```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.api import qqplot
from sklearn.metrics import mean_squared_error as mse
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
```

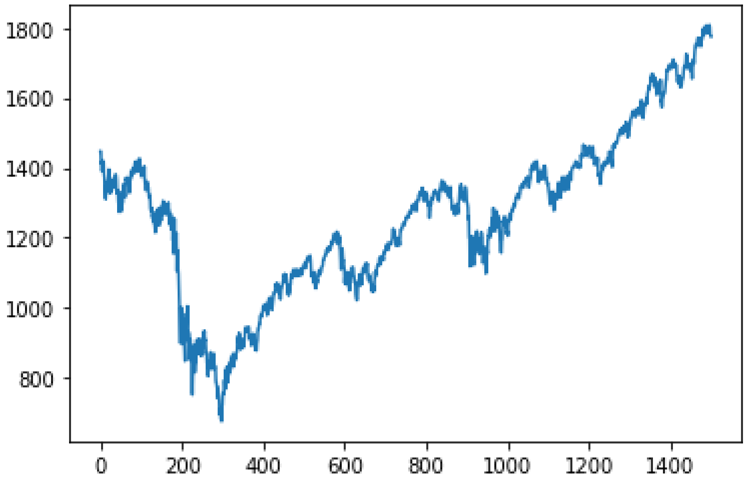
```
#print(sm.datasets.sunspots.NOTE)
#dta = sm.datasets.sunspots.load_pandas().data
#dta.index = pd.Index(sm.tsa.datetools.dates_from_range('1700', '2008'))
#del dta["YEAR"]
```

```
::

    Number of Observations - 309 (Annual 1700 - 2008)
    Number of Variables - 1
    Variable name definitions::

        SUNACTIVITY - Number of sunspots for each year

    The data file contains a 'YEAR' variable that is not returned by load.
```

```
dta = pd.read_csv('data.csv', index_col=0, header=0, error_bad_lines=False).tail(1500).res
```

```
dta['close'].plot()
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
dta.head()
```

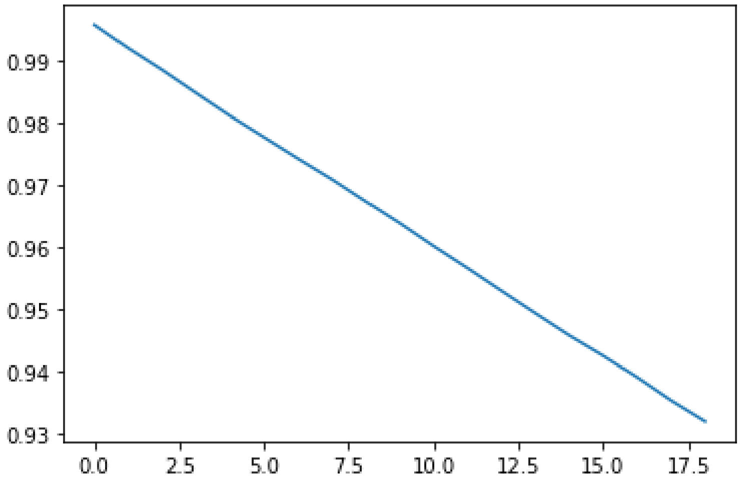
| | open | high | low | close | volume |
|---|---|---|---|---|---|
| **0** | 1467.969971 | 1471.770020 | 1442.069946 | 1447.160034 | 3452650000 |
| **1** | 1447.550049 | 1456.800049 | 1443.729980 | 1447.160034 | 3429500000 |
| **2** | 1444.010010 | 1444.010010 | 1411.189941 | 1411.630005 | 4166000000 |
| **3** | 1414.069946 | 1423.869995 | 1403.449951 | 1416.180054 | 4221260000 |
| **4** | 1415.709961 | 1430.280029 | 1388.300049 | 1390.189941 | 4705390000 |

