In [1]:

```
# https://www.youtube.com/watch?v=hOLSGMEEwlI
# Install the dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('bmh')
import seaborn as sns

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
```

In [2]:

```
# Load in data and store in a dataframe
df = pd.read_csv("TSLA_desc.csv")
```

Step 1: Initial EDA & Data Preparation

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       253 non-null    object 
 1   Open       253 non-null    float64
 2   High       253 non-null    float64
 3   Low        253 non-null    float64
 4   Close      253 non-null    float64
 5   Adj Close  253 non-null    float64
 6   Volume     253 non-null    int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 14.0+ KB
```

In [4]:

```
df.head()
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2/24/2020	167.80	172.70	164.44	166.76	166.76	75960800
1	2/25/2020	169.80	171.32	157.40	159.98	159.98	86452400
2	2/26/2020	156.50	162.66	155.22	155.76	155.76	70769200
3	2/27/2020	146.00	147.95	133.80	135.80	135.80	121385800
4	2/28/2020	125.94	138.10	122.30	133.60	133.60	122820850

In [5]:

```
df['Date']= pd.to_datetime(df['Date'])
```

In [6]:

```
# Get the number of trading days
df.shape
```

Out[6]:

```
(253, 7)
```

In [7]:

```
df.describe()
```

Out[7]:

	Open	High	Low	Close	Adj Close	Volume
count	253.000000	253.000000	253.000000	253.000000	253.000000	2.530000e+02
mean	385.082530	394.864585	374.585099	385.804941	385.804941	6.422319e+07
std	233.896736	238.236753	229.178670	234.690171	234.690171	3.352482e+07
min	74.940000	80.970000	70.100000	72.240000	72.240000	1.515765e+07
25%	165.560000	167.000000	161.600000	163.880000	163.880000	3.810520e+07
50%	372.140000	382.500000	360.500000	372.720000	372.720000	5.906245e+07
75%	497.990000	502.500000	487.570000	498.320000	498.320000	8.381685e+07
max	891.380000	900.400000	871.600000	883.090000	883.090000	2.221262e+08

In [8]:

```
df.head()
```

Out[8]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-02-24	167.80	172.70	164.44	166.76	166.76	75960800
1	2020-02-25	169.80	171.32	157.40	159.98	159.98	86452400
2	2020-02-26	156.50	162.66	155.22	155.76	155.76	70769200
3	2020-02-27	146.00	147.95	133.80	135.80	135.80	121385800
4	2020-02-28	125.94	138.10	122.30	133.60	133.60	122820850

In [9]:

```
df.shape
```

Out[9]:

```
(253, 7)
```

Step 2: EDA & Data Preprocessing

In [10]:

```
# this will show only features that have nonzero missing values
# df_na[df_na!=0]
df_na = df.isna()
df_na.sum()
```

Out[10]:

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [11]:

```
# limit to numerical data using df.select_dtypes()
df_num = df.select_dtypes(include=['number'])
```

In [12]:

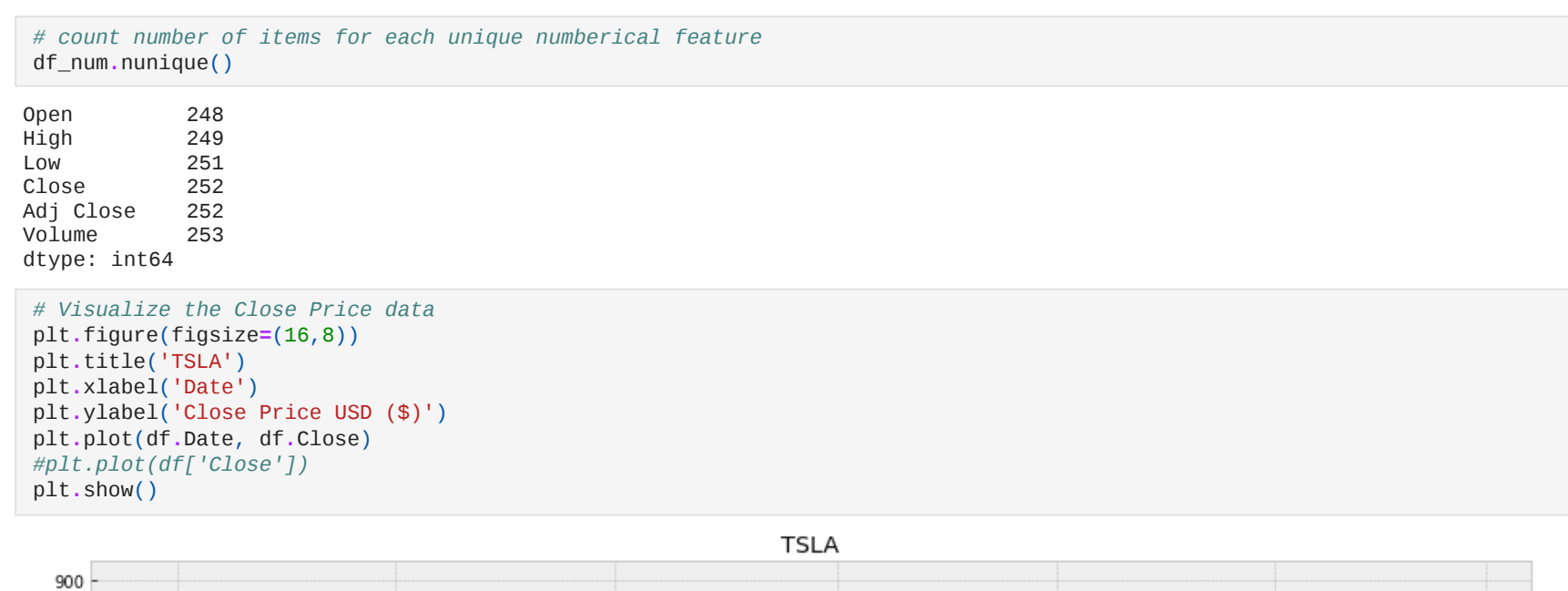
```
# count number of items for each unique numberical feature
df_num.nunique()
```

