In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#Importing data
df = pd.read_csv(r'F:\Prthon Programming\Time Series Modelling\TCS.csv')
#Printing head
df.head()
```

Out[18]:

	Date	Close
0	8/31/2004	988
1	9/30/2004	1027
2	10/31/2004	1154
3	11/30/2004	1275
4	12/31/2004	1336

In [19]:
```
df.shape
```

Out[19]: (181, 2)

In [20]:
```
from datetime import datetime
df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)
df = df.set_index(['Date'])
```

In [21]:
```
df.head()
```

Out[21]:

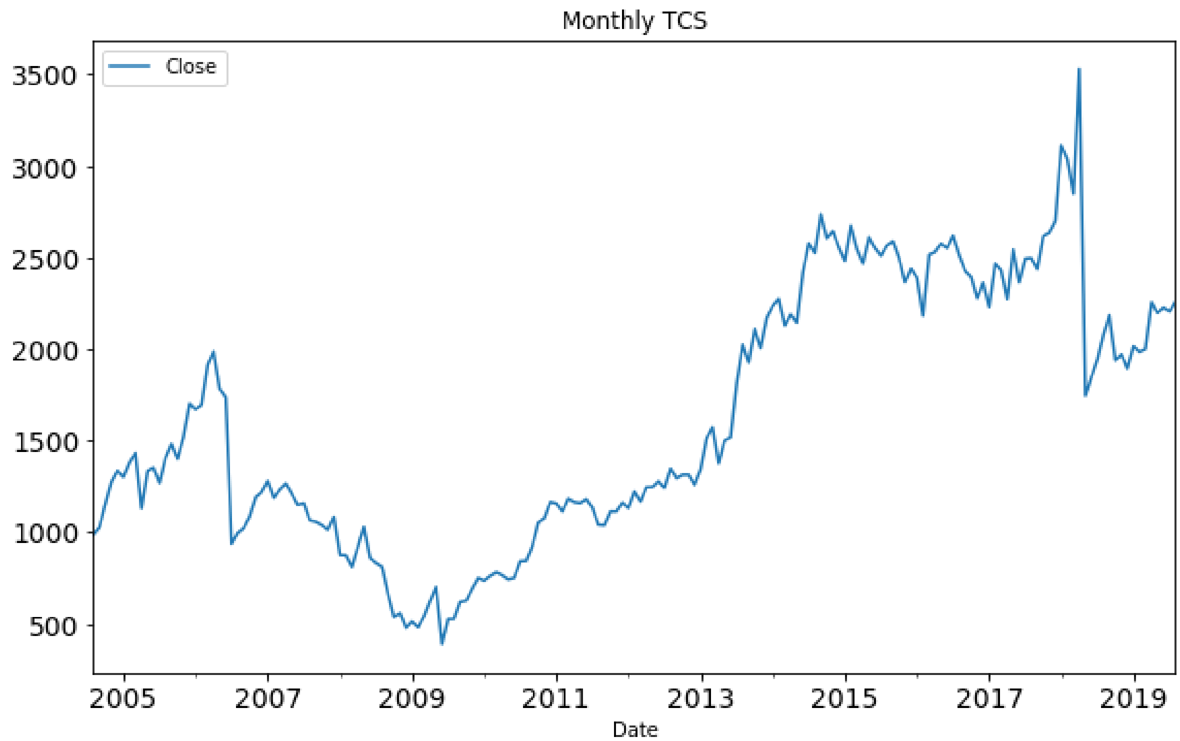
Date	Close
2004-08-31	988
2004-09-30	1027
2004-10-31	1154
2004-11-30	1275
2004-12-31	1336

In [22]:
```
df.tail()
```

Out[22]:

	Close
Date	
2019-04-30	2255
2019-05-31	2197
2019-06-30	2227
2019-07-31	2205
2019-08-31	2258

In [23]:
```
df.plot(figsize=(10,6), title= 'Monthly TCS', fontsize=14)
plt.show()
```

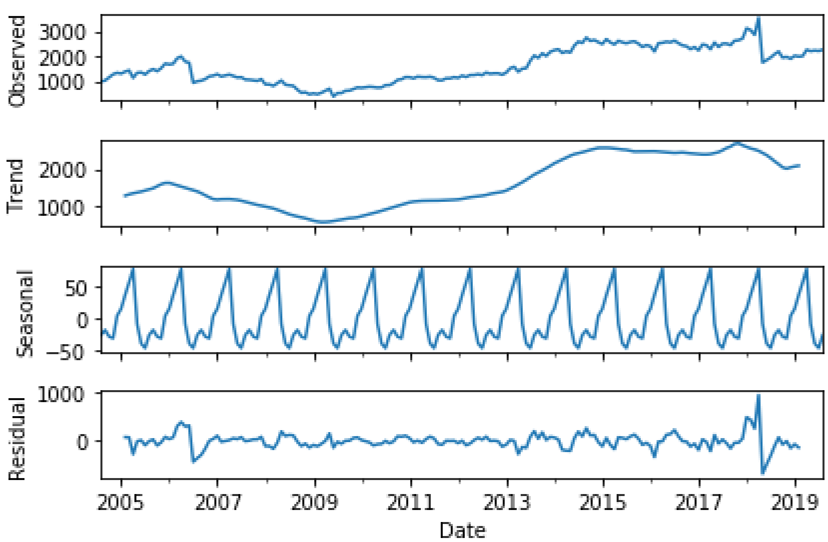


In [25]:
```
#checking stationarity
from statsmodels.tsa.stattools import adfuller

