In [1]:

```
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\Nifty50.csv')
#Printing head
df.head()
```

Out[1]:

	Date	Close
0	9/1/2007	5021
1	10/1/2007	5901
2	11/1/2007	5763
3	12/1/2007	6139
4	1/1/2008	5137

In [2]:

```
df.shape
```

Out[2]: (144, 2)

In [5]:

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

In [6]:

```
df.head()
```

Out[6]:

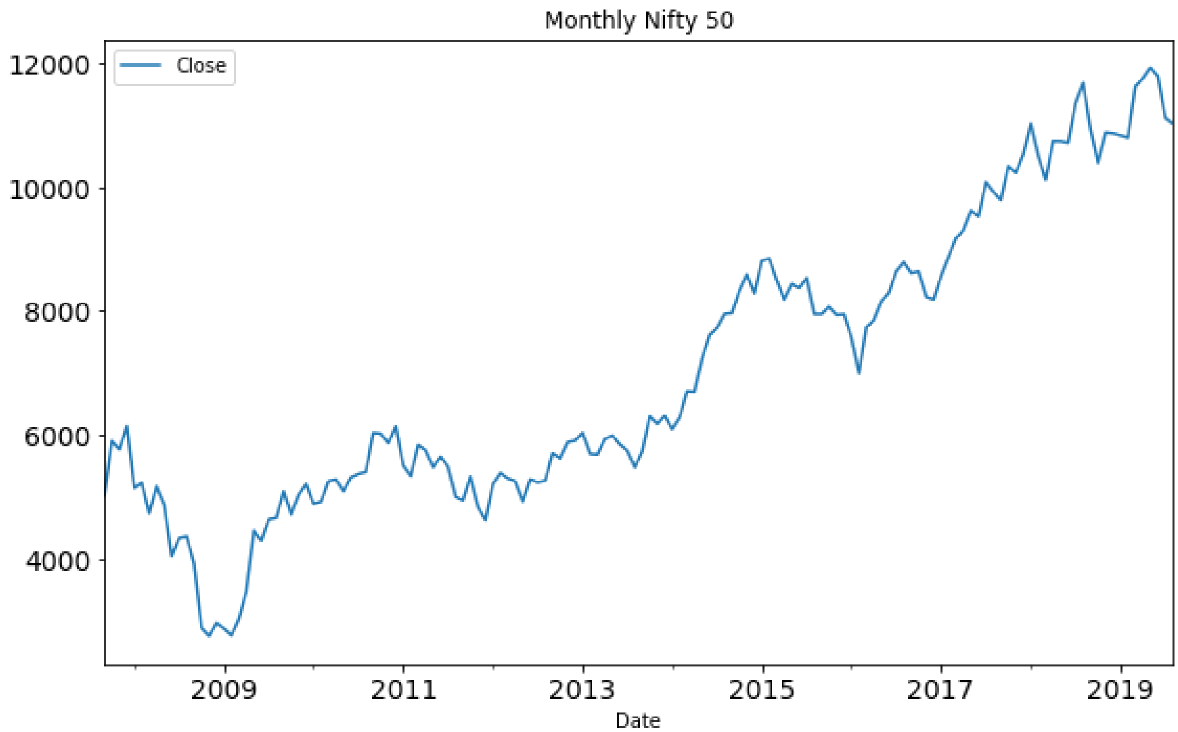
	Close
Date	
2007-09-01	5021
2007-10-01	5901
2007-11-01	5763
2007-12-01	6139
2008-01-01	5137

In [7]:
```
df.tail()
```

Out[7]:

	Close
Date	
2019-04-01	11748
2019-05-01	11923
2019-06-01	11789
2019-07-01	11118
2019-08-01	11023

In [11]:
```
df.plot(figsize=(10,6), title= 'Monthly Nifty 50', fontsize=14)
plt.show()
```

