Bitcoin Currency Predictor- Sutow Brett

July 13, 2021

[1]:
```
#This notebook will try to predict cryptocurrency price changes#
#The goal is to be able to accurately use machine learning to predict the next␣
 ↪60 days of changes#
#We will utilize machine learning to see the results of this process#
#Setup#
import math
import pandas as pd
import pandas_datareader as web
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
from statistics import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

[2]:
```
#Pulls Data#
#Prints head and tail#
BTC = pd.read_csv('/Users/Brett/Desktop/Bitcoin.csv')
print(BTC.head())
print(BTC.tail())
```

```
     Date   Open   High    Low  Close
0  1/1/21  28965  29650  28754  29259
1  1/2/21  29243  33219  29050  31690
2  1/3/21  31714  34524  31494  33581
3  1/4/21  33581  33649  28258  31027
4  1/5/21  31029  34221  29991  33788
       Date   Open   High    Low  Close
185  7/5/21  35542  35945  33171  33847
186  7/6/21  33847  35058  33601  33941
187  7/7/21  33941  35040  33724  34550
188  7/8/21  34550  34550  32142  32788
189  7/9/21  32788  33667  32302  33508
```

[3]:
```
BTC
```

[3]:

	Date	Open	High	Low	Close
0	1/1/21	28965	29650	28754	29259
1	1/2/21	29243	33219	29050	31690
2	1/3/21	31714	34524	31494	33581
3	1/4/21	33581	33649	28258	31027
4	1/5/21	31029	34221	29991	33788
..	…	…	…	…	…
185	7/5/21	35542	35945	33171	33847
186	7/6/21	33847	35058	33601	33941
187	7/7/21	33941	35040	33724	34550
188	7/8/21	34550	34550	32142	32788
189	7/9/21	32788	33667	32302	33508

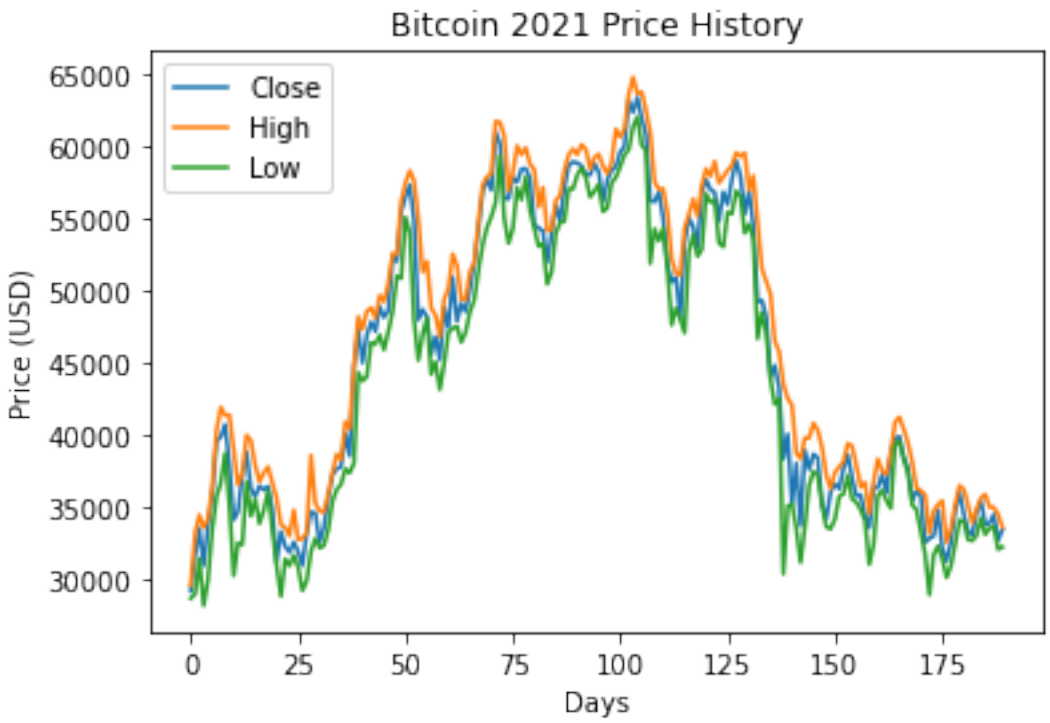
[190 rows x 5 columns]

[4]:
```
#Describes for graphing#