result = adfuller(df.Close)
print('ADF Statistic:',result[0])
print('p-value: %f' %result[1])


print("The test statistic is positive, meaning we are much less likely to reje
ct the null hypothesis (it looks non-stationary). Comparing the test statistic
to the critical values, it looks like we would have to fail to reject the null
hypothesis that the time series is non-stationary and does have time-dependent
structure.")
```

```
ADF Statistic: -1.040636747556359
p-value: 0.738156
The test statistic is positive, meaning we are much less likely to reject the
null hypothesis (it looks non-stationary). Comparing the test statistic to th
e critical values, it looks like we would have to fail to reject the null hyp
othesis that the time series is non-stationary and does have time-dependent s
tructure.
```

In [111]:
```
#"""f=y.diff( periods= 1)
#f.plot(figsize=(10, 6))
#plt.show()
#"""
```

In [26]:
```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(df.Close).plot()
plt.show()
```



In [27]:
```
import itertools
p = d = q = range(0, 3)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q
))]

print('SARIMAX:',pdq[1],'x', seasonal_pdq[0])
```

```
SARIMAX: (0, 0, 1) x (0, 0, 0, 12)
```

When looking to fit time series data with a seasonal ARIMA model,

Our first goal is to find the values of ARIMA(p,d,q)(P,D,Q)s that optimize a metric of interest. There are many guidelines and best practices to achieve this goal, yet the correct parametrization of ARIMA models can be a painstaking manual process that requires domain expertise and time. Other statistical programming languages such as R provide automated ways to solve this issue, but those have yet to be ported over to Python. In this section, we will resolve this issue by writing Python code to programmatically select the optimal parameter values for our ARIMA(p,d,q)(P,D,Q)s time series model.

We will use a “grid search” to iteratively explore different combinations of parameters. For each combination of parameters, we fit a new seasonal ARIMA model with the SARIMAX() function from the statsmodels module and assess its overall quality. Once we have explored the entire landscape of parameters, our optimal set of parameters will be the one that yields the best performance for our criteria of interest. Let’s begin by generating the various combination of parameters that we wish to assess:

In [28]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q
))]

print('Examples of parameter combinations for Seasonal ARIMA…')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA…
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [29]:
```
import warnings
import itertools
import statsmodels.api as sm

