In [1]:
```python
 1  # This program predicts stock price using four machine learning regression r
 2  # 1) Creates a .csv file with historical data of the stock
 3  # 2) Calculates and plots stock's Rolling Mean/Moving Average, stock Return
 4  # 3) Performs/plots Linear Regression, predicts stock price, calculates rmse
 5  # 4) Performs/plots Ridge Regression, predicts stock price, calculates rmse,
 6  # 5) Performs/plots Lasso Regression, predicts stock price, calculates rmse,
 7  # 6) Performs/plots KNN Regression, predicts stock price, calculates rmse, v
 8
 9  # Import yfinance with alias of yf
10  import yfinance as yf
11  # Import numpy as np for large, multi-dimensional arrays, matrices and mathe
12  import numpy as np
13  # Import pandas as pd for data manipulation, analysis, data structures and c
14  import pandas as pd
15  # import external pandas_datareader library with alias of pdr
16  import pandas_datareader as pdr
17  # import matplot library with alias of mplt. Pyplot module provides simple f
18  import matplotlib.pyplot as mplt
19  from matplotlib import style
20  %matplotlib inline
21  # import datetime internal datetime python module which supplies classes for
22  import datetime
23  # import math module for mathematical operations can be performed with ease
24  import math
25  from math import sqrt
26  import warnings
27  warnings.filterwarnings('ignore')
28  # sklearn.metrics has a mean_squared_error function. The RMSE is just the sq
29  from sklearn.metrics import mean_squared_error
30  # R^2 (coefficient of determination) regression score function. Best possibl
31  from sklearn.metrics import r2_score
32  # split the dataset into the training and the testing datasets; train_test_s
33  from sklearn.model_selection import train_test_split
34  # LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=
35  from sklearn.linear_model import LinearRegression
36  #  supervised learning through Ridge regression. In ridge regression, the co
37  from sklearn.linear_model import Ridge
38  #  supervised learning through Lasso regression. Lasso performs both variabl
39  from sklearn.linear_model import Lasso
40  # KNN alogrithm uses 'feature similarity' to predict values of any new data
41  from sklearn.neighbors import KNeighborsRegressor
42
43  ### define the stock, start date and end date for downloading stock data fr
44  start = datetime.datetime(2017,1,1)
45  end = datetime.datetime(2019,9,12)
46
47  # Choose the stock ticker symbol
48  myStock = 'AAPL'
49
50  ### Extract stock historical data from Yahoo into dataframe
51  df_Stock = pdr.DataReader(myStock, 'yahoo', start, end)
52  # Create .csv file with stock historical data
53  df_Stock.to_csv(myStock + '.csv')
54  print ('1- Downloaded ' + myStock + ' stock historical data from ' + start.s
55  # Display the last 5 records of the stock dataframe
56  print ('\033[1m \033[4mStock Prices from Yahoo Finance \033[0m' + "\n")
```

```python
57   with pd.option_context('expand_frame_repr', False):
58       print(df_Stock.tail())
59   print("\n")
60
61   ### Plot stock's adjusted close price and moving average
62   print ('2- Note: ' + myStock + ' Moving Average steadily rises over the wind
63   # Plot stock's adjusted close price
64   df_Stock["Adj Close"].plot(title=df_Stock.columns[5], label = myStock, color
65   # Calculate the Rolling Mean (Moving Average) of the stock to determine the
66   df_Stock_close_pr = df_Stock['Close']
67   df_Stock_MovingAv = df_Stock_close_pr.rolling(window=50).mean()
68   # Display the last 5 stock moving average records
69   with pd.option_context('expand_frame_repr', False):