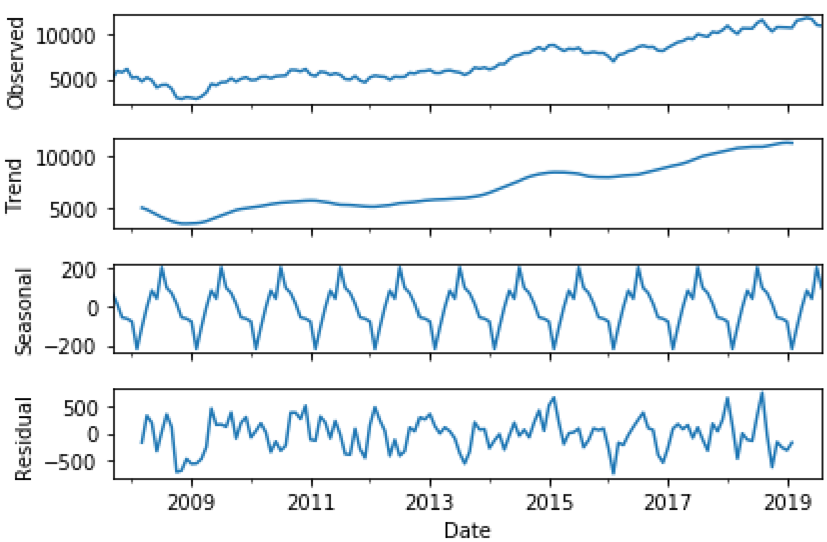


In [18]:
```
#checking stationarity
from statsmodels.tsa.stattools import adfuller

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: 0.06797278240531189
p-value: 0.963805
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 [19]:
```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(df.Close).plot()
plt.show()
```



In [21]:
```
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 [66]:
```
# 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 [72]:
```
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:2956.799556421978
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:2645.023207985234
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:2252.499091916189
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:2031.2947196433176
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:2248.0505828373084
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:2132.307238038289
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:2048.003248587142
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:2028.6173705945655
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:2836.253361019642
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:2577.5869886540822
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:2104.1775070641816
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1895.4369735379707
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:2132.030324316658
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:2003.179674404164
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1927.8126928339989
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1887.6883185574904
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:2074.628464883538
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1882.8020906791603
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1983.4777279908787
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1734.3424103576583
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:1904.5843688943396
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:1884.7961408591736
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:1771.071214940701
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:1734.3672479618908
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:2062.69128442083
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1870.6699785309302
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:1971.04508508612
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:1726.1485791455004
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:1906.551743143783
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:1872.4692147435592
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:1772.5047744840058
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:1723.6721764017022
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:2094.7367326549333
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:1906.7365042085667
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:2008.5020313782234
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:1751.5007059171949
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:1904.5123957797805
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:1906.0387241106625
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:1770.3271235815546
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:1752.3112140228438
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:2077.024129088607
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:1886.7077481572514
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:1978.7330072754307
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:1744.2918578185672
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:1905.0803402414708
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1883.2959056032414
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:1772.2580002253617
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:1741.650122048662
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:2076.4179566696343
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:1884.7745624062673
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:1985.328803103961
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:1734.6118337068185
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:1884.7801675057522
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:1886.7632858830939
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:1758.0905626688789
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:1734.388038933478
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:2058.4532848360586
```