BTC.describe()
```

[4]:

	Open	High	Low	Close
count	190.000000	190.000000	190.000000	190.000000
mean	45266.842105	46860.136842	43436.805263	45279.126316
std	10126.846413	10106.467886	10103.933379	10098.741809
min	28965.000000	29650.000000	28258.000000	29259.000000
25%	35876.250000	37262.500000	33775.000000	35831.500000
50%	45152.000000	47750.000000	43838.000000	45147.000000
75%	55805.000000	57341.500000	53600.250000	55810.250000
max	63381.000000	64788.000000	62034.000000	63381.000000

[5]:
```
#Price Changes#
plt.figure()
plt.plot(BTC["Close"])
plt.plot(BTC["High"])
plt.plot(BTC["Low"])
plt.title('Bitcoin 2021 Price History')
plt.ylabel('Price (USD)')
plt.xlabel('Days')
plt.legend(['Close','High', 'Low'], loc='upper left')

plt.show()
```



[6]:
```
#Checks real-time vs predictions using LSTM model#
#Creating a new dataset filtering by closing price#
closedata=BTC.filter(['Close'])
newdata= closedata.values

#Training LSTM#
trainingdata=math.ceil(len(newdata) * .8)
trainingdata
```

[6]: 152

[7]:
```
#Scales data to make it easier with using LSTM Model#
scaler = MinMaxScaler(feature_range=(0,1))
datascaled= scaler.fit_transform(newdata)
datascaled
```

[7]:
```
array([[0.        ],
       [0.07124436],
       [0.12666315],
       [0.05181408],
       [0.13272962],
       [0.19310123],
       [0.30294238],
```

```
[0.31264287],
[0.33731903],
[0.25570013],
[0.14433503],
[0.16139148],
[0.23413047],
[0.28146064],
[0.20705117],
[0.19125491],
[0.21308833],
[0.20502901],
[0.21203329],
[0.16508411],
[0.05908212],
[0.12062599],
[0.09029365],
[0.07839517],
[0.09917355],
[0.08012426],
[0.051902  ],
[0.11766602],
[0.15989684],
[0.1579626 ],
[0.09867534],
[0.12918352],
[0.18835355],
[0.23518551],
[0.24772874],
[0.25186097],
[0.31780083],
[0.27404607],
[0.45536604],
[0.53039095],
[0.46245824],
[0.51825802],
[0.54492703],
[0.52608288],
[0.57622648],
[0.55459821],
[0.56804994],
[0.67622062],
[0.66819061],
[0.76882949],
[0.79837055],
[0.82272434],
[0.7520075 ],
[0.549704  ],
```

[0.57033585],
[0.55506711],
[0.47951468],
[0.51535666],
[0.46878846],
[0.57396987],
[0.53390774],
[0.6357482],
[0.5470078],
[0.58258601],
[0.56632085],
[0.6108962],
[0.66341363],
[0.7342477],
[0.8110017],
[0.83412461],
[0.81243772],
[0.93027959],
[0.90598441],
[0.79535197],
[0.79479515],
[0.83743626],
[0.82861497],
[0.85478577],
[0.85595803],
[0.83620538],
[0.74295176],
[0.73521482],
[0.73020339],
[0.66613915],
[0.72548502],
[0.78122619],
[0.75036633],
[0.81815251],
[0.86196589],
[0.86961491],
[0.8673583],
[0.86149698],
[0.84414747],
[0.84318035],
[0.86603951],
[0.84945197],
[0.78828908],
[0.83295235],
[0.85267569],
[0.86050056],
[0.88939687],

```
[0.90123674],
[0.99457828],
[0.96999004],
[1.        ],
[0.95841393],
[0.92561983],
[0.79069222],
[0.7888166 ],
[0.80865717],
[0.75523123],
[0.66186038],
[0.6260184 ],
[0.63583612],
[0.55143309],
[0.70447219],
[0.75634488],
[0.73843854],
[0.69594397],
[0.80722115],
[0.83350917],
[0.81346345],
[0.8078952 ],
[0.74975089],
[0.80871578],
[0.78330696],
[0.83347987],
[0.87201805],
[0.84177364],
[0.7639939 ],
[0.81129477],
[0.7409003 ],
[0.58595627],
[0.58882832],
[0.55670828],
[0.43382568],
[0.45671414],
[0.41202157],
[0.26759862],
[0.31794737],
[0.17595686],
[0.25868941],
[0.13281754],
[0.28544634],
[0.24608757],
[0.27759217],
[0.27011898],
[0.17542934],
```

```
       [0.14673817],
       [0.20092609],
       [0.2173964 ],
       [0.20722701],
       [0.24852002],
       [0.27647852],
       [0.2289432 ],
       [0.19245648],
       [0.19453725],
       [0.15274603],
       [0.12833363],
       [0.20945431],
       [0.20957154],
       [0.23656292],
       [0.1987574 ],
       [0.28049352],
       [0.30918469],
       [0.31328762],
       [0.27228767],
       [0.248901  ],
       [0.18269738],
       [0.203212  ],
       [0.19324776],
       [0.09855811],
       [0.10655882],
       [0.11157025],
       [0.16405838],
       [0.08654241],
       [0.06066467],
       [0.09932009],
       [0.16007268],
       [0.20212766],
       [0.15608698],
       [0.12168103],
       [0.11327003],
       [0.15989684],
       [0.1841334 ],
       [0.13445871],
       [0.13721353],
       [0.15506125],
       [0.10342301],
       [0.12452377]])
```

[8]:
```
#Create training set#
train_data=datascaled[0:trainingdata, :]