Out[12]:

```
Open         248
High         249
Low          251
Close        252
Adj Close    252
Volume       253
dtype: int64
```

In [13]:

```
# Visualize the Close Price data
plt.figure(figsize=(16,8))
plt.title('TSLA')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(df.Date, df.Close)
#plt.plot(df['Close'])
plt.show()
```



In [14]:

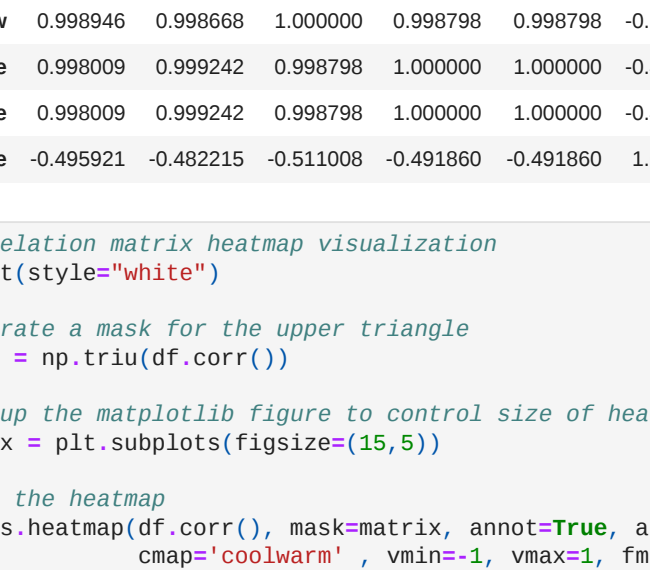
```
# look at correlations in the numerical independent variables, as well as the dependent variable
df_num.corr()
```

Out[14]:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999128	0.998946	0.998009	0.998009	-0.495921
High	0.999128	1.000000	0.998668	0.999242	0.999242	-0.482215
Low	0.998946	0.998668	1.000000	0.998798	0.998798	-0.511008
Close	0.998009	0.999242	0.998798	1.000000	1.000000	-0.491860
Adj Close	0.998009	0.999242	0.998798	1.000000	1.000000	-0.491860
Volume	-0.495921	-0.482215	-0.511008	-0.491860	-0.491860	1.000000

In [15]:

```
# correlation matrix heatmap visualization
sns.set(style="white")

# Generate a mask for the upper triangle
matrix = np.triu(df.corr())

# Set up the matplotlib figure to control size of heatmap
fig, ax = plt.subplots(figsize=(15,5))

# Plot the heatmap
_ = sns.heatmap(df.corr(), mask=matrix, annot=True, annot_kws={"size": 12}, square=True,
                cmap='coolwarm' , vmin=-1, vmax=1, fmt='.2f')  # annot=True display corr label

# _ = sns.heatmap(df.corr(), annot=True, annot_kws={"size": 12}, square=True,
#                 cmap='coolwarm' , vmin=-1, vmax=1, fmt='.2f')

# Prevent Heatmap Cut-Off Issue
# bottom, top = ax.get_ylim()
# ax.set_ylim(bottom+0.5, top-0.5)
```



In [16]:

```
# Get the Close Price
df = df[['Close']]
df.head()
```

Out[16]:

	Close
0	166.76
1	159.98
2	155.76
3	135.80
4	133.60

In [17]:

```
# Create a variable to product 'x' days out into the future
future_days = 25
# Create a new column (target) shifted 'x' units/days up
df['Prediction'] = df[['Close']].shift(-future_days)
df.head()
```

Out[17]:

	Close	Prediction
0	166.76	100.43
1	159.98	104.80
2	155.76	96.31
3	135.80	90.89
4	133.60	96.00