```
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:1871.9579079786836
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:1967.749649362732
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:1720.4523582829731
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:1885.6681552514592
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:1872.9135096418806
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:1756.2826079018453
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:1723.5892326899593
```

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:

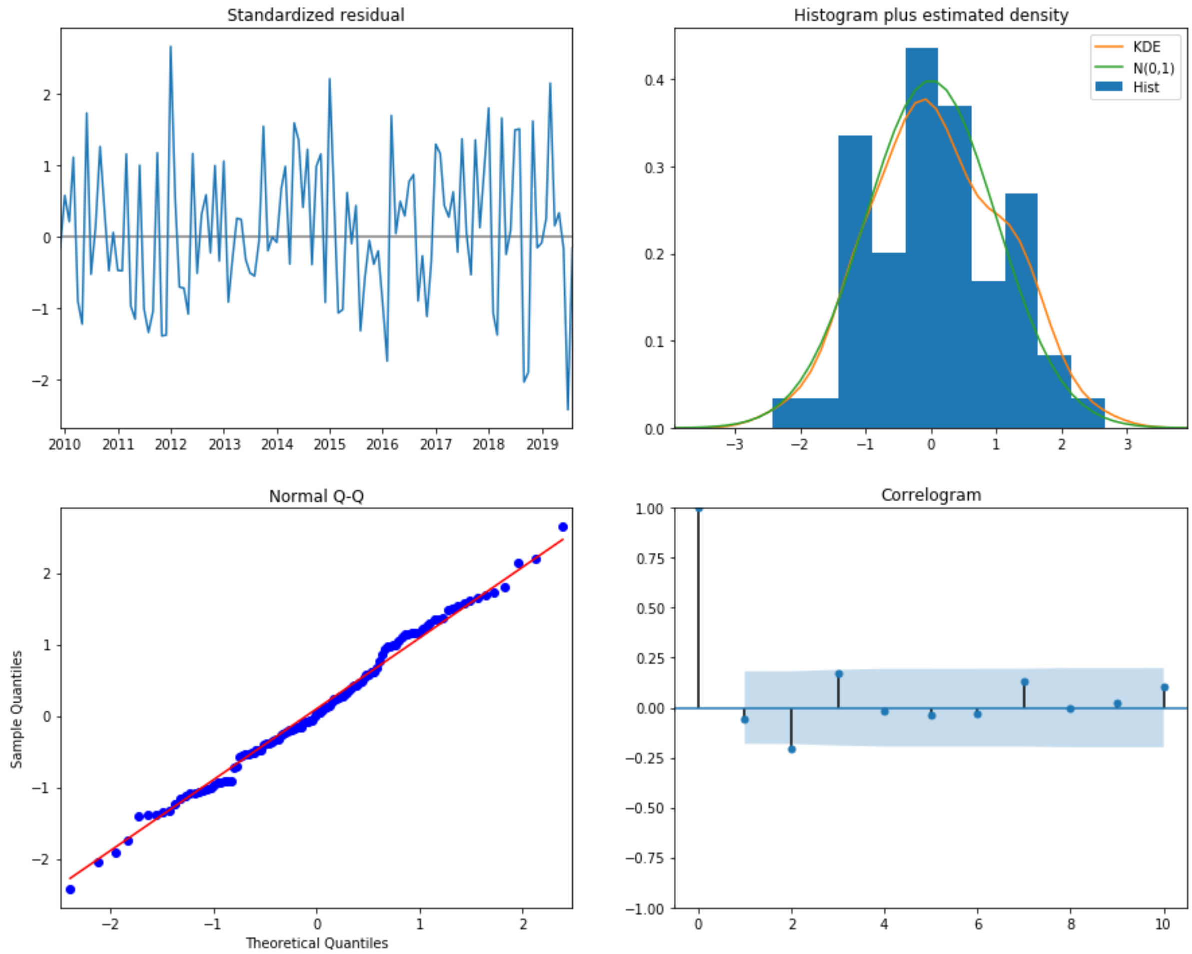
In [81]:
```
mod = sm.tsa.statespace.SARIMAX(df,
                                order=(1, 1, 1),
                                seasonal_order=(0, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

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          0.8992      0.067     13.478      0.000       0.768       1.03
0
ma.L1         -0.9926      0.225     -4.421      0.000      -1.433      -0.55
3
ma.S.L12      -1.1434      0.160     -7.157      0.000      -1.457      -0.83
0
sigma2      9.105e+04   3.06e+04      2.974      0.003    3.11e+04    1.51e+0
5
==============================================================================
=
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 [84]:
```
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.")
```



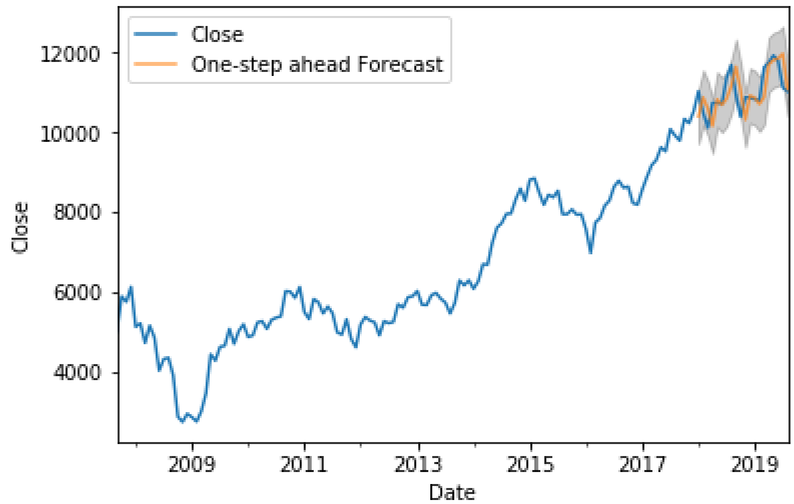
Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.In this case, our model diagnostics suggests that the model 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 [91]:

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

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

In [110]:
```
ax = df['2007':].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 [95]:
```
y_forecasted = pred.predicted_mean
y_truth = df['2018-01-01':]

