         stat_status = stationarity_tests(fuel_sales_d)
```

```
ADF: p-value = 7.760535126212442e-07 <= 0.05
KPSS: p-value = 0.1 > 0.05
The series is  stationary.
```

```
c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\statto
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

Conclusion The differentiated fuel sales series is stationary.

1.2 C. ARIMA modelling

1.2.1 B.1 Fuel sales modelling

```
In [54]: #Input parameters
         feature = 'Fuel'
         train_start = '2016-01'
         train_end = '2019-12'
         test_start = '2020-01'
         test_end = '2020-02'
```

Question 1

```
In [55]: #Define the training and test sets
         x_train = df.loc[train_start:train_end][feature].values
         x_test = df.loc[test_start:test_end][feature].values
```

Question 2 **Comments:** * According to the **PACF** plot, **p** is in {1, 8} * According to the **ACF** plot, **q** is in {1, 2, 3, 4}

Question 3

```
In [56]: x_preds = []
         fitted_models = []

         #Try the proposed parameter settings
         for param_id, (p, q) in enumerate(zip([1, 1, 8], [1, 2, 2])):
             try:
                 print('\n p={0}, q={1}'.format(p, q))

                 #Prepare the training of the ARMA model
                 model = ARIMA(x_train, order=(p, 0, q))

                 #Train the ARMA model
                 model_fit = model.fit(disp=0)
                 fitted_models.append(model_fit)

                 #Test the model
                 x_pred = model_fit.forecast(steps=len(x_test))[0]
                 x_preds.append(x_pred)

                 #Get useful metrics
                 print('Standard error: {0}'.format(model_fit.bse[0]))
```

```python
            print('Log-Likelihood function: {0}'.format(model_fit.llf))
            print('AIC score: {0}'.format(model_fit.aic))
            print('BIC score: {0}'.format(model_fit.bic))

            #Update the result vector
            x_preds.append(x_pred)

            #Best score
            if param_id == 0:
                best_bic = model_fit.bic
                best_aic = model_fit.aic
                best_p = p
                best_q = q
            elif model_fit.aic < best_aic and  model_fit.bic < best_bic:
                best_bic = model_fit.bic
                best_aic = model_fit.aic
                best_p = p
                best_q = q
        except:
            pass
    print('\n Best parameter values: (p, q) = ({0}, {1})'.format(best_p, best_q))
```

```
 p=1, q=1
Standard error: 9.539056089308582
Log-Likelihood function: -135.30956264070812
AIC score: 278.61912528141625
BIC score: 286.10392932504783

 p=1, q=2
Standard error: 9.638335543903757
Log-Likelihood function: -135.30475691706795
AIC score: 280.6095138341359
BIC score: 289.9655188886754

 p=8, q=2
Standard error: 8.890392668089305
Log-Likelihood function: -129.2911939449957
AIC score: 282.5823878899914
BIC score: 305.0368000208861

 Best parameter values: (p, q) = (1, 1)
```

Question 4 **Comments** * The BIC criterion value should be as low as possible: p=1 and q=1 * The AIC criterion value should be as low as possible: p=1 and q=1 * Thus, the best model according to the BIC and AIC values is ARMA(1, 1)

Question 5

```
In [57]: #Residuals
         for p, q, model_fit in zip([1, 1, 8], [1, 2, 2], fitted_models):

             #Box-Pierce test
             _, _, _, bppval = sm.stats.acorr_ljungbox(model_fit.resid, lags=12,
                                                        boxpierce=True)
             #pp = 100 * np.sum(bppval < 0.05) / len(bppvalue)
             if bppval[0] <= 0.05:
                 sign1 = '<='
             else:
                 sign1 = '>'
             bppval = round(bppval[0] * 1000) / 1000

             #Shapiro Wilk test
             W, spval = stats.shapiro(model_fit.resid)
             if spval <= 0.05:
                 sign2 = '<='
             else:
                 sign2 = '>'
             spval = round(spval * 100000) / 100000

             #Check the stationarity
             stat_status = stationarity_tests(x)

             #Display results
             print('p={0}, q={1}: *Box-Pierce: p-value = {2}{3}0.05%'
                   '\n       *Shapiro-Wilk pvalue = {4}{5}0.05%\n'.format(p, q,
                                                                       bppval,
                                                                       sign1,
                                                                       spval,
                                                                       sign2))



             #ACF and PACF of fuel sales
             fig = plt.figure(figsize=(12,8))
             ax1 = fig.add_subplot(211)
             fig = sm.graphics.tsa.plot_acf(model_fit.resid, lags=12, ax=ax1)
             ax2 = fig.add_subplot(212)
             fig = sm.graphics.tsa.plot_pacf(fuel_sales, lags=12, ax=ax2)
```

```
ADF: p-value = 0.0 <= 0.05
KPSS: p-value = 0.1 > 0.05
The series is  stationary.
p=1, q=1: *Box-Pierce: p-value = 0.855>0.05%
       *Shapiro-Wilk pvalue = 0.00013<=0.05%

ADF: p-value = 0.0 <= 0.05
```