70       print('\033[1m \033[4mStock Rolling Mean/Moving Average(50 Day)\033[0m'
71       print(df_Stock_MovingAv.tail().to_string(header=None))
72   # Plot Stock's Rolling Mean Moving Averge
73   df_Stock_MovingAv.plot(title=df_Stock.columns[5], label = myStock + ' Moving
74   mplt.show()
75
76   ### Calculate stock's return rate
77   df_Stock_ret_rate = (df_Stock['Adj Close'] / df_Stock['Adj Close'].shift(1))
78   # Display stock's last 5 return rate records
79   df_Stock_ret_rate.tail().T
80   # Plot stock return rate
81   print('\033[1m \033[4m3- Stock Return Rate \033[0m')
82   df_Stock_ret_rate.plot(title=df_Stock.columns[-1], label=myStock+' Return Ra
83   mplt.show()
84
85   ### Feature Engineering
86   df_Stock_Pr_Regr = df_Stock.loc[:,['Adj Close','Volume']]
87   df_Stock_Pr_Regr['HL_PCT'] = ((df_Stock['High']-df_Stock['Low'])/df_Stock['C
88   df_Stock_Pr_Regr['PCT_change'] = ((df_Stock['Close']-df_Stock['Open'])/df_St
89   # Display the last 5 records of the dataframe
90   print('\033[1m \033[4m4-Display Stock Adj Close, Volume, Price(High-Low) % C
91   with pd.option_context('expand_frame_repr', False):
92       print(df_Stock_Pr_Regr.tail())
93   print("\n")
94
95   print('\033[1m \033[4m5- Regression Data Details\033[0m' '\n')
96   # Split 80% data to train and 20% data to test the model
97   Total_Data_Ind = df_Stock.shape[0]
98   print('Total data size is ' + str(Total_Data_Ind))
99   train_size = 0.8
100  #Keep 80% data for training
101  Train_Data_Ind = int(train_size * Total_Data_Ind)
102  print('Training data size is ' + str(Train_Data_Ind))
103  #Keep 20% data for testing
104  Test_Data_Ind = Total_Data_Ind - Train_Data_Ind
105  print('Test data size is ' + str(Test_Data_Ind))
106  Train_test_split_time = df_Stock.index[Train_Data_Ind]
107  print('Train Test Split Time is = ' + str(Train_test_split_time) + '\n')
108
109  # Create X and y train data sets to train the model
110  X_daterange = df_Stock.iloc[:,:-1].to_numpy()
111  y_closepx = df_Stock.iloc[:,-1].to_numpy()
112
113  # Split into 80% train data and 20 % test data
```

```python
114  X_daterange_train, X_daterange_test, y_closepx_train, y_closepx_test = trair

115
116  ##################### Linear Regression ####################################
117
118  print ('\x1b[43m \033[1m \033[4m6- Starting Linear Regression \033[0m \x1b[6
119
120  # Create Linear Regression Estimator/Model Object for training, testing/pre
121  LinReg_Model = LinearRegression(n_jobs= -1)
122
123  # Train the Liner Regression model using training set
124  LinReg_Model.fit(X_daterange_train, y_closepx_train)
125
126  # Test to predict with this model using training data
127  Training_Predications = LinReg_Model.predict(X_daterange_train).reshape(-1,1
128
129  # Test to predict with this model using test data
130  Test_Predictions = LinReg_Model.predict(X_daterange_test).reshape(-1,1)
131
132  # Combine the training and testing predictions; Joining the two arrays alonc
133  All_Predictions = np.concatenate((Training_Predications, Test_Predictions),
134
135  # transform the prediction data to dataframe and plot the prediction results
136  df_lin_reg = pd.DataFrame(All_Predictions, columns=[myStock+' Linear Regr '+
137  df_lin_reg[df_Stock.columns[-1]] = y_closepx
138  # plot the results of linear regression
139  df_lin_reg.plot(title=df_Stock.columns[-1], label=myStock+'_Linear Regressic
140  # Add the train and test split timing line
141  mplt.axvline(pd.Timestamp(Train_test_split_time),color='r', linestyle='--')
142  print('\033[1m \033[4m6A- Plot Linear Regression \033[0m' '\n')
143  mplt.show()
144
145  # calculate the coefficients
