This is a Starter Notebook for Stock Price Prediction using Linear Regression

About the Dataset -

The dataset has around 60 features which includes features extracted from OHLC, other index prices such as QQQ(Nasdaq-100 ETF) & S&P 500, technical Indicators such as Bollinger bands, EMA(Exponential Moving Averages, Stocastic %K oscillator, RSI etc)

Furthermore, I have created lagged features from previous day price data as we know previous day prices affect the future stock price.

Then, the data has date features which specifies, if its a leap year, if its month start or end, Quarter start or end, etc.

All of these features have something to offer for forcasting. Some tells us about the trend, some gives us a signal if the stock is overbought or oversold, some portrays the strength of the price trend.

In this notebook, I will analyse the data and create a basic Linear regression model to forecast Stock Prices. In future notebooks, I will use other algorithms like Random Forest, XGBoost and LSTM for this task.

I will also create a Notebook explaining how I have extracted this data using only OHLC(Open High Low Close) data.

In [9]:
```
import pandas as pd
import matplotlib.pyplot as plt
from pylab import rcParams
import numpy as np
import seaborn as sns
import os

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, train_test_split, Grid
from sklearn.feature_selection import RFECV, SelectFromModel, SelectKBest
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
%matplotlib inline
```

Load the data

I will use the Apple Stock Data for this notebook

In [10]:
```
Stock = pd.read_csv('AAPL.csv',  index_col=0)

df_Stock = Stock
df_Stock = df_Stock.rename(columns={'Close(t)':'Close'})
df_Stock.head()
```

Out[10]:

Date	Open	High	Low	Close	Volume	SD20	Upper_Band	Lower_Band	S_Close(t-1)
2005-10-17	6.66	6.69	6.50	6.60	154208600	0.169237	6.827473	6.150527	6.67
2005-10-18	6.57	6.66	6.44	6.45	152397000	0.168339	6.819677	6.146323	6.60
2005-10-19	6.43	6.78	6.32	6.78	252170800	0.180306	6.861112	6.139888	6.45
2005-10-20	6.72	6.97	6.71	6.93	339440500	0.202674	6.931847	6.121153	6.78
2005-10-21	7.02	7.03	6.83	6.87	199181500	0.216680	6.974860	6.108140	6.93

5 rows × 63 columns

In [11]:
```
df_Stock.tail(5)
```

Out[11]:

Date	Open	High	Low	Close	Volume	SD20	Upper_Band	Lower_Band	S_Clo
2020-08-07	452.82	454.70	441.17	444.45	49453300	27.954399	455.316298	343.498702	45
2020-08-10	450.40	455.10	440.00	450.91	53100900	29.847338	462.586675	343.197325	44
2020-08-11	447.88	449.93	436.43	437.50	46975600	30.576290	466.543079	344.237921	45
2020-08-12	441.99	453.10	441.19	452.04	41486200	32.050532	472.583564	344.381436	43
2020-08-13	457.72	464.17	455.71	460.04	52520500	33.532634	479.279768	345.149232	45