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.")
```

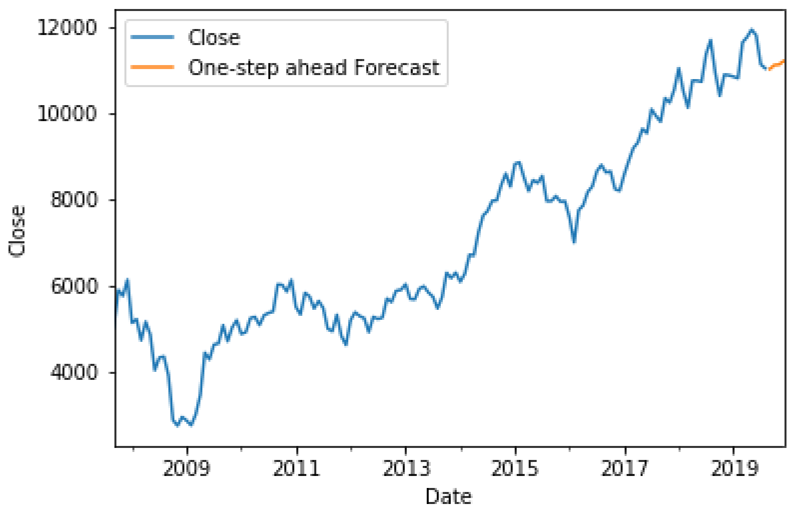
```
The Mean Squared Error of our forecasts is 2018-01-01 00:00:00   NaN
2018-02-01 00:00:00   NaN
2018-03-01 00:00:00   NaN
2018-04-01 00:00:00   NaN
2018-05-01 00:00:00   NaN
2018-06-01 00:00:00   NaN
2018-07-01 00:00:00   NaN
2018-08-01 00:00:00   NaN
2018-09-01 00:00:00   NaN
2018-10-01 00:00:00   NaN
2018-11-01 00:00:00   NaN
2018-12-01 00:00:00   NaN
2019-01-01 00:00:00   NaN
2019-02-01 00:00:00   NaN
2019-03-01 00:00:00   NaN
2019-04-01 00:00:00   NaN
2019-05-01 00:00:00   NaN
2019-06-01 00:00:00   NaN
2019-07-01 00:00:00   NaN
2019-08-01 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 [96]:
```
y_forecasted = pred.predicted_mean
y_truth = df['2018-01-01':]

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

```
461.4659449113668
```

In [106]:
```
pred_uc = results.get_forecast(steps=4)
plt.figure(figsize=(10,6))
ax = df['2007':].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()
```

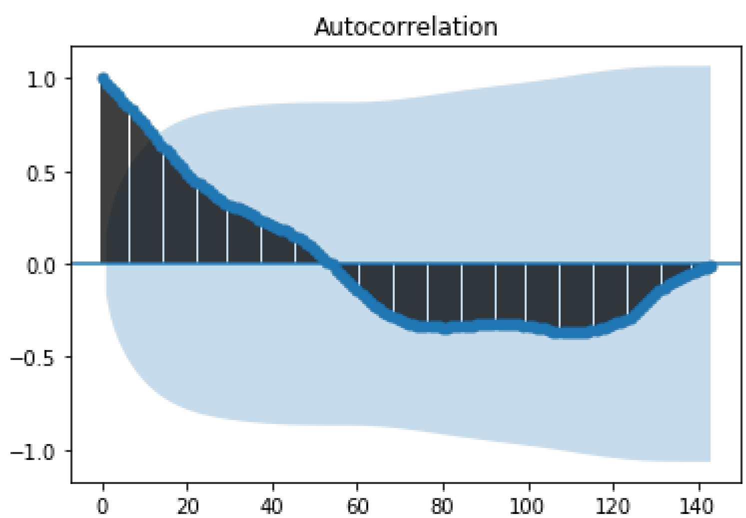
```
<Figure size 720x432 with 0 Axes>
```



In [107]:
```
pred_uc.predicted_mean
```

Out[107]:
```
2019-09-01    11007.658426
2019-10-01    11099.091625
2019-11-01    11109.250354
2019-12-01    11189.140369
Freq: MS, dtype: float64
```

In [108]:
```
import statsmodels
a=statsmodels.graphics.tsaplots.plot_acf(df)
```



In [109]:
```
b=statsmodels.graphics.tsaplots.plot_pacf(df)
```