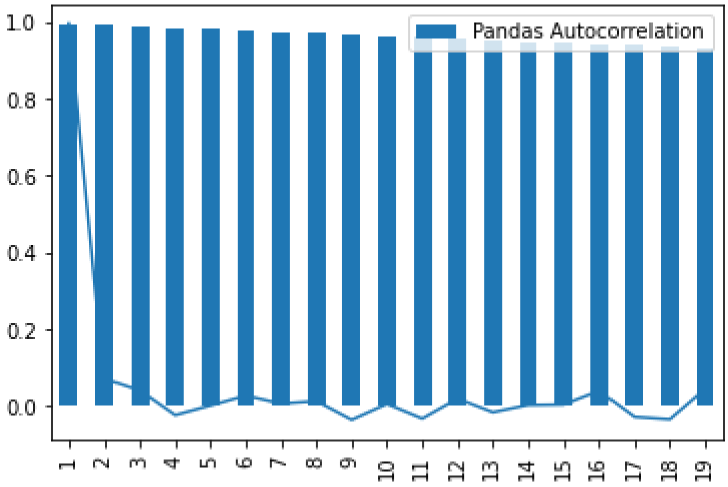


```
#split into test and train
percentage = 0.6
series = dta['close'].tolist()
size = int(len(series) * 0.66)
train, test = series[0:size], series[size:len(series)]
model = ARIMA(train , order = (9,0,0))
model_fit = model.fit()
```

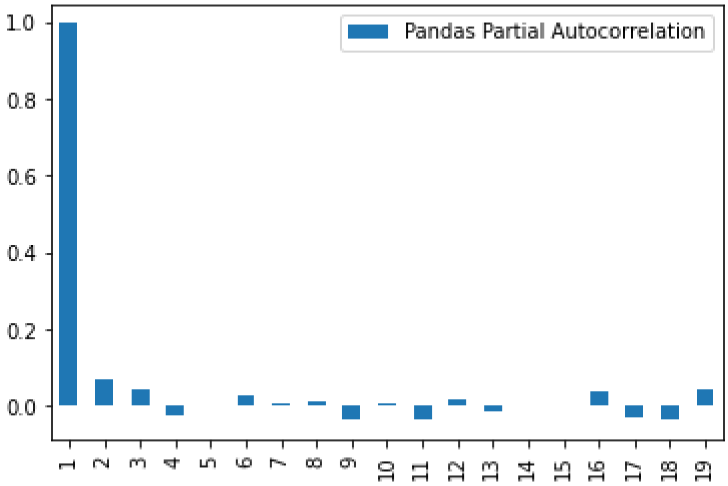
```
from statsmodels.tsa.stattools import acf, pacf
acf_1 = acf(series)[1:20]
plt.plot(acf_1)
test_df = pd.DataFrame([acf_1]).T
test_df.columns = ["Pandas Autocorrelation"]
test_df.index += 1
test_df.plot(kind='bar')
pacf_1 = pacf(series)[1:20]
plt.plot(pacf_1)
plt.show()
test_df = pd.DataFrame([pacf_1]).T
test_df.columns = ['Pandas Partial Autocorrelation']
test_df.index += 1
test_df.plot(kind='bar')
#from the figures we conclude that it is an AR process with a lag of 8-9
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541: FutureWarnir
  warnings.warn(msg, FutureWarning)
```





```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b508a7240>
```