5 rows × 63 columns

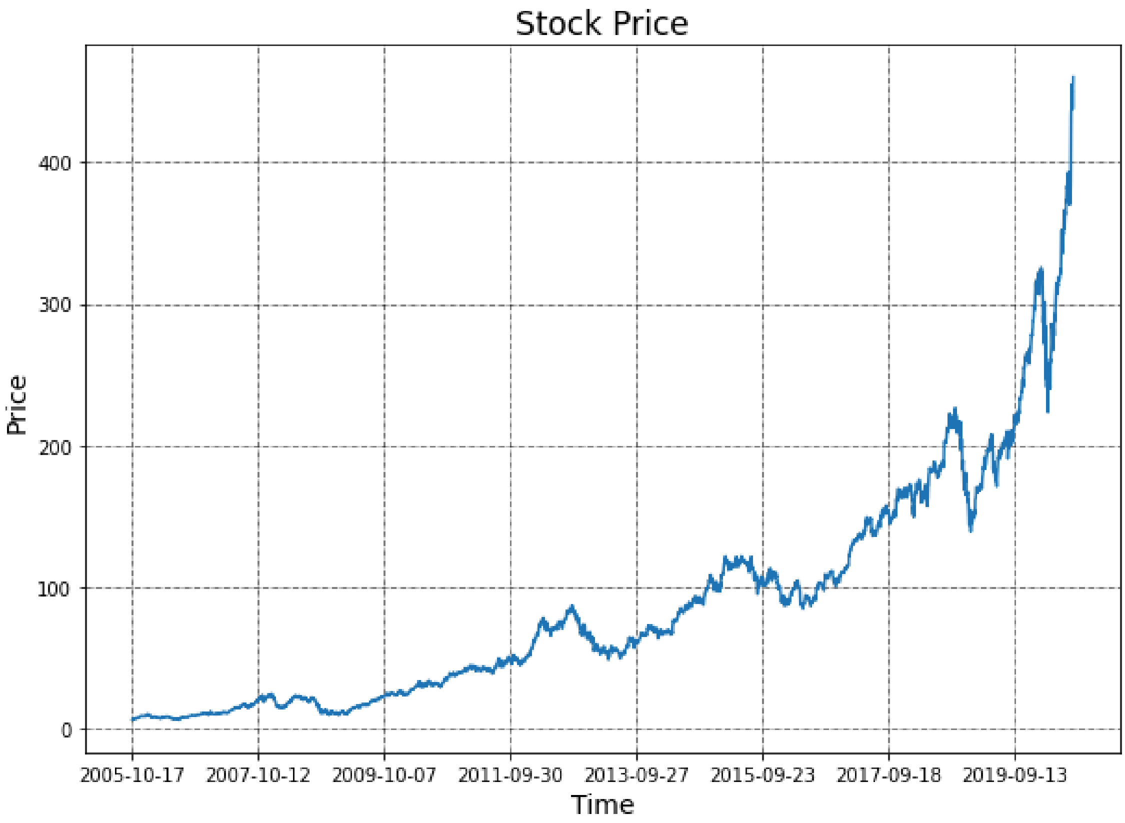
In [12]:
```
df_Stock.shape
```

Out[12]: (3732, 63)

In [ ]:
```
df_Stock.columns
```

Plot Time Series chart for AAPL

In [13]:
```
df_Stock['Close'].plot(figsize=(10, 7))
plt.title("Stock Price", fontsize=17)
plt.ylabel('Price', fontsize=14)
plt.xlabel('Time', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



Remove some of the columns which are not required

In [14]:
```
df_Stock = df_Stock.drop(columns='Date_col')
```

Test Train Set

Close_forecast is the column that we are trying to predict here which is the price for the next day.

In [16]:
```
def create_train_test_set(df_Stock):

    features = df_Stock.drop(columns=['Close_forcast'], axis=1)
    target = df_Stock['Close_forcast']


    data_len = df_Stock.shape[0]
    print('Historical Stock Data length is - ', str(data_len))

    #create a chronological split for train and testing
    train_split = int(data_len * 0.88)
    print('Training Set length - ', str(train_split))

    val_split = train_split + int(data_len * 0.1)
    print('Validation Set length - ', str(int(data_len * 0.1)))

    print('Test Set length - ', str(int(data_len * 0.02)))

    # Splitting features and target into train, validation and test samples
    X_train, X_val, X_test = features[:train_split], features[train_split:v
    Y_train, Y_val, Y_test = target[:train_split], target[train_split:val_s

    #print shape of samples
    print(X_train.shape, X_val.shape, X_test.shape)
    print(Y_train.shape, Y_val.shape, Y_test.shape)

    return X_train, X_val, X_test, Y_train, Y_val, Y_test
```

In [17]:
```
X_train, X_val, X_test, Y_train, Y_val, Y_test = create_train_test_set(df_S
```

```
Historical Stock Data length is -  3732
Training Set length -  3284
Validation Set length -  373
Test Set length -  74
(3284, 61) (373, 61) (75, 61)
(3284,) (373,) (75,)
```

Prediction using Linear Regression

In [18]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, Y_train)
```

Out[18]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:
```
print('LR Coefficients: \n', lr.coef_)
print('LR Intercept: \n', lr.intercept_)
```

```
LR Coefficients:
 [ 8.63716903e-03  1.86051913e-01  1.55487066e-01  1.12263755e+00
  1.27287035e-10  6.75244705e-03  1.40229156e-01  1.13219368e-01
  4.25628102e-02  8.96348462e-02  1.01914954e-01  5.94183542e-02
  7.95194238e-02  7.10399831e-02  2.71425001e-01  1.26724262e-01
  8.79333126e-02 -5.87980441e-03 -3.31643386e-01 -3.31643386e-01
 -3.31643386e-01 -3.31643386e-01 -3.31643386e-01  1.88650012e+00
 -1.27270725e+00 -1.65042227e-01 -4.36658178e-04 -3.18220103e-12
 -5.07434422e-03  9.02936480e-03  5.78317097e-04  5.78317091e-04
 -5.57918110e-01 -2.02303507e-10  4.18932111e-11  1.69322438e-02
  1.61636704e-02 -1.75659582e-02  6.12165520e-03  2.15420350e-01
  1.13979655e-01 -2.41954674e-01  7.63050311e-02  3.73276597e-01
 -6.66133815e-16 -5.60843988e-02  4.08788805e-02  5.13473863e-01
 -2.94431538e-02 -8.41335081e-02  5.10939134e-02 -8.14435724e-03
 -1.95035195e-02  5.67587250e-02  4.39707788e-02  1.29311738e-02
 -9.99967543e-03 -3.89778364e-03 -1.62174814e-03  1.44436900e-03
  2.83455425e-04]
LR Intercept:
 -83.36486405151932
```

Evaluation

In [20]:
```
print("Performance (R^2): ", lr.score(X_train, Y_train))
```

```
Performance (R^2):  0.9994516474373267
```