146  print('\033[1m \033[4m6B- Linear Regression Coefficients \033[0m' '\n')
147  print('  Coeficiencts: ', list(LinReg_Model.coef_))
148  print('\n')
149
150  # calculate root mean squared error
151  Training_rmse = sqrt(mean_squared_error(y_closepx_train.reshape(-1,1), Trair
152  Test_rmse = sqrt(mean_squared_error(y_closepx_test.reshape(-1,1), Test_Predi
153  print('\033[1m \033[4m6C- Linear Regression Root Mean Square Error \033[0m'
154  print('      Linear Regression Training RMSE = ' + str(Training_rmse))
155  print('      Linear Regression Testing RMSE = ' + str(Test_rmse))
156  print('\n')
157
158  # calculate the variance which is squared deviation of a variable from its r
159  #A low value for variance indicates that the data are clustered together anc
160  # whereas a high value would indicate that the data in the given set are muc
161  print('\033[1m \033[4m6D-  Linear Regression Variance \033[0m' '\n')
162  print('  Variance: ', r2_score(y_closepx_test, Test_Predictions))
163  print('\n')
164
165  ### Use the score method to evaluate the trained model
166  # The score method finds the mean accuracy of self.predict(X) with y of the
167  Confidence_Score = LinReg_Model.score(X_daterange_test, y_closepx_test)
168  print('\033[1m \033[4m6E-  Linear Regression Confidence \033[0m' '\n')
169  print(' The linear regression confidence is ' + str(Confidence_Score) + '\n
170
```

```python
171  #################### Ridge Regression ################################
172  # Ridge Regression is a technique for analyzing multiple regression data tha
173  # By adding a degree of bias to the regression estimates, ridge regression r
174  # Create Ridge Estimator/Model Object for training, testing/prediction
175
176  print ('\x1b[43m \033[1m \033[4m7- Starting Ridge Regression \033[0m \x1b[0m
177
178  RidgeReg_Model = Ridge(alpha=.5,normalize=True)
179  RidgeReg_Model.fit(X_daterange_train, y_closepx_train)
180
181  # Test to predict with this model using training data
182  Ridge_Training_Predications = RidgeReg_Model.predict(X_daterange_train).resh
183
184  # Test to predict with this model using test data
185  Ridge_Test_Predictions = RidgeReg_Model.predict(X_daterange_test).reshape(-1
186
187  # Combine the training and testing predictions; Joining the two arrays along
188  Ridge_All_Predictions = np.concatenate((Ridge_Training_Predications, Ridge_T
189
190  # transform the prediction data to dataframe and plot the prediction results
191  df_Ridge_reg = pd.DataFrame(Ridge_All_Predictions, columns=[myStock+' Ridge
192  df_Ridge_reg[df_Stock.columns[-1]] = y_closepx
193  # plot the results of Ridge regression
194  df_Ridge_reg.plot(label=myStock+'_Ridge Regression', figsize=(14,7), title=
195  Train_test_split_time = df_Stock.index[Train_Data_Ind]
196  # Add the train and test split timing line
197  mplt.axvline(pd.Timestamp(Train_test_split_time),color='r', linestyle='--')
198  print('\033[1m \033[4m7A- Plot Ridge Regression \033[0m' '\n')
199  mplt.show()
200
201  # calculate the coefficients
202  print('\033[1m \033[4m7B- Ridge Regression Coefficients \033[0m' '\n')
203  print('  Coeficiencts: ', list(RidgeReg_Model.coef_))
204  print('\n')
205
206  # calculate root mean squared error
207  Ridge_Training_rmse = sqrt(mean_squared_error(y_closepx_train.reshape(-1,1),
208  Ridge_Test_rmse = sqrt(mean_squared_error(y_closepx_test.reshape(-1,1), Ridg
209  print('\033[1m \033[4m7C- Ridge Root Mean Square Error \033[0m' '\n')
210  print('     Ridge Regression Training RMSE = ' + str(Ridge_Training_rmse))
211  print('     Ridge Regression Testing RMSE = ' + str(Ridge_Test_rmse))
212  print('\n')
213