#Splits data into x and y train#
```

```
x_train=[]
y_train=[]
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i <= 60:
        print(x_train)
        print(y_train)
        print()
```

```
[array([0.        , 0.07124436, 0.12666315, 0.05181408, 0.13272962,
       0.19310123, 0.30294238, 0.31264287, 0.33731903, 0.25570013,
       0.14433503, 0.16139148, 0.23413047, 0.28146064, 0.20705117,
       0.19125491, 0.21308833, 0.20502901, 0.21203329, 0.16508411,
       0.05908212, 0.12062599, 0.09029365, 0.07839517, 0.09917355,
       0.08012426, 0.051902  , 0.11766602, 0.15989684, 0.1579626 ,
       0.09867534, 0.12918352, 0.18835355, 0.23518551, 0.24772874,
       0.25186097, 0.31780083, 0.27404607, 0.45536604, 0.53039095,
       0.46245824, 0.51825802, 0.54492703, 0.52608288, 0.57622648,
       0.55459821, 0.56804994, 0.67622062, 0.66819061, 0.76882949,
       0.79837055, 0.82272434, 0.7520075 , 0.549704  , 0.57033585,
       0.55506711, 0.47951468, 0.51535666, 0.46878846, 0.57396987])]
[0.5339077428052283]
```

[9]:
```
#Convert to arrays#
x_train, y_train = np.array(x_train), np.array(y_train)
```

[10]:
```
#Reshapes data#
x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
x_train
```

[10]:
```
array([[[0.        ],
        [0.07124436],
        [0.12666315],
        …,
        [0.51535666],
        [0.46878846],
        [0.57396987]],

       [[0.07124436],
        [0.12666315],
        [0.05181408],
        …,
        [0.46878846],
        [0.57396987],
        [0.53390774]],
```

```
       [[0.12666315],
        [0.05181408],
        [0.13272962],
        …,
        [0.57396987],
        [0.53390774],
        [0.6357482 ]],

       …,

       [[0.86961491],
        [0.8673583 ],
        [0.86149698],
        …,
        [0.27011898],
        [0.17542934],
        [0.14673817]],

       [[0.8673583 ],
        [0.86149698],
        [0.84414747],
        …,
        [0.17542934],
        [0.14673817],
        [0.20092609]],

       [[0.86149698],
        [0.84414747],
        [0.84318035],
        …,
        [0.14673817],
        [0.20092609],
        [0.2173964 ]]])
```

[11]:
```
#LSTM model being built to test what it should be#
lstmmodel= Sequential()
lstmmodel.add(LSTM(50, return_sequences=True, input_shape=(60,1)))
lstmmodel.add(LSTM(50, return_sequences=False))
lstmmodel.add(Dense(25))
lstmmodel.add(Dense(1))
```

[12]:
```
#Model Continues MSE#
lstmmodel.compile(optimizer='adam', loss= 'mean_squared_error')
```

[13]:
```
#LSTM Model Trained#
lstmmodel.fit(x_train, y_train, batch_size=1, epochs=3)
```