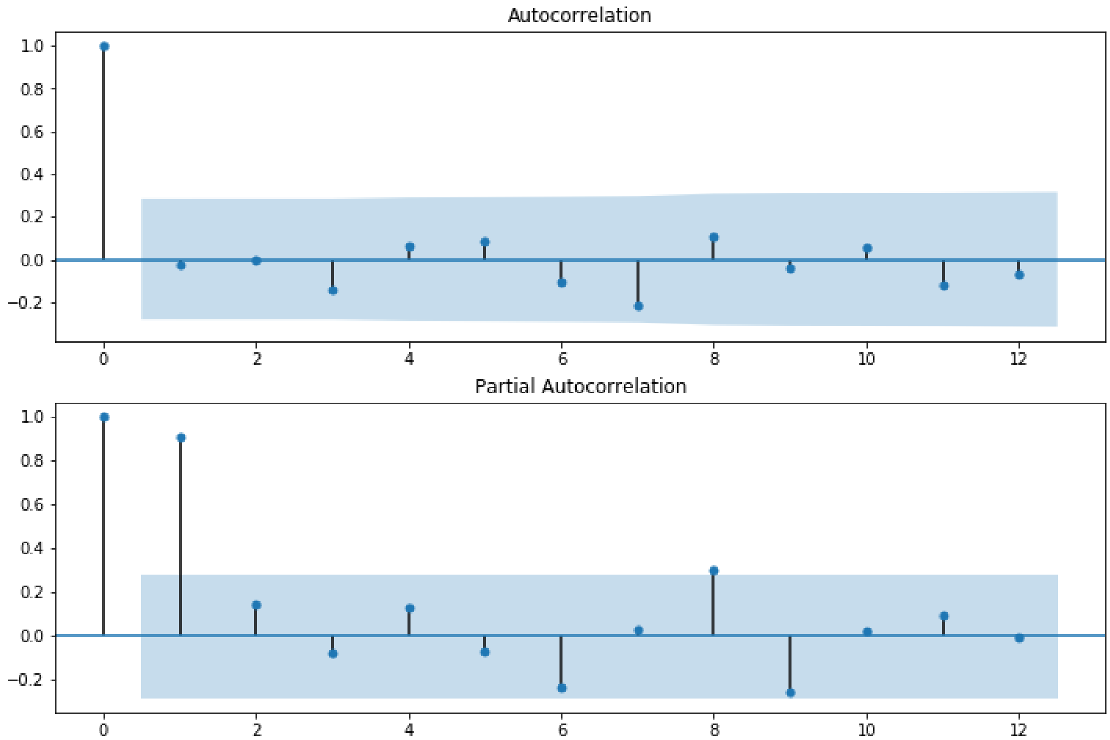
```
c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\stattoo
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

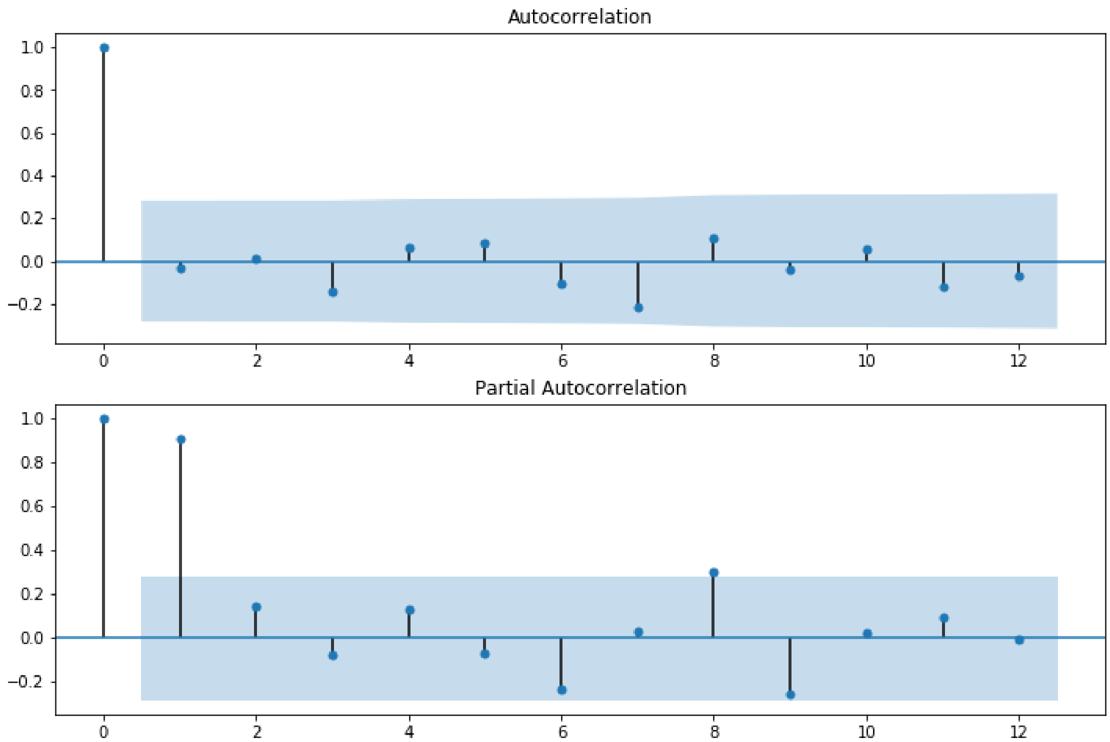
```
KPSS: p-value = 0.1 > 0.05
The series is  stationary.
p=1, q=2: *Box-Pierce: p-value = 0.839>0.05%
       *Shapiro-Wilk pvalue = 0.00013<=0.05%