warnings.filterwarnings("ignore") # specify to ignore warning messages

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(df,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, result
s.aic))
        except:
            continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:3205.138857101255
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:2808.2322203024332
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:2524.494562904066
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:2346.5376737442
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:2541.4046194154653
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:2528.4728284085054
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:2361.4152304948366
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:2348.56778404571
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:2968.013657325264
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:2636.0813577071385
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:2425.531528955507
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:2247.3070324857913
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:2455.934781901293
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:2424.463270149163
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:2274.434395551468
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:2248.4100323449516
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:2393.338501306628
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:2240.264600483128
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:2365.8021866745703
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:2103.9192336672854
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:2253.118324112525
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:2240.2749051095807
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:2133.102929920741
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:2104.458566259088
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:2366.311893117101
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:2215.355370494454
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:2334.125981711975
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:2074.87067039067
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:2240.5937638279947
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:2217.4610625451583
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:2120.406065027957
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:2076.2830736843202
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:2407.659400955427
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:2254.8163893402193
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:2361.983368561636
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:2115.276827500802
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:2255.123594051364
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:2254.9465555125607
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:2127.9375152032294
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:2115.4303305632347
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:2381.0813564598902
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:2229.9975812240036
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:2340.5821390310225
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:2088.1802639715006
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:2242.509689659515
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:2229.558561656366
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:2119.1573708833357
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:2088.707403592326
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:2378.789535127041
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:2227.3272453591485
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:2347.128789802009
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:2087.389741183601
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:2227.413629246639
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:2229.399967335284
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:2105.324733022343
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:2088.4943490458677
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:2367.7380223557957
```

```
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:2216.794326092174
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:2335.147490140809
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:2076.120013003274
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:2229.2798058958674
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:2218.961140551869
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:2107.2478370314443
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:2077.452857769408
```

Using grid search,

we have identified the set of parameters that produces the best fitting model to our time series data. We can proceed to analyze this particular model in more depth. We'll start by plugging the optimal parameter values into a new SARIMAX model:

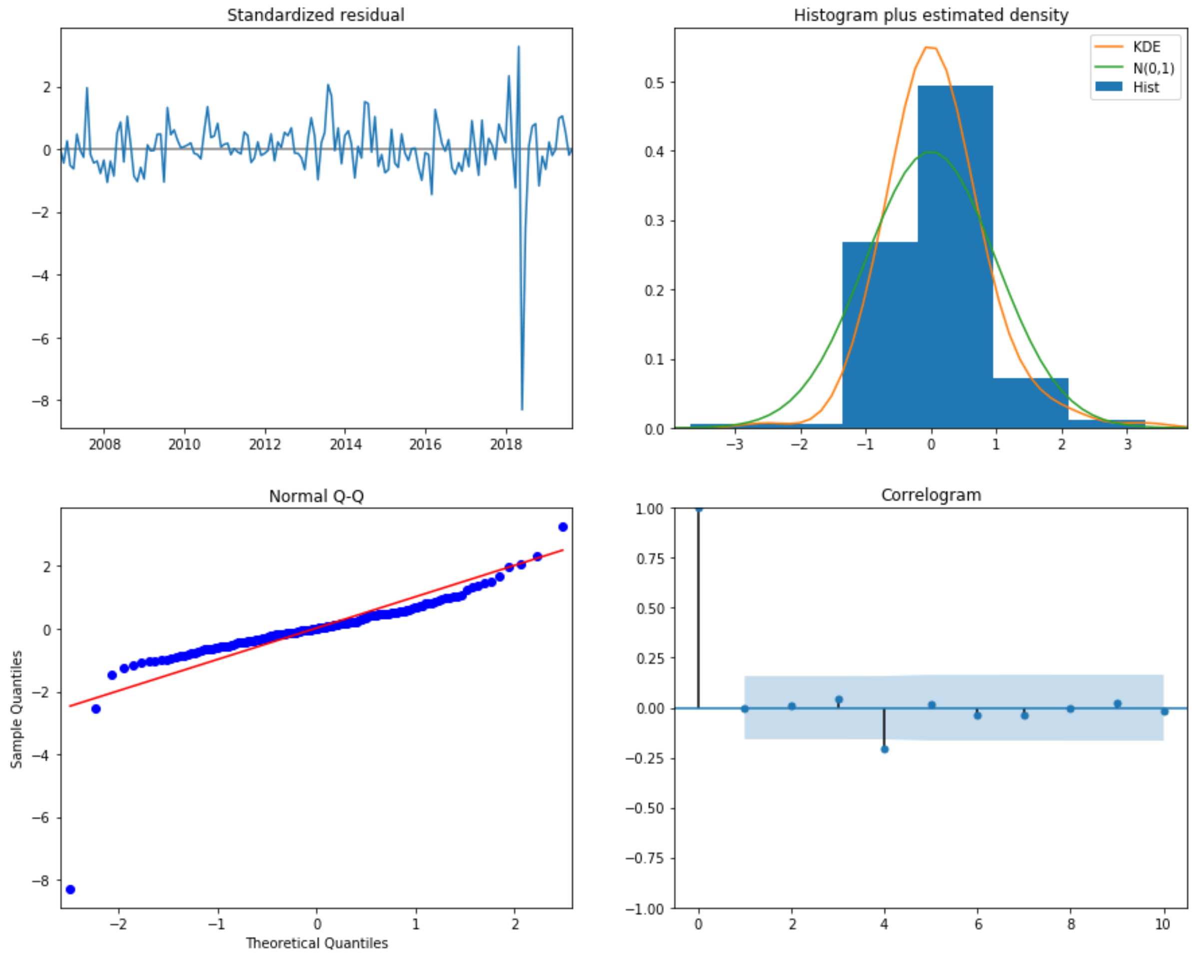
In [30]:
```
mod = sm.tsa.statespace.SARIMAX(df,
                                order=(1, 1, 1),
                                seasonal_order=(0, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])

print("The summary attribute that results from the output of SARIMAX returns a significant amount of information, but we’ll focus our attention on the table of coefficients. The coef column shows the weight (i.e. importance) of each feature and how each one impacts the time series. The P>|z| column informs us of the significance of each feature weight. Here, each weight has a p-value lower or close to 0.05, so it is reasonable to retain all of them in our model.")
```

```
==============================================================================
=
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
ar.L1         -0.2074      0.389     -0.533      0.594      -0.970       0.55
5
ma.L1         -0.1496      0.441     -0.339      0.735      -1.014       0.71
5
ma.S.L12      -0.8601      0.102     -8.468      0.000      -1.059      -0.66
1
sigma2      3.757e+04   2217.058     16.945      0.000    3.32e+04    4.19e+0
4
==============================================================================
=
The summary attribute that results from the output of SARIMAX returns a signi
ficant amount of information, but we’ll focus our attention on the table of c
oefficients. The coef column shows the weight (i.e. importance) of each featu
re and how each one impacts the time series. The P>|z| column informs us of t
he significance of each feature weight. Here, each weight has a p-value lower
or close to 0.05, so it is reasonable to retain all of them in our model.
```

When fitting seasonal ARIMA models (and any other models for that matter),

it is important to run model diagnostics to ensure that none of the assumptions made by the model have been violated. The plot_diagnostics object allows us to quickly generate model diagnostics and investigate for any unusual behavior.

In [31]:
```
results.plot_diagnostics(figsize=(15, 12))
plt.show()