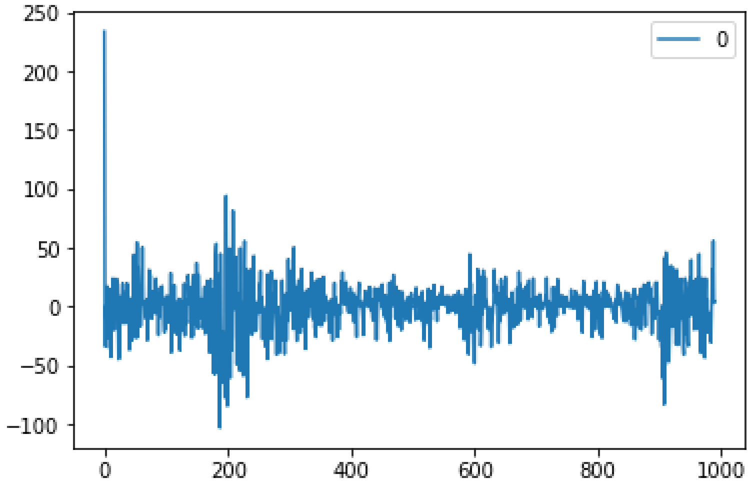
```
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout
from sklearn import preprocessing
from keras.wrappers.scikit_learn import KerasRegressor
```

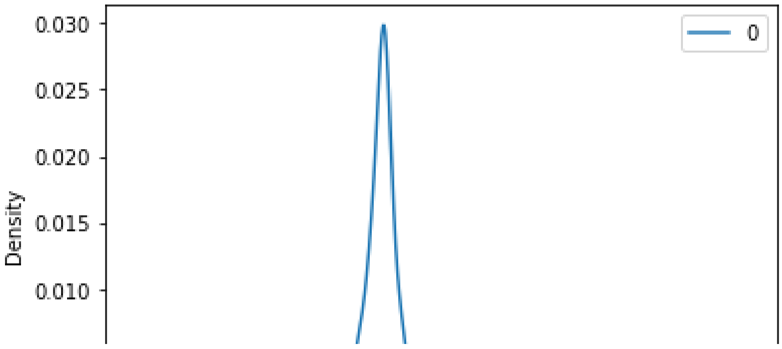
```
Using TensorFlow backend.
```

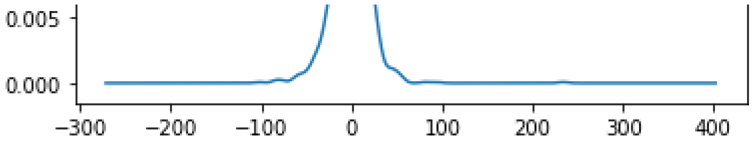
```
"""
Arima Rolling Forecast
```

```
"""
predicted1, resid_test = [], []
history = train
for t in range(len(test)):
    model = ARIMA(history, order=(9,0,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    resid_test.append(test[t] - output[0])
    predicted1.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
test_resid = []
for i in resid_test:
    test_resid.append(i[0])
error = mean_squared_error(test, predicted1)
print('Test MSE: %.3f' % error)
plt.plot(test)
plt.plot(predicted1, color='red')
plt.show()


"""
Residual Diagnostics
"""
train, test = series[0:size], series[size:len(series)]
model = ARIMA(train, order=(9,0,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
residuals.plot(kind='kde')
plt.show()
print(residuals.describe())
#plot the acf for the residuals
acf_1 = acf(model_fit.resid)[1:20]
plt.plot(acf_1)
test_df = pd.DataFrame([acf_1]).T
test_df.columns = ["Pandas Autocorrelation"]
test_df.index += 1
test_df.plot(kind='bar')
#from the acf obtained from the residuals we concule that
#there is still a nonlinear relationship among the residuals
```