In [18]:

```
# Create the future dataset (X) and convert it to a numpy array and remove the last 'x' rows/days
X = np.array(df.drop(['Prediction'],1))[:-future_days]
print(X)
df.shape
```

```
[[166.76]
 [159.98]
 [155.76]
 [135.8 ]
 [133.6 ]
 [148.72]
 [149.1 ]
 [149.9 ]
 [144.91]
 [140.7 ]
 [121.6 ]
 [129.07]
 [126.85]
 [112.11]
 [109.32]
 [ 89.01]
 [ 86.04]
 [ 72.24]
 [ 85.53]
 [ 85.51]
 [ 86.86]
 [101.  ]
 [107.85]
 [105.63]
 [102.87]
 [100.43]
 [104.8 ]
 [ 96.31]
 [ 90.89]
 [ 96.  ]
 [103.25]
 [109.09]
 [109.77]
 [114.6 ]
 [130.19]
 [141.98]
 [145.97]
 [149.04]
 [150.78]
 [149.27]
 [137.34]
 [146.42]
 [141.13]
 [145.03]
 [159.75]
 [153.82]
 [160.1 ]
 [156.38]
 [140.26]
 [152.24]
 [153.64]
 [156.52]
 [156.01]
 [163.88]
 [162.26]
 [161.88]
 [158.19]
 [160.67]
 [159.83]
 [162.73]
 [161.6 ]
 [163.11]
 [165.52]
 [163.38]
 [163.77]
 [164.05]
 [161.16]
 [167.  ]
 [179.62]
 [176.31]
 [176.59]
 [172.88]
 [177.13]
 [189.98]
 [188.13]
 [205.01]
 [194.57]
 [187.06]
 [198.18]
 [196.43]
 [198.36]
 [200.79]
 [200.18]
 [198.86]
 [200.36]
 [192.17]
 [197.2 ]
 [191.95]
 [201.87]
 [215.96]
 [223.93]
 [241.73]
 [274.32]
 [277.97]
 [273.18]
 [278.86]
 [308.93]
 [299.41]
 [303.36]
 [309.2 ]
 [300.13]
 [300.17]
 [328.6 ]
 [313.67]
 [318.47]
 [302.61]
 [283.4 ]
 [307.92]
 [295.3 ]
 [299.82]
 [297.5 ]
 [286.15]
 [297.  ]
 [297.4 ]
 [297.  ]
 [297.92]
 [290.54]
 [283.71]
 [274.88]
 [310.95]
 [324.2 ]
 [330.14]
 [367.13]
 [377.42]
 [375.71]
 [400.37]
 [410.  ]
 [402.84]
 [404.67]
 [430.63]
 [447.75]
 [442.68]
 [498.32]
 [475.05]
 [447.37]
 [407.  ]
 [418.32]
 [330.21]
 [366.28]
 [371.34]
 [372.72]
 [419.62]
 [449.76]
 [441.76]
 [423.43]
 [442.15]
 [449.39]
 [424.23]
 [380.36]
 [387.79]
 [407.34]
 [421.2 ]
 [419.07]
 [429.01]
 [448.16]
 [415.09]
 [425.68]
 [413.98]
 [425.3 ]
 [425.92]
 [434.  ]
 [442.3 ]
 [446.65]
 [461.3 ]
 [448.88]
 [439.67]
 [430.83]
 [421.94]
 [422.64]
 [425.79]
 [420.63]
 [420.28]
 [424.68]
 [406.02]
 [410.83]
 [388.04]
 [400.51]
 [423.9 ]
 [420.98]
 [438.09]
 [429.95]
 [421.26]
 [410.36]
 [417.13]
 [411.76]
 [408.5 ]
 [408.09]
 [441.61]
 [486.64]
 [499.27]
 [489.61]
 [521.85]
 [555.38]
 [574.  ]
 [585.76]
 [567.6 ]
 [584.76]
 [568.82]
 [593.38]
 [599.04]
 [641.76]
 [649.88]
 [604.48]
 [627.07]
 [609.99]
 [639.83]
 [633.25]
 [622.77]
 [655.9 ]
 [695.  ]
 [649.86]
 [640.34]
 [645.98]
 [661.77]
 [663.69]
 [665.99]
 [694.78]
 [705.67]
 [729.77]
 [735.11]
 [755.98]
 [816.04]
 [880.02]
 [811.19]
 [849.44]
 [854.41]
 [845.  ]
 [826.16]]
```

Out[18]:

```
(253, 2)
```

In [19]:

```
# Create the target dataset(Y) and convert it to a numpy array and get target values except the 'x' rows/days
Y = np.array(df['Prediction'])[:-future_days]
print(Y)
df.shape
```

```
[100.43 104.8   96.31  90.89  96.   103.25 109.09 109.77 114.6  130.19
 141.98 145.97 149.04 150.78 149.27 137.34 146.42 141.13 145.03 159.75
 153.82 160.1  156.38 140.26 152.24 153.64 156.52 156.01 163.88 162.26
 161.88 158.19 160.67 159.83 162.73 161.6  163.11 165.52 163.38 163.77
 164.05 161.16 167.   179.62 176.31 176.59 172.88 177.13 189.98 188.13
 205.01 194.57 187.06 198.18 196.43 198.36 200.79 200.18 198.86 200.36
 192.17 197.2  191.95 201.87 215.96 223.93 241.73 274.32 277.97 273.18
 278.86 308.93 299.41 303.36 309.2  300.13 300.17 328.6  313.67 318.47
 302.61 283.4  307.92 295.3  299.82 297.5  286.15 297.   297.4  297.
 297.92 290.54 283.71 274.88 310.95 324.2  330.14 367.13 377.42 375.71
 400.37 410.   402.84 404.67 430.63 447.75 442.68 498.32 475.05 447.37
 407.   418.32 330.21 366.28 371.34 372.72 419.62 449.76 441.76 423.43
 442.15 449.39 424.23 380.36 387.79 407.34 421.2  419.07 429.01 448.16
 415.09 425.68 413.98 425.3  425.92 434.   442.3  446.65 461.3  448.88
 439.67 430.83 421.94 422.64 425.79 420.63 420.28 424.68 406.02 410.83
 388.04 400.51 423.9  420.98 438.09 429.95 421.26 410.36 417.13 411.76
 408.5  408.09 441.61 486.64 499.27 489.61 521.85 555.38 574.   585.76
 567.6  584.76 568.82 593.38 599.04 641.76 649.88 604.48 627.07 609.99
 639.83 633.25 622.77 655.9  695.   649.86 640.34 645.98 661.77 663.69
 665.99 694.78 705.67 729.77 735.11 755.98 816.04 880.02 811.19 849.44
 854.41 845.   826.16 844.55 850.45 844.99 846.64 880.8  883.09 864.16
 835.43 793.53 839.81 872.79 854.69 849.99 852.23 863.42 849.46 804.82
 811.66 816.12 796.22 798.15 787.38 781.3  714.5  698.84]
```