ADF: p-value = 0.0 <= 0.05
```

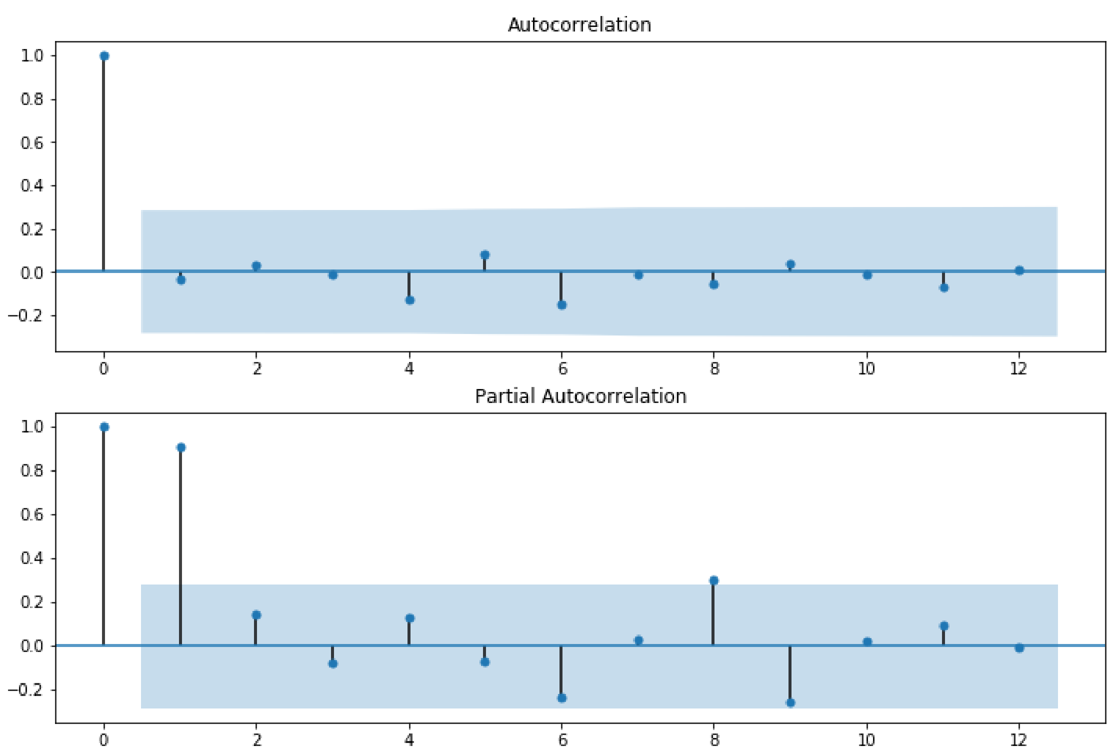
```
c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\stattoo
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

```
KPSS: p-value = 0.1 > 0.05
The series is  stationary.
p=8, q=2: *Box-Pierce: p-value = 0.804>0.05%
       *Shapiro-Wilk pvalue = 3e-05<=0.05%
```

```
c:\users\helen\appdata\local\programs\python\python37\lib\site-packages\statsmodels\tsa\stattoo
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```




Autocorrelation
Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8