```
Epoch 1/3
92/92 [==============================] - 8s 44ms/step - loss: 0.0938
Epoch 2/3
92/92 [==============================] - 3s 37ms/step - loss: 0.0316
Epoch 3/3
92/92 [==============================] - 4s 40ms/step - loss: 0.0183
```

[13]:
```
<tensorflow.python.keras.callbacks.History at 0x7fe86518c1c0>
```

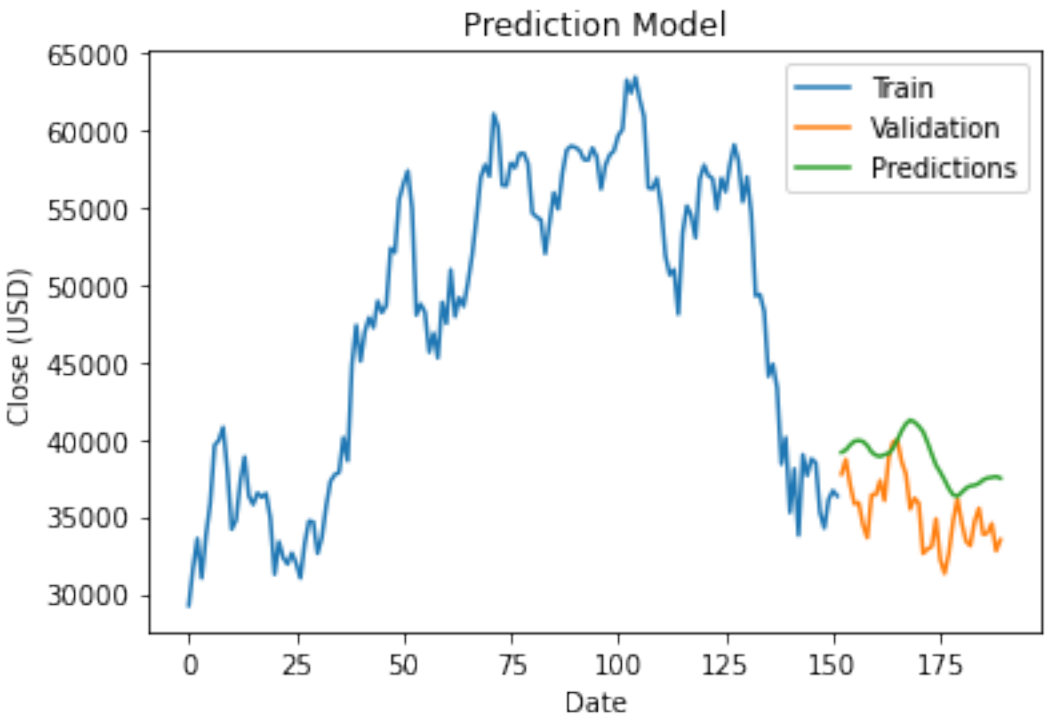
[14]:
```
#Testing Dataset#
testdata=datascaled[trainingdata -60:, :]

x_test= []
y_test= newdata[trainingdata:,:]
for i in range(60, len(testdata)):
    x_test.append(testdata[i-60:i,0])
```

[15]:
```
#Convert to numpy#
x_test=np.array(x_test)
#Reshapes#
x_test=np.reshape(x_test, (38, 60,1))
```

[16]:
```
#Prediction for Xtest#
pred= lstmmodel.predict(x_test)
pred= scaler.inverse_transform(pred)
```

[17]:
```
#Plotting Data to show actual verus predictions from the LSTM model#
train= closedata[:trainingdata]
val= closedata[trainingdata:]
val['Predictions']= pred
plt.figure()
plt.title('Prediction Model')
plt.xlabel('Date')
plt.ylabel('Close (USD)')
plt.plot(train['Close'])
plt.plot(val[['Close','Predictions']])
plt.legend(['Train', 'Validation','Predictions'])
plt.show()
```

```
<ipython-input-17-49734c30c769>: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
  val['Predictions']= pred
```



[18]:
```
#Shows what the predictions based off previous days#
val
```

[18]:

	Close	Predictions
152	37739	39154.453125
153	38693	39301.636719
154	37071	39629.289062
155	35826	39869.339844
156	35897	39915.570312
157	34471	39852.937500
158	33638	39592.757812
159	36406	39166.296875
160	36410	38955.527344
161	37331	38886.593750
162	36041	38998.625000
163	38830	39078.566406
164	39809	39406.671875
165	39949	39937.660156
166	38550	40546.492188
167	37752	40999.269531
168	35493	41237.195312
169	36193	41101.507812
170	35853	40836.386719

```
171  32622  40486.046875
172  32895  39807.242188
173  33066  39042.945312
174  34857  38321.871094
175  32212  37870.550781
176  31329  37367.949219
177  32648  36809.332031
178  34721  36400.273438
179  36156  36314.195312
180  34585  36562.625000
181  33411  36835.078125
182  33124  36985.843750
183  34715  37024.105469
184  35542  37143.855469
185  33847  37376.738281
186  33941  37490.500000
187  34550  37534.449219
188  32788  37590.375000
189  33508  37478.371094
```

[19]:
```
#Setup for prediction column#
futuredays= 60
BTC['Prediction'] = BTC[['Close']].shift(-futuredays)
```

[20]:
```
#Checks to make sure prediction column was added#
BTC
```

[20]:
```
       Date   Open   High    Low  Close  Prediction
0    1/1/21  28965  29650  28754  29259     47477.0
1    1/2/21  29243  33219  29050  31690     50952.0
2    1/3/21  31714  34524  31494  33581     47924.0
3    1/4/21  33581  33649  28258  31027     49138.0
4    1/5/21  31029  34221  29991  33788     48583.0
..      …    …    …    …    …          …
185  7/5/21  35542  35945  33171  33847         NaN
186  7/6/21  33847  35058  33601  33941         NaN
187  7/7/21  33941  35040  33724  34550         NaN
188  7