Out[19]:

```
(253, 2)
```

Train and Test the dataset

In [20]:

```
# Split the data into 80% training and 20% testing
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2)
```

In [21]:

```
Rtree = RandomForestRegressor(n_estimators=100, random_state = 50)
```

In [22]:

```
# Create the Models
# Create the decision tree regressor model
tree = DecisionTreeRegressor().fit(X_train, Y_train)
# Create the linear regression model
lr = LinearRegression().fit(X_train, Y_train)
# Create the Random Tree regression model
Rforest = RandomForestRegressor().fit(X_train, Y_train)
```

Model Evaluation - R Square, MSE, MAE

In [23]:

```
# Create a Gaussian Regressor - It means that the model has successfully explained x% variance in our dataset
Rforest_confidence = Rforest.score(X_test, Y_test)
print("Random Forest confidence (R Square): ", Rforest_confidence)
```

```
Random Forest confidence (R Square):  0.937541693506432
```

In [24]:

```
# https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56
# https://www.youtube.com/watch?v=TtIjAiSojFE&t=311s

forest = RandomForestRegressor()
forest.fit(X_train, Y_train)
```

Out[24]:

```
RandomForestRegressor()
```

In [25]:

```
Y_pred_RFor = forest.predict(X_test)
Y_pred_RFor
```

Out[25]:

```
array([159.2728, 156.3664, 156.3002, 205.7271, 299.3722, 171.2497,
       434.3128, 454.5168, 425.9368, 861.0584, 199.9994, 771.3127,
       199.9994, 125.5586, 273.1664, 146.6549, 315.2309, 838.396 ,
       273.1664, 127.1098, 130.9148, 430.4282, 134.7931, 349.9444,
       843.2802, 845.4096, 146.6549, 151.8585, 146.6549, 169.4689,
       173.3523, 197.2262, 847.6828, 419.6008, 305.7507, 299.095 ,
       862.6987, 462.6992, 430.3303, 413.038 , 434.7149, 436.773 ,
       408.068 , 199.5962, 158.257 , 305.9727])
```

In [26]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('MSE (RForest):',mean_squared_error(Y_test,Y_pred_RFor))
print('MAE (RForest):',mean_absolute_error(Y_test, Y_pred_RFor))
```

```
MSE (RForest): 3409.2186878391253
MAE (RForest): 33.36201304347835
```

In [27]:

```
# Testing Model: Score returns the coefficient of determination R^2 of the prediction
# The best possible score is 1.0
lr_confidence = lr.score(X_test, Y_test)
print("linear confidence (R Square): ", lr_confidence)
```

```
linear confidence (R Square):  0.9156896923008622
```

In [28]:

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

Out[28]:

```
LinearRegression()
```

In [29]:

```
Y_pred_Lin = linear.predict(X_test)
Y_pred_Lin
```

Out[29]:

```
array([196.59328102, 161.15576714, 160.66067614, 217.15606072,
       260.62505088, 209.42163904, 390.76797284, 497.06951243,
       532.58404046, 762.00921178, 220.68770989, 980.41035673,
       220.56668764, 204.61375528, 235.26538945, 134.46486101,
       257.34644824, 755.38599434, 234.67128025, 212.02911832,
       216.52894545, 507.62045183, 224.12034085, 367.14663093,
       730.29038144, 725.55951185, 135.04796819, 174.17115954,
       119.8651774 , 212.43619315, 216.14387467, 219.42247732,
       671.90264902, 385.48700213, 246.19039761, 261.29617425,
       804.78507453, 488.16887638, 369.79811831, 352.52494327,
       499.31392499, 506.76229409, 380.27204355, 220.87474426,
       150.8798783 , 249.40298812])
```

In [30]:

```
print('MSE (Linear):',mean_squared_error(Y_test,Y_pred_Lin))
print('MAE (Linear):',mean_absolute_error(Y_test, Y_pred_Lin))
```

```
MSE (Linear): 4551.5782175674185
MAE (Linear): 50.639425028925416
```

In [31]:

```
tree_confidence = tree.score(X_test, Y_test)
print("Decision Tree confidence (R Square): ", tree_confidence)
```