10
12

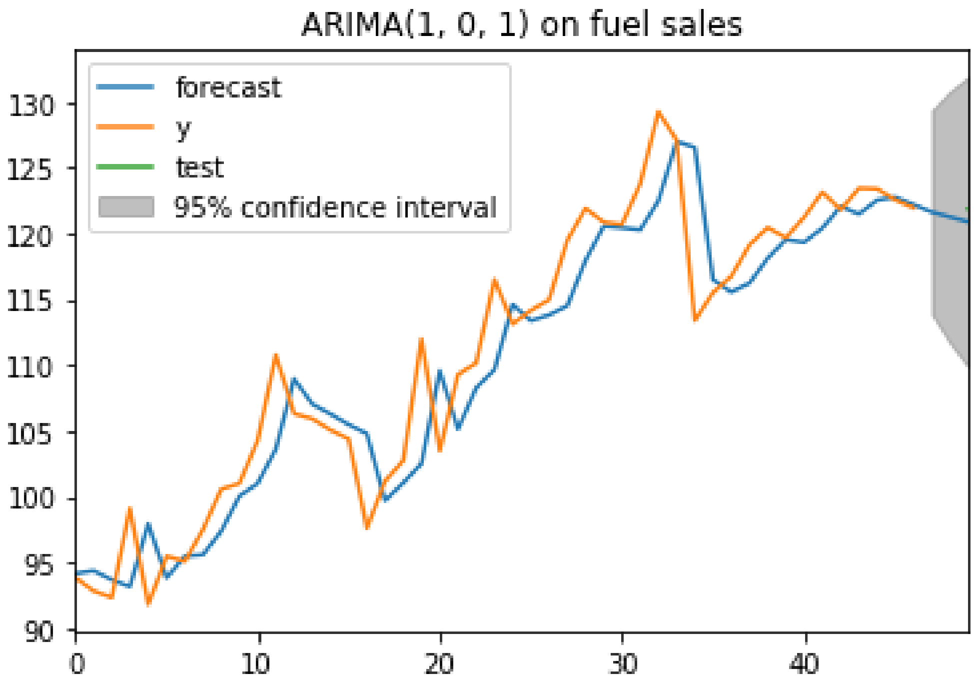

Autocorrelation
Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8
10
12

Comments: * According to the Box-Pierce test, all the residuals are white noise * According to the Shapiro-Wilk test, all the residuals are not gaussian * We cannot conclude directly on the stationarity of these residuals using our previous answer * According to the ADF and KPSS tests, all the residuals are stationary * (p, q) = (1, 1) is the best choice because it gives the highest p-values

Question 6