In [22]:
```
def get_mape(y_true, y_pred):
    """
    Compute mean absolute percentage error (MAPE)
    """
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Predict for the test dataset

In [23]:
```
Y_train_pred = lr.predict(X_train)
Y_val_pred = lr.predict(X_val)
Y_test_pred = lr.predict(X_test)
```

In [24]:
```
print("Training R-squared: ",round(metrics.r2_score(Y_train,Y_train_pred),2
print("Training Explained Variation: ",round(metrics.explained_variance_sco
print('Training MAPE:', round(get_mape(Y_train,Y_train_pred), 2))
print('Training Mean Squared Error:', round(metrics.mean_squared_error(Y_tr
print("Training RMSE: ",round(np.sqrt(metrics.mean_squared_error(Y_train,Y_
print("Training MAE: ",round(metrics.mean_absolute_error(Y_train,Y_train_pr

print(' ')

print("Validation R-squared: ",round(metrics.r2_score(Y_val,Y_val_pred),2))
print("Validation Explained Variation: ",round(metrics.explained_variance_s
print('Validation MAPE:', round(get_mape(Y_val,Y_val_pred), 2))
print('Validation Mean Squared Error:', round(metrics.mean_squared_error(Y_
print("Validation RMSE: ",round(np.sqrt(metrics.mean_squared_error(Y_val,Y_
print("Validation MAE: ",round(metrics.mean_absolute_error(Y_val,Y_val_pred

print(' ')

print("Test R-squared: ",round(metrics.r2_score(Y_test,Y_test_pred),2))
print("Test Explained Variation: ",round(metrics.explained_variance_score(Y
print('Test MAPE:', round(get_mape(Y_test,Y_test_pred), 2))
print('Test Mean Squared Error:', round(metrics.mean_squared_error(Y_test,Y
print("Test RMSE: ",round(np.sqrt(metrics.mean_squared_error(Y_test,Y_test_
print("Test MAE: ",round(metrics.mean_absolute_error(Y_test,Y_test_pred),2)
```

```
Training R-squared:  1.0
Training Explained Variation:  1.0
Training MAPE: 1.45
Training Mean Squared Error: 1.48
Training RMSE:  1.22
Training MAE:  0.76

Validation R-squared:  0.99
Validation Explained Variation:  0.99
Validation MAPE: 1.68
Validation Mean Squared Error: 1.48
Validation RMSE:  5.91
Validation MAE:  3.75

Test R-squared:  0.96
Test Explained Variation:  0.97
Test MAPE: 1.77
Test Mean Squared Error: 79.21
Test RMSE:  8.9
Test MAE:  6.5
```

We have a decent Mean Absolute error but not great. I will create further tuned models in later notebooks. This is just to get you started with the dataset.

In [25]:
```
df_pred = pd.DataFrame(Y_val.values, columns=['Actual'], index=Y_val.index)
df_pred['Predicted'] = Y_val_pred
df_pred = df_pred.reset_index()
df_pred.loc[:, 'Date'] = pd.to_datetime(df_pred['Date'],format='%Y-%m-%d')
df_pred
```

Out[25]:

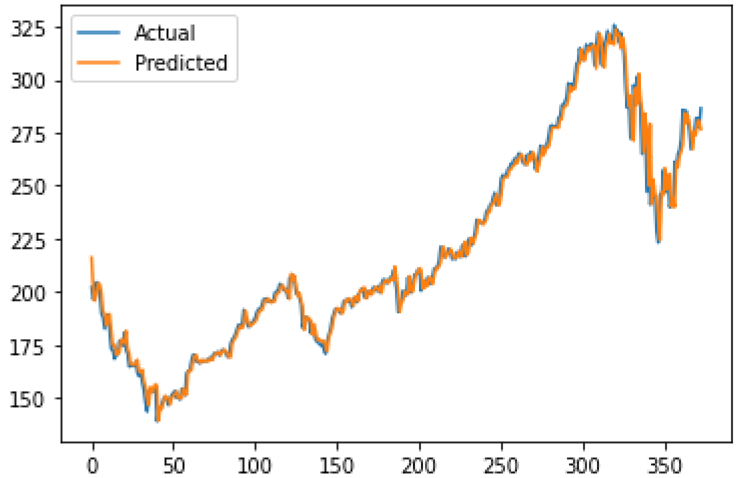
	Date	Actual	Predicted
0	2018-11-01 00:00:00	202.30	216.289778
1	2018-11-02 00:00:00	196.56	201.470181
2	2018-11-05 00:00:00	198.68	195.948932
3	2018-11-06 00:00:00	204.71	199.043601
4	2018-11-07 00:00:00	204.00	204.193666
...	...	...	...
368	2020-04-22 00:00:00	273.79	275.751756
369	2020-04-23 00:00:00	281.70	274.020243
370	2020-04-24 00:00:00	281.90	280.710045
371	2020-04-27 00:00:00	277.33	280.543603
372	2020-04-28 00:00:00	286.44	276.562125

373 rows × 3 columns

Plot Predicted vs Actual Prices on Time Series plot

In [26]:
```
df_pred[['Actual', 'Predicted']].plot()
```

Out[26]: <Axes: >



Overall the Predictions looks good for the test data!

Future Notebooks

I will create a Notebook explaining how I have extracted this data using only OHLC(Open High Low Close) data and a custom pipeline