214  # calculate the variance which is squared deviation of a variable from its r
215  #A low value for variance indicates that the data are clustered together and
216  # whereas a high value would indicate that the data in the given set are muc
217  print('\033[1m \033[4m7D-  Ridge Regression Variance \033[0m' '\n')
218  print('  Ridge Variance: ', r2_score(y_closepx_test, Ridge_Test_Predictions)
219  print('\n')
220
221  ### Use the score method to evaluate the trained model
222  # The score method finds the mean accuracy of self.predict(X) with y of the
223  Ridge_Confidence_Score = RidgeReg_Model.score(X_daterange_test, y_closepx_te
224  print('\033[1m \033[4m7E-  Ridge Regression Confidence \033[0m'  '\n')
225  print(' The Ridge regression confidence is ' + str(Ridge_Confidence_Score) +
226
227  #################### Lasso Regression ################################
```

```python
228  # Lasso uses shrinkage. Shrinkage is where data values are shrunk towards a
229  # When alpha is 0 , Lasso produces same coefficients as linear regression. W
230  # Create Ridge Estimator/Model Object for training, testing/prediction
231
232  print ('\x1b[43m \033[1m \033[4m8- Starting Lasso Regression \033[0m \x1b[0m
233
234  LassoReg_Model = Lasso(alpha=.1,normalize=True)
235  LassoReg_Model.fit(X_daterange_train, y_closepx_train)
236
237  # Test to predict with this Lasso model using training data
238  Lasso_Training_Predications = LassoReg_Model.predict(X_daterange_train).resh
239
240  # Test to predict with this Lasso model using test data
241  Lasso_Test_Predictions = LassoReg_Model.predict(X_daterange_test).reshape(-1
242
243  # Combine the training and testing predictions; Joining the two arrays along
244  Lasso_All_Predictions = np.concatenate((Lasso_Training_Predications, Lasso_T
245
246  # transform the prediction data to dataframe and plot the prediction results
247  df_Lasso_reg = pd.DataFrame(Lasso_All_Predictions, columns=[myStock+' Lasso
248  df_Lasso_reg[df_Stock.columns[-1]] = y_closepx
249  # plot the results of Ridge regression
250  df_Lasso_reg.plot(label=myStock+'_Lasso Regression', figsize=(14,7), title=d
251  Train_test_split_time = df_Stock.index[Train_Data_Ind]
252  # Add the train and test split timing line
253  mplt.axvline(pd.Timestamp(Train_test_split_time),color='r', linestyle='--')
254  print('\033[1m \033[4m8A- Plot Lasso Regression \033[0m' '\n')
255  mplt.show()
256
257  # calculate the coefficients
258  print('\033[1m \033[4m8B- Lasso Regression Coefficients \033[0m' '\n')
259  print('  Coeficiencts: ', list(LassoReg_Model.coef_))
260  print('\n')
261
262  # calculate root mean squared error
263  Lasso_Training_rmse = sqrt(mean_squared_error(y_closepx_train.reshape(-1,1),
264  Lasso_Test_rmse = sqrt(mean_squared_error(y_closepx_test.reshape(-1,1), Lass
265  print('\033[1m \033[4m8C- Lasso Root Mean Square Error \033[0m' '\n')
266  print('     Lasso Regression Training RMSE = ' + str(Lasso_Training_rmse))
267  print('     Lasso Regression Testing RMSE = ' + str(Lasso_Test_rmse))
268  print('\n')
269
270  # calculate the variance which is squared deviation of a variable from its m
271  #A low value for variance indicates that the data are clustered together and
272  # whereas a high value would indicate that the data in the given set are muc
273  print('\033[1m \033[4m8D-  Lasso Regression Variance \033[0m' '\n')
274  print('  Lasso Variance: ', r2_score(y_closepx_test, Lasso_Test_Predictions)
275  print('\n')
276
277  ### Use the score method to evaluate the trained model
278  # The score method finds the mean accuracy of self.predict(X) with y of the
279  Lasso_Confidence_Score = LassoReg_Model.score(X_daterange_test, y_closepx_te