```
In [58]: #Choose the best model
         model = ARIMA(x_train, order=(1, 0, 1))

         #Train the ARMA model
         model_fit = model.fit(disp=0)

         #Test the model
         x_pred = model_fit.forecast(steps=len(x_test))[0]

         #Compare visually the predicted and true sale vales
         plt.figure();
         model_fit.plot_predict(1, len(x_train) + len(x_test));
         plt.plot(range(len(x_train)+1, len(x_train)+1 + len(x_test)), x_test,
                  label="test");
         plt.title("ARIMA(1, 0, 1) on fuel sales");
         plt.legend();
         plt.show();

         #Compute the RMSE
         rmse = np.sqrt(np.mean((x_pred - x_test)**2))
         print('RMSE from 2019-01 to 2019-02: {0}'.format(rmse))
```

```
<Figure size 432x288 with 0 Axes>
```

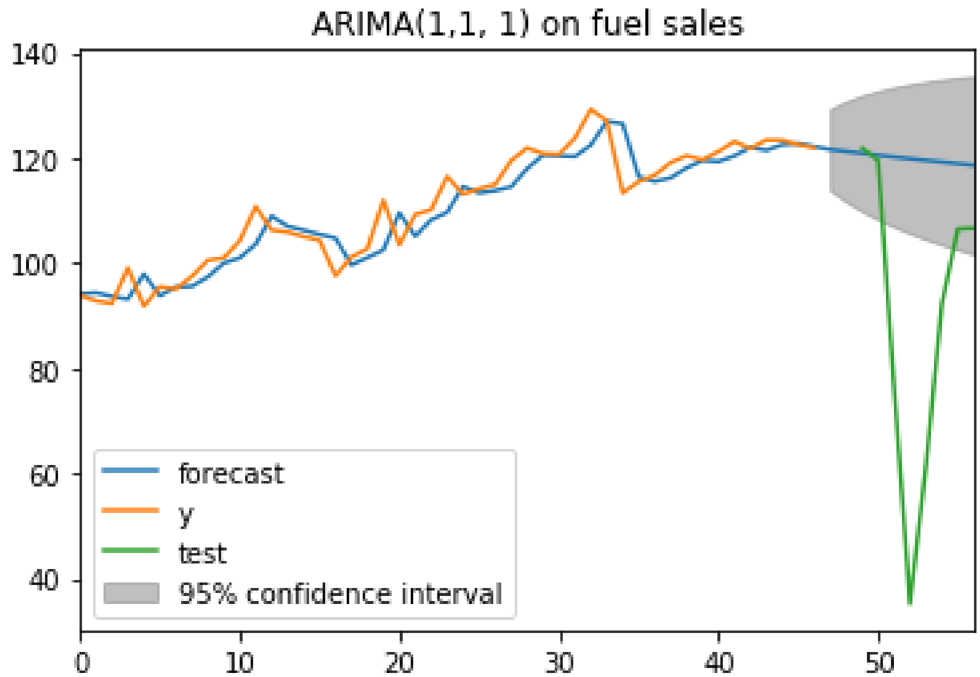



```
RMSE from 2019-01 to 2019-02: 1.2170914135744542
```

Question 7

```
In [59]: #Modify the test values
         test_start = '2020-01'
         test_end = '2020-09'
         x_test = df.loc[test_start:test_end][feature].values

         #Test the model
         x_pred = model_fit.forecast(steps=len(x_test))[0]

         #Compare visually the predicted and true sale vales
         plt.figure();
         model_fit.plot_predict(1, len(x_train) + len(x_test));
         plt.plot(range(len(x_train)+1, len(x_train)+1 + len(x_test)), x_test,
                  label="test");
         plt.title("ARIMA(1,1, 1) on fuel sales");
         plt.legend();
         plt.show();

         #Compute the RMSE
         rmse = np.sqrt(np.mean((x_pred - x_test)**2))
         print('RMSE from 2019-01 to 2019-09: {0}'.format(rmse))
```

```
<Figure size 432x288 with 0 Axes>
```




```
RMSE from 2019-01 to 2019-09: 39.40830837046906
```