print("Our primary concern is to ensure that the residuals of our model are un
correlated and normally distributed with zero-mean. If the seasonal ARIMA mode
l does not satisfy these properties, it is a good indication that it can be fu
rther improved.In this case, our model diagnostics suggests that the model res
iduals are normally distributed based on the following: In the top right plot,
we see that the red KDE line follows closely with the N(0,1) line (where N(0,
1)) is the standard notation for a normal distribution with mean 0 and standar
d deviation of 1). This is a good indication that the residuals are normally d
istributed.The qq-plot on the bottom left shows that the ordered distribution
 of residuals (blue dots) follows the linear trend of the samples taken from a
standard normal distribution with N(0, 1). Again, this is a strong indication
 that the residuals are normally distributed. The residuals over time (top lef
t plot) don’t display any obvious seasonality and appear to be white noise. Th
is is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom r
ight, which shows that the time series residuals have low correlation with lag
ged versions of itself.")
```

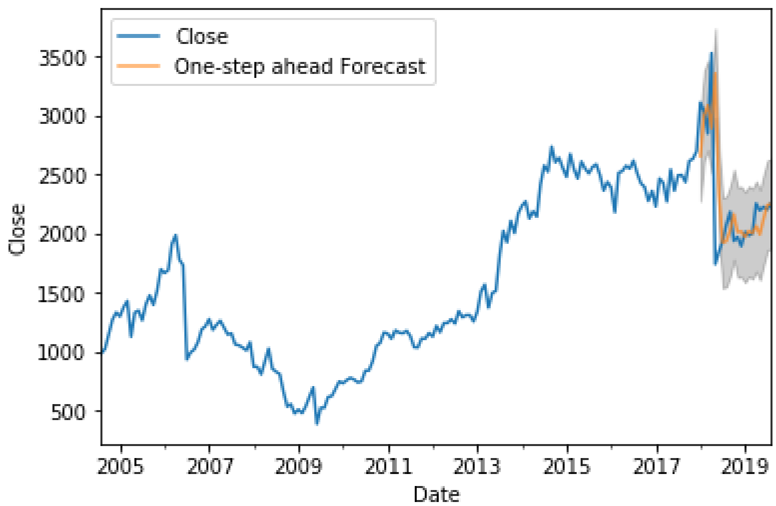


Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.In this case, our model diagnostics suggests that the model residuals are normally distributed based on the following: In the top right plot, we see that the red KDE line follows closely with the N(0,1) line (where N(0,1)) is the standard notation for a normal distribution with mean 0 and standard deviation of 1). This is a good indication that the residuals are normally distributed.The qq-plot on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this is a strong indication that the residuals are normally distributed. The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. This is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.

In [33]:
```
pred = results.get_prediction(start=pd.to_datetime('2018-01-31'), dynamic=False)
plt.figure(figsize=(10,6))
pred_ci = pred.conf_int()
```

```
<Figure size 720x432 with 0 Axes>
```

In [34]:
```
ax = df['2004':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Close')
plt.legend()

plt.show()

print("Overall, our forecasts align with the true values very well, showing an
overall increase trend.")
```




```
Overall, our forecasts align with the true values very well, showing an overa
ll increase trend.
```

In [36]:
```
y_forecasted = pred.predicted_mean
y_truth = df['2018-01-31':]