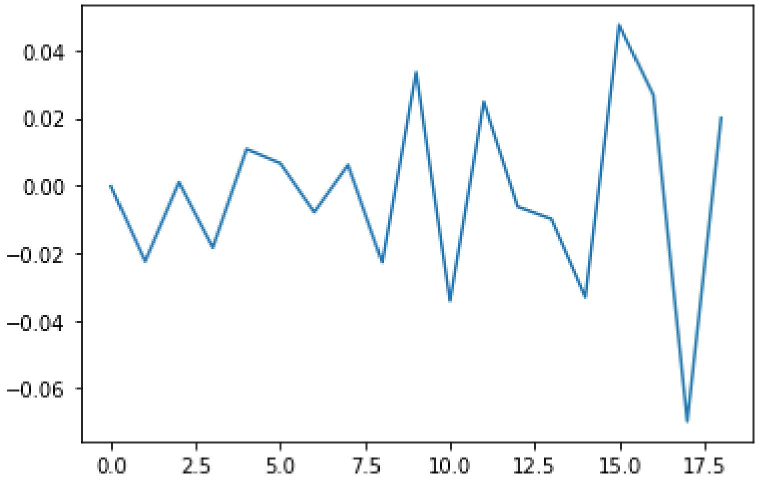
```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  990
Model:                     ARMA(9, 0)   Log Likelihood               -4304.444
Method:                       css-mle   S.D. of innovations             18.665
Date:                Wed, 15 Apr 2020   AIC                           8630.887
Time:                        06:35:22   BIC                           8684.762
Sample:                             0   HQIC                          8651.374

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       1213.6978    113.115     10.730      0.000     991.997    1435.399
ar.L1.y        0.8682      0.032     27.362      0.000       0.806       0.930
ar.L2.y        0.0570      0.042      1.356      0.175      -0.025       0.139
ar.L3.y        0.0745      0.042      1.767      0.078      -0.008       0.157
ar.L4.y       -0.0045      0.042     -0.107      0.915      -0.087       0.078
ar.L5.y       -0.0412      0.042     -0.974      0.330      -0.124       0.042
ar.L6.y        0.0477      0.042      1.127      0.260      -0.035       0.131
ar.L7.y       -0.0296      0.042     -0.700      0.484      -0.113       0.053
ar.L8.y        0.0646      0.042      1.527      0.127      -0.018       0.148
ar.L9.y       -0.0412      0.032     -1.287      0.198      -0.104       0.022
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.3704           -0.0000j            1.3704           -0.5000
AR.2           -0.9471           -1.0833j            1.4389           -0.3643
AR.3           -0.9471           +1.0833j            1.4389            0.3643
AR.4           -0.0276           -1.4772j            1.4774           -0.2530
AR.5           -0.0276           +1.4772j            1.4774            0.2530
AR.6            1.0036           -0.0000j            1.0036           -0.0000
AR.7            1.0978           -1.0517j            1.5203           -0.1216
AR.8            1.0978           +1.0517j            1.5203            0.1216
AR.9            1.6898           -0.0000j            1.6898           -0.0000
-----------------------------------------------------------------------------
```

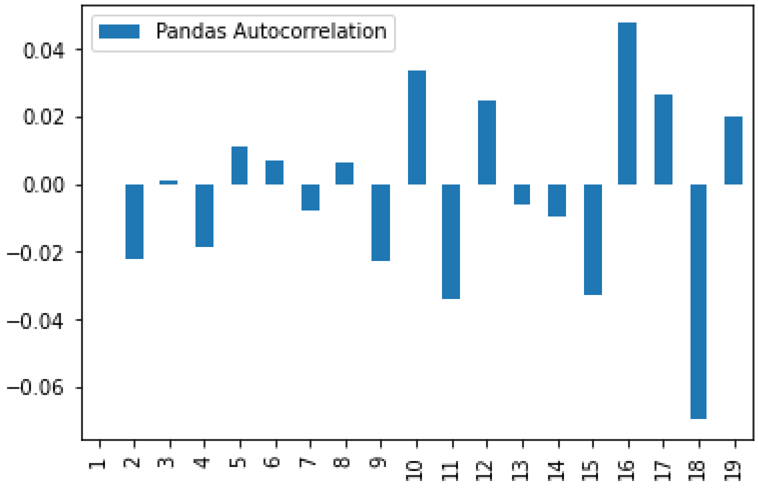






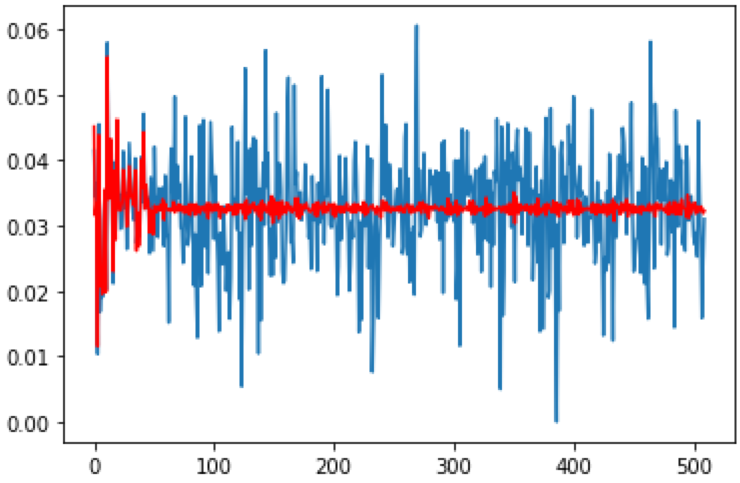
```
                0
count  990.000000
mean    -0.327643
std     20.079317
min   -103.450298
25%     -9.477502
50%      1.216403
75%      8.601497
max    233.462208
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541: FutureWarnir
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f8acd141c88>
```





```
"""
Hybrid Model
"""
window_size = 50
def make_model(window_size):
    model = Sequential()
    model.add(Dense(50, input_dim=window_size, init="uniform",
    activation="tanh"))
    model.add(Dense(25, init="uniform", activation="tanh"))
    model.add(Dense(1))
    model.add(Activation("linear"))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model



model = make_model(50)

min_max_scaler = preprocessing.MinMaxScaler()
train = np.array(train).reshape(-1,1)

train_scaled = min_max_scaler.fit_transform(test_data)

train_X,train_Y = [],[]
for i in range(0 , len(train_scaled) - window_size):
    train_X.append(train_scaled[i:i+window_size])
    train_Y.append(train_scaled[i+window_size])

new_train_X,new_train_Y = [],[]
for i in train_X:
    new_train_X.append(i.reshape(-1))
for i in train_Y:
    new_train_Y.append(i.reshape(-1))
new_train_X = np.array(new_train_X)
new_train_Y = np.array(new_train_Y)
#new_train_X = np.reshape(new_train_X, (new_train_X.shape[0], new_train_X.shape[1], 1))
model.fit(new_train_X,new_train_Y, nb_epoch=500, batch_size=512, validation_split = .05)


test_extended = train.tolist()[-1*window_size:] + test_resid
test_data = []
for i in test_extended:
    try:
        test_data.append(i[0])
    except:
        test_data.append(i)
test_data = np.array(test_data).reshape(-1,1)
min_max_scaler = preprocessing.MinMaxScaler()
test_scaled = min_max_scaler.fit_transform(test_data)
test_X,test_Y = [],[]
for i in range(0 , len(test_scaled) - window_size):
    test_X.append(test_scaled[i:i+window_size])
    test_Y.append(test_scaled[i+window_size])
    new_test_X,new_test_Y = [],[]
for i in test_X:
    new_test_X.append(i.reshape(-1))
for i in test_Y:
    new_test_Y.append(i.reshape(-1))
new_test_X = np.array(new_test_X)
new_test_Y = np.array(new_test_Y)
#new_test_X = np.reshape(new_test_X, (new_test_X.shape[0], new_test_X.shape[1], 1))
predictions = model.predict(new_train_X)
predictions_rescaled=min_max_scaler.inverse_transform(predictions)
Y = pd.DataFrame(new_train_Y)
pred = pd.DataFrame(predictions)
plt.plot(Y)
plt.plot(pred , color = 'r')
```

```
#p.plot()
plt.show()
error = mse(test_resid,predictions)
print('Test MSE: %.3f' % error)
```