280  print('\033[1m \033[4m8E-  Lasso Regression Confidence \033[0m'  '\n')
281  print(' The Lasso regression confidence is ' + str(Lasso_Confidence_Score) +
282
283  #################### KNN - K Nearest Neighbor Regression ##################
284  # K nearest neighbors is a simple algorithm that stores all available cases
```

```python
285   # Create KNN Estimator/Model Object for training, testing/prediction
286
287   print ('\x1b[43m \033[1m \033[4m9- Starting KNN Regression \033[0m \x1b[0m'
288
289   KNNReg_Model = KNeighborsRegressor(n_neighbors=5)
290   KNNReg_Model.fit(X_daterange_train, y_closepx_train)
291
292   # Test to predict with this KNN model using training data
293   KNN_Training_Predications = KNNReg_Model.predict(X_daterange_train).reshape(
294
295   # Test to predict with this KNN model using test data
296   KNN_Test_Predictions = KNNReg_Model.predict(X_daterange_test).reshape(-1,1)
297
298   # Combine the training and testing predictions; Joining the two arrays along
299   KNN_All_Predictions = np.concatenate((KNN_Training_Predications, Lasso_Test_
300
301   # transform the prediction data to dataframe and plot the prediction results
302   df_KNN_reg = pd.DataFrame(KNN_All_Predictions, columns=[myStock+' KNN Regr '
303   df_KNN_reg[df_Stock.columns[-1]] = y_closepx
304   # plot the results of Ridge regression
305   df_KNN_reg.plot(label=myStock+'_KNN Regression', figsize=(14,7), title=df_St
306   Train_test_split_time = df_Stock.index[Train_Data_Ind]
307   # Add the train and test split timing line
308   mplt.axvline(pd.Timestamp(Train_test_split_time),color='r', linestyle='--')
309   print('\033[1m \033[4m9A- Plot KNN Regression \033[0m' '\n')
310   mplt.show()
311
312   # calculate the coefficients
313   print('\033[1m \033[4m9B- KNN Regression Coefficients \033[0m' '\n')
314   #print('  Coeficiencts: ', KNNReg_Model.coef_)
315   print('   KNN classifier does not expose .coef_ or "feature impoertances_" a
316   print('\n')
317
318   # calculate root mean squared error
319   KNN_Training_rmse = sqrt(mean_squared_error(y_closepx_train.reshape(-1,1), K
320   KNN_Test_rmse = sqrt(mean_squared_error(y_closepx_test.reshape(-1,1), KNN_Te
321   print('\033[1m \033[4m9C- KNN Root Mean Square Error \033[0m' '\n')
322   print('     KNN Regression Training RMSE = ' + str(KNN_Training_rmse))
323   print('     KNN Regression Testing RMSE = ' + str(KNN_Test_rmse))
324   print('\n')
325
326   # calculate the variance which is squared deviation of a variable from its m
327   #A low value for variance indicates that the data are clustered together and
328   # whereas a high value would indicate that the data in the given set are muc
329   print('\033[1m \033[4m9D-  KNN Regression Variance \033[0m' '\n')
330   print('  KNN Variance: ', r2_score(y_closepx_test, KNN_Test_Predictions))
331   print('\n')
332
333   ### Use the score method to evaluate the trained model
334   # The score method finds the mean accuracy of self.predict(X) with y of the
335   KNN_Confidence_Score = KNNReg_Model.score(X_daterange_test, y_closepx_test)
336   print('\033[1m \033[4m9E-  KNN Regression Confidence \033[0m'  '\n')
337   print(' The KNN regression confidence is ' + str(KNN_Confidence_Score) + '\n
```
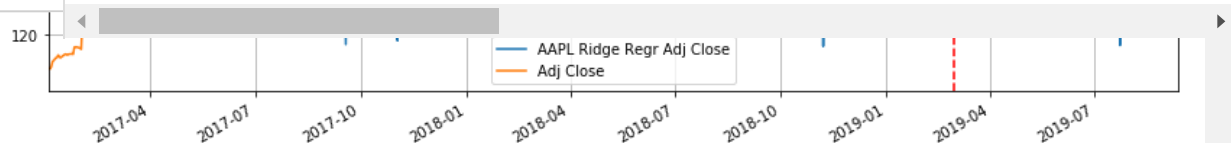
Date

7B- Ridge Regression Coefficients

  Coeficiencts: [0.22711130079399186, 0.23305398592477777, 0.227265190863661
18, 0.23427660564532615, 1.037471050896199e-09]

7C- Ridge Root Mean Square Error

      Ridge Regression Training RMSE = 3.3533378052243212
      Ridge Regression Testing RMSE = 3.6930781224734592

7D- Ridge Regression Variance