from sklearn.metrics import mean_squared_error
from math import sqrt

# Compute the mean square error
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))

print("An MSE of 0 would that the estimator is predicting observations of the
 parameter with perfect accuracy, which would be an ideal scenario but it not
 typically possible.")
```

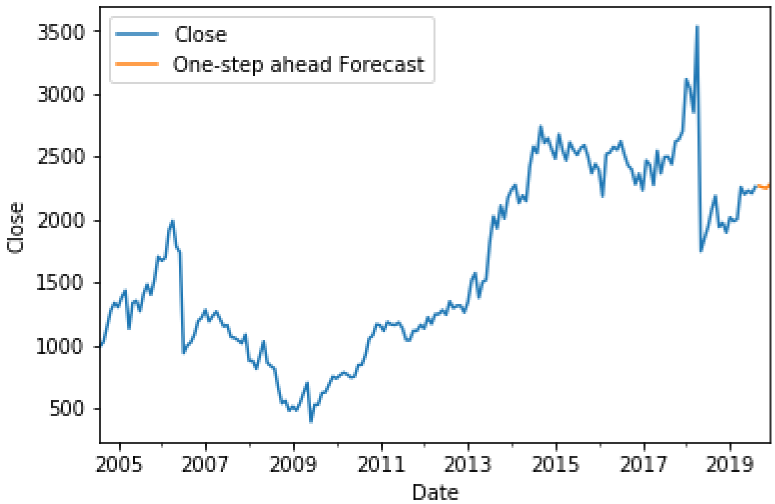
```
The Mean Squared Error of our forecasts is 2018-01-31 00:00:00   NaN
2018-02-28 00:00:00   NaN
2018-03-31 00:00:00   NaN
2018-04-30 00:00:00   NaN
2018-05-31 00:00:00   NaN
2018-06-30 00:00:00   NaN
2018-07-31 00:00:00   NaN
2018-08-31 00:00:00   NaN
2018-09-30 00:00:00   NaN
2018-10-31 00:00:00   NaN
2018-11-30 00:00:00   NaN
2018-12-31 00:00:00   NaN
2019-01-31 00:00:00   NaN
2019-02-28 00:00:00   NaN
2019-03-31 00:00:00   NaN
2019-04-30 00:00:00   NaN
2019-05-31 00:00:00   NaN
2019-06-30 00:00:00   NaN
2019-07-31 00:00:00   NaN
2019-08-31 00:00:00   NaN
Close                 NaN
dtype: float64
An MSE of 0 would that the estimator is predicting observations of the parame
ter with perfect accuracy, which would be an ideal scenario but it not typica
lly possible.
```

In [37]:
```
y_forecasted = pred.predicted_mean
y_truth = df['2018-01-31':]

from sklearn.metrics import mean_squared_error
from math import sqrt
rms = sqrt(mean_squared_error(y_truth,y_forecasted))
print(rms)
```

```
431.6427161331798
```

In [38]:
```
pred_uc = results.get_forecast(steps=4)
plt.figure(figsize=(10,6))
ax = df['2004':].plot(label='observed')
pred_uc.predicted_mean.plot(ax=ax, label='One-step ahead Forecast')
ax.set_xlabel('Date')
ax.set_ylabel('Close')
plt.legend()
plt.show()
```

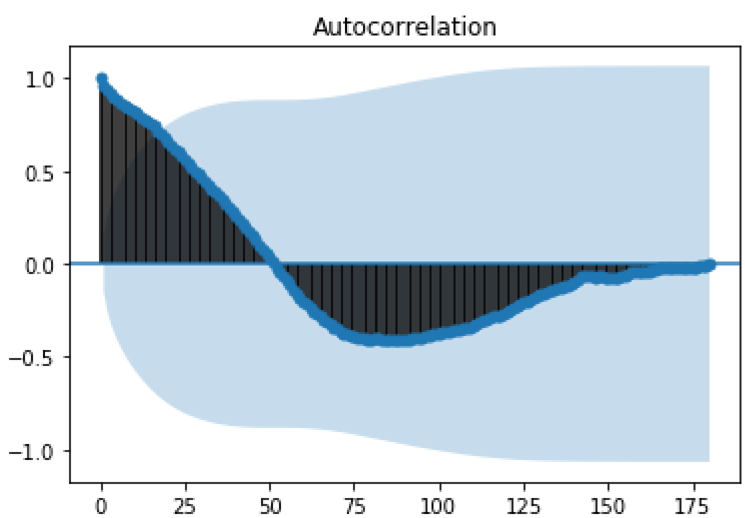
```
<Figure size 720x432 with 0 Axes>
```



In [39]:
```
pred_uc.predicted_mean
```

Out[39]:
```
2019-09-30    2264.788278
2019-10-31    2252.946399
2019-11-30    2245.645824
2019-12-31    2277.826494
Freq: M, dtype: float64
```

In [40]:
```
import statsmodels
a=statsmodels.graphics.tsaplots.plot_acf(df)
```



In [41]:
```
b=statsmodels.graphics.tsaplots.plot_pacf(df)
```