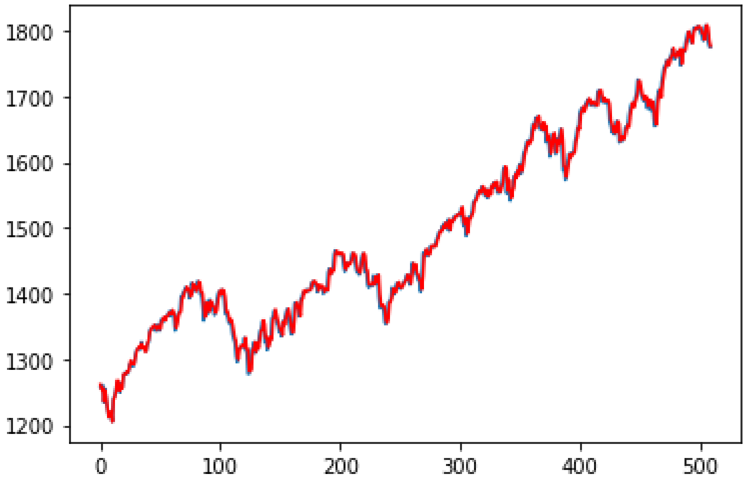


```
Test MSE: 132.290
```

```
pred_final = predictions_rescaled + predicted1
error = mse(test,pred_final)
print('Test MSE: %.3f' % error)
```

```
Test MSE: 118.799
```

```
Y = pd.DataFrame(test)
pred = pd.DataFrame(pred_final)
plt.plot(Y)
plt.plot(pred , color = 'r')
#p.plot()
plt.show()
```



```
from sklearn.metrics import mean_absolute_error as mae
from sklearn.utils import check_array
#from sklearn.metrics import mean_absolute_percentage_error as mape
from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
error = mse(test,pred_final)
print('Test MSE: %.3f' % error)
```

```
Test MSE: 118.799
```

```
error = mae(test,pred_final)
print('Test MAE: %.3f' % error)
```

```
Test MAE: 8.106
```

```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
rms = sqrt(mean_squared_error(test, pred_final))
print('Test RMS:%.3f'% rms)
```

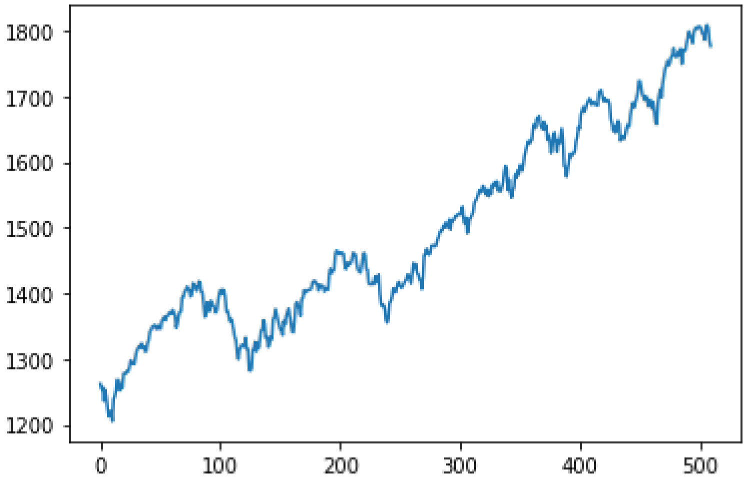
```
Test RMS:10.900
```

```
print('Test MAPE:%.3f'% mean_absolute_percentage_error(test, pred_final))
```

```
Test MAPE:11.638
```

```
plt.plot(pred_final)
```

```
[<matplotlib.lines.Line2D at 0x7f8acc09aeb8>]
```


1800
1700
1600
1500
1400
1300
1200
0
100
200
300
400
500