```
Decision Tree confidence (R Square):  0.9237677129434003
```

In [32]:

```
Dtree = DecisionTreeRegressor()
Dtree.fit(X_train, Y_train)
```

Out[32]:

```
DecisionTreeRegressor()
```

In [33]:

```
Y_pred_Dtree = tree.predict(X_test)
Y_pred_Dtree
```

Out[33]:

```
array([167.  , 158.19, 158.19, 241.73, 299.82, 176.59, 442.15, 410.36,
       415.09, 872.79, 201.87, 787.38, 201.87, 109.09, 273.18, 145.03,
       318.47, 835.43, 273.18,  96.31, 104.8 , 424.68, 100.43, 330.21,
       846.64, 846.64, 145.03, 149.04, 145.03, 194.57, 198.86, 197.2 ,
       880.02, 423.43, 309.2 , 299.82, 864.16, 388.04, 447.37, 442.68,
       442.3 , 424.68, 375.71, 201.87, 160.1 , 309.2 ])
```

In [34]:

```
print('MSE (Decision Tree):',mean_squared_error(Y_test,Y_pred_Dtree))
print('MAE (Decision Tree):',mean_absolute_error(Y_test, Y_pred_Dtree))
```

```
MSE (Decision Tree): 4115.4780086956525
MAE (Decision Tree): 37.98434782608695
```

In [35]:

```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_rbf.fit(X_train, Y_train)
```

Out[35]:

```
SVR(C=1000.0, gamma=0.1)
```

In [36]:

```
svm_confidence = svr_rbf.score(X_test, Y_test)
print("SVM confidence: ", svm_confidence)
```

```
SVM confidence:  0.6094434574410632
```

In [37]:

```
Y_pred_SVM = svr_rbf.predict(X_test)
Y_pred_SVM
```

Out[37]:

```
array([151.9810692 , 155.12704913, 157.26556897, 202.11659798,
       298.73594965, 178.4075081 , 493.40761296, 366.14298634,
       440.91949708, 514.13793152, 209.62463838, 516.2201271 ,
       207.92880853, 111.18209626, 264.30527695, 145.27796815,
       316.74298503, 834.53215784, 270.07308409, 186.4001134 ,
       199.99142941, 418.21792247,  96.14715611, 373.91119968,
       501.10444395, 493.51541678, 145.51361828, 472.54528553,
       493.5094998 , 192.38584136, 198.43502384, 196.10116209,
       531.47597554, 558.23909748, 346.85879655, 301.53953308,
       862.26931808, 610.00230744, 402.51207189, 436.22255035,
       412.60253579, 434.62054168, 411.35575268, 212.02802973,
       173.23269643, 288.96375642])
```

In [38]:

```
print('MSE (SVM):',mean_squared_error(Y_test,Y_pred_SVM))
print('MAE (SVM):',mean_absolute_error(Y_test, Y_pred_SVM))
```

```
MSE (SVM): 21084.594521742893
MAE (SVM): 86.66977213451734
```

Summary of R Squared, MSE,MAE

In [39]:

```
print("Random Forest confidence (R Square): ", Rforest_confidence)
print('MSE (RForest):',mean_squared_error(Y_test,Y_pred_RFor))
print('MAE (RForest):',mean_absolute_error(Y_test, Y_pred_RFor))
print()
print("Decision Tree confidence (R Square): ", tree_confidence)
print('MSE (Decision Tree):',mean_squared_error(Y_test,Y_pred_Dtree))
print('MAE (Decision Tree):',mean_absolute_error(Y_test, Y_pred_Dtree))
print()
print("linear confidence (R Square): ", lr_confidence)
print('MSE (Linear):',mean_squared_error(Y_test,Y_pred_Lin))
print('MAE (Linear):',mean_absolute_error(Y_test, Y_pred_Lin))
print()
print("SVM confidence: ", svm_confidence)
print('MSE (SVM):',mean_squared_error(Y_test,Y_pred_SVM))
print('MAE (SVM):',mean_absolute_error(Y_test, Y_pred_SVM))
```

```
Random Forest confidence (R Square):  0.937541693506432
MSE (RForest): 3409.2186878391253
MAE (RForest): 33.36201304347835

Decision Tree confidence (R Square):  0.9237677129434003
MSE (Decision Tree): 4115.4780086956525
MAE (Decision Tree): 37.98434782608695

linear confidence (R Square):  0.9156896923008622
MSE (Linear): 4551.5782175674185
MAE (Linear): 50.639425028925416

SVM confidence:  0.6094434574410632
MSE (SVM): 21084.594521742893
MAE (SVM): 86.66977213451734
```

Visualizing the data

In [40]:

```
# Get the last 'x' rows of the feature dataset
X_future = df.drop(['Prediction'],1)[:-future_days]
X_future = X_future.tail(future_days)
X_future = np.array(X_future)
X_future
```

Out[40]:

```
array([[627.07],
       [609.99],
       [639.83],
       [633.25],
       [622.77],
       [655.9 ],
       [695.  ],
       [649.86],
       [640.34],
       [645.98],
       [661.77],
       [663.69],
       [665.99],
       [694.78],
       [705.67],
       [729.77],
       [735.11],
       [755.98],
       [816.04],
       [880.02],
       [811.19],
       [849.44],
       [854.41],
       [845.  ],
       [826.16]])
```

In [41]:

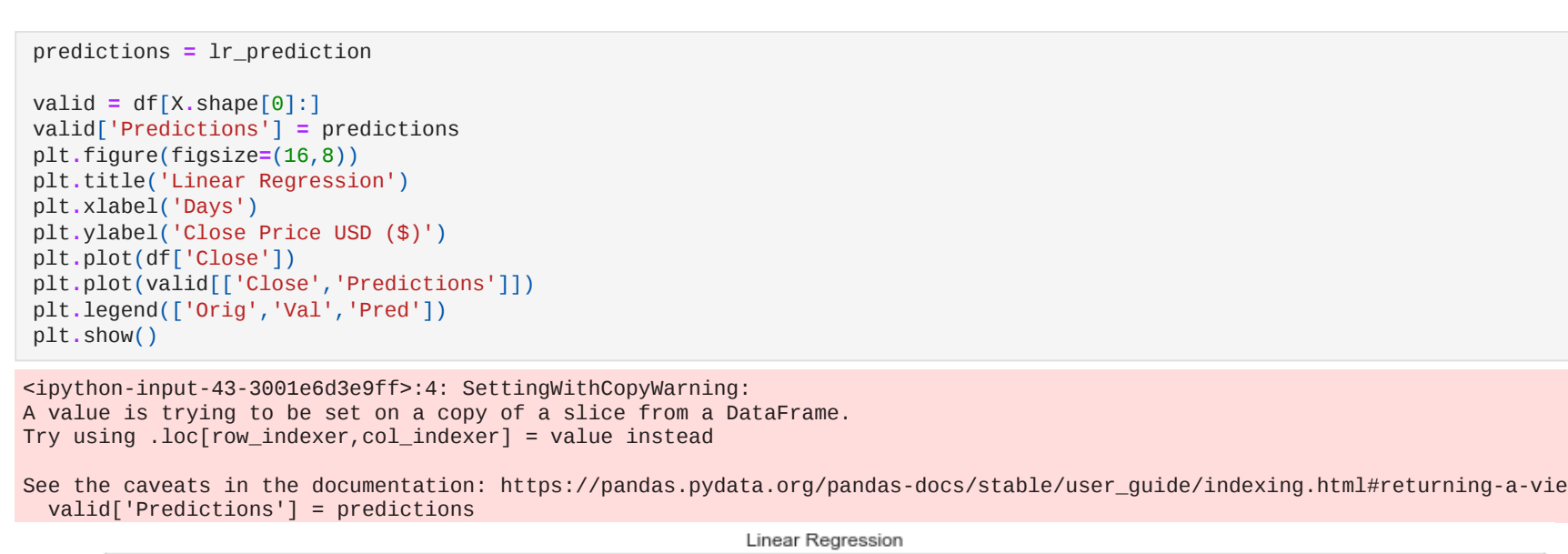
```
# Show the model tree prediction
print('Prediction - Decision Tree')
tree_prediction = tree.predict(X_future)
print(tree_prediction)
print()
# Show the model linear regression prediction
print('Prediction - Linear Regression')
lr_prediction = lr.predict(X_future)
print(lr_prediction)
print()
print('Prediction - Random Forest')
Rforest_prediction = Rforest.predict(X_future)
print(Rforest_prediction)
print()
print('Prediction - SVM')
SVM_prediction = svr_rbf.predict(X_future)
print(SVM_prediction)
```

```
Prediction - Decision Tree
[846.64 850.45 844.99 846.64 846.64 872.79 864.16 835.43 793.53 839.81
 872.79 854.69 849.99 864.16 863.42 849.46 804.82 811.66 816.12 796.22
 798.15 787.38 787.38 714.5  698.84]

Prediction - Linear Regression
[ 730.29038144  711.49892733  744.32896192  737.08963123  725.55951185
  762.00921178  805.02711902  755.3639903   744.89006505  751.09520564
  768.46739887  770.57978716  773.11025229  804.78507453  816.76627683
  843.28115061  849.15623052  872.11745109  938.19559711 1008.58653588
  932.85961629  974.94235164  980.41035673  970.05745373  949.32964369]

Prediction - Random Forest
[842.0956 841.931  833.0303 841.9454 846.1756 861.1144 862.8716 837.0718
 812.4463 838.7362 865.0696 858.1916 852.3416 862.8716 860.5515 840.4315
 822.5947 817.7691 799.6994 779.7359 792.6911 760.3941 767.2107 735.3034
 730.4001]

Prediction - SVM
[501.10444395 850.34996993 839.41177862 846.53973604 493.51541678
 514.13793152 864.06001656 835.33019634 847.86618394 839.70982541
 872.68999276 854.79044613 849.88969774 862.26931808 863.31999521
 849.36005709 804.72026442 811.55954261 816.02000027 796.12043448
 798.05035644 787.27994232 516.2201271  714.40041488 698.73958166]
```

In [42]:

```
# Visualize the data
predictions = tree_prediction

valid = df[X.shape[0]:]
valid['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Decision Tree')
plt.xlabel('Days')
plt.ylabel('Close Price USD ($)')
plt.plot(df['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Orig','Val','Pred'])
plt.show()
```

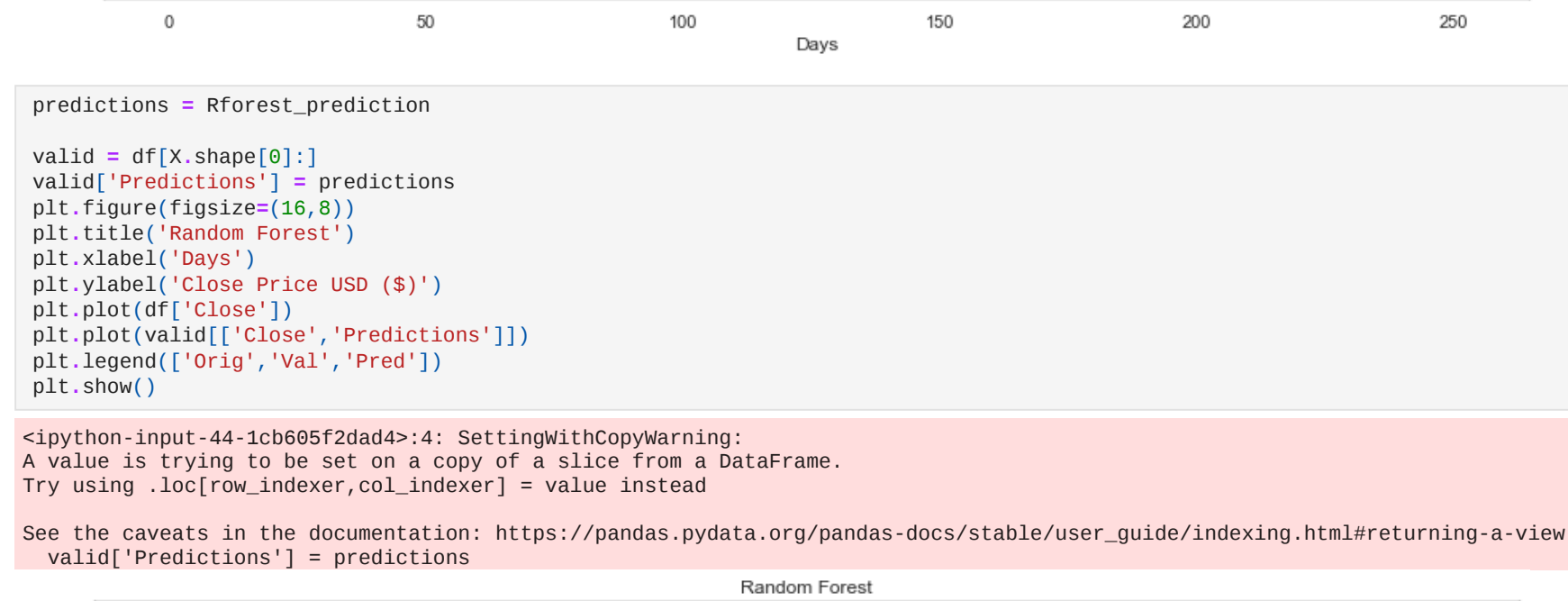
```
<ipython-input-42-46f76765a9c0>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions
```



In [43]:

```
predictions = lr_prediction

valid = df[X.shape[0]:]
valid['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Linear Regression')
plt.xlabel('Days')
plt.ylabel('Close Price USD ($)')
plt.plot(df['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Orig','Val','Pred'])
plt.show()
```

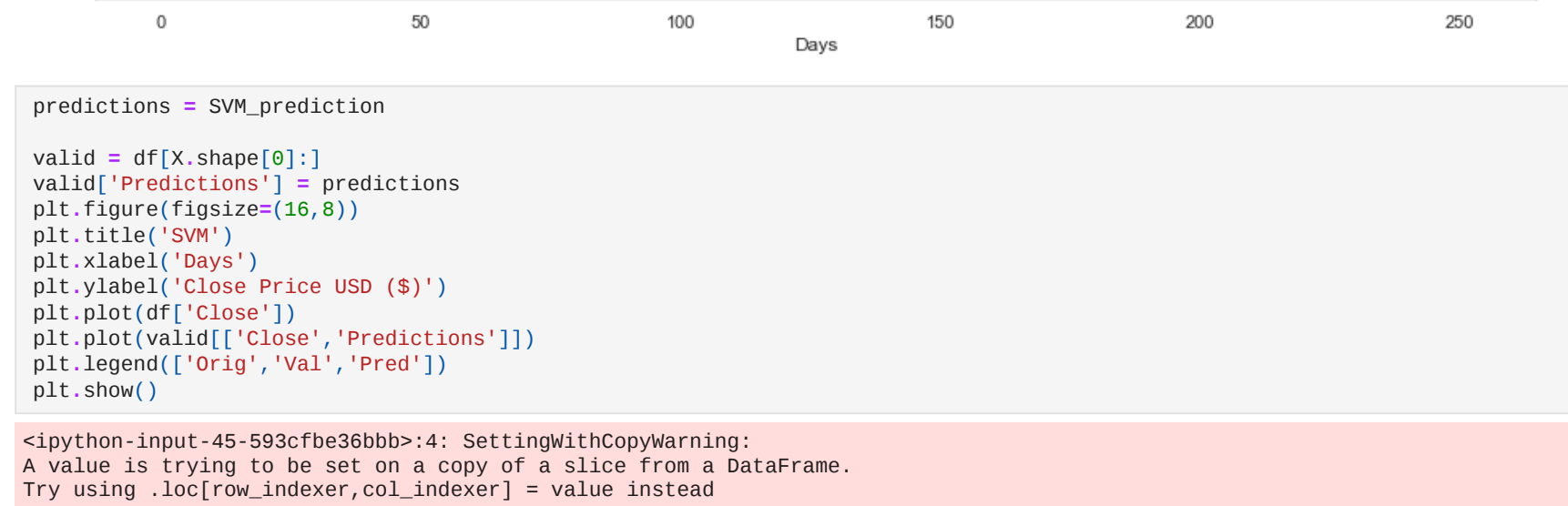
```
<ipython-input-43-3091e6d3e9ff>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions
```



In [44]:

```
predictions = Rforest_prediction

valid = df[X.shape[0]:]
valid['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Random Forest')
plt.xlabel('Days')
plt.ylabel('Close Price USD ($)')
plt.plot(df['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Orig','Val','Pred'])
plt.show()
```

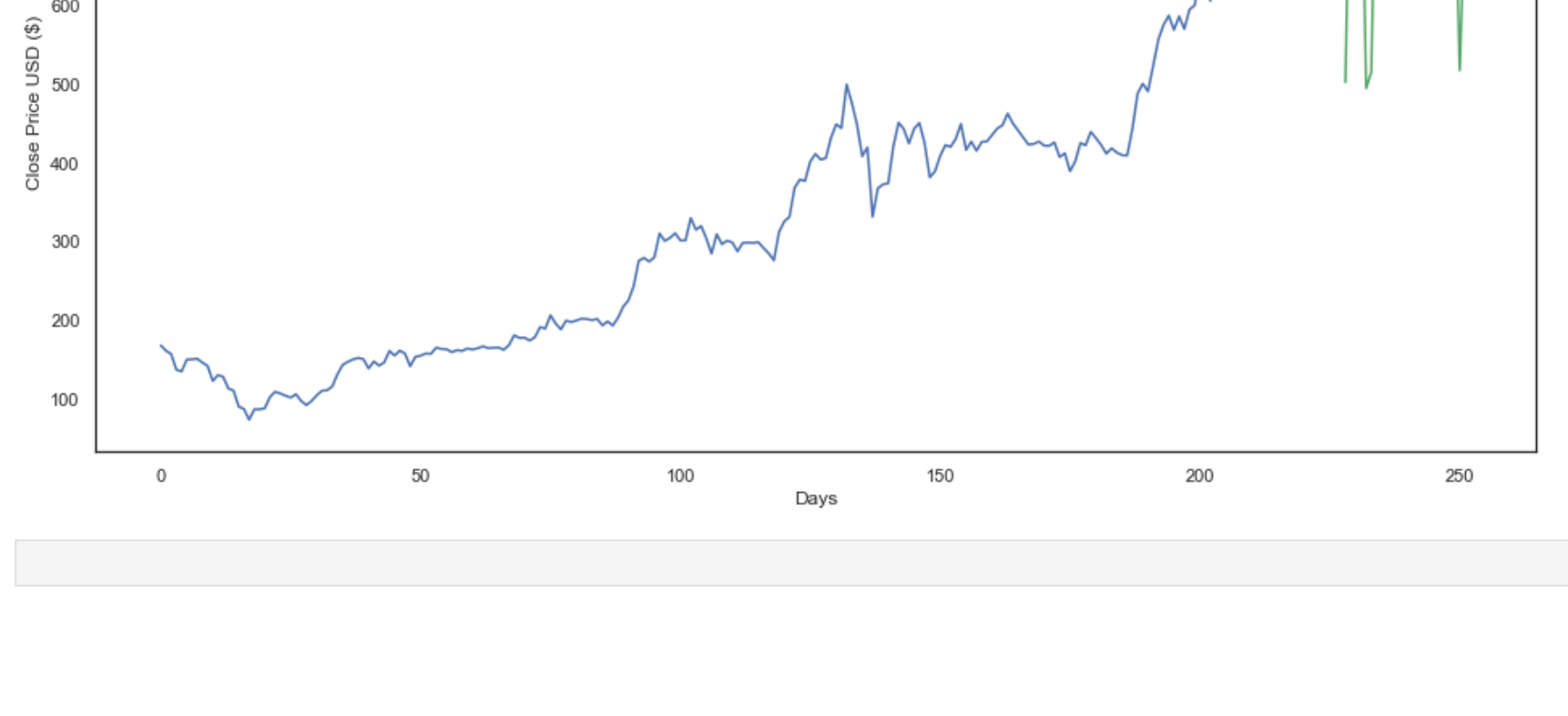
```
<ipython-input-44-1cb605f2dad4>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions
```



In [45]:

```
predictions = SVM_prediction

valid = df[X.shape[0]:]
valid['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('SVM')
plt.xlabel('Days')
plt.ylabel('Close Price USD ($)')
plt.plot(df['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Orig','Val','Pred'])
plt.show()
```

```
<ipython-input-45-593cfbe36bbb>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions
```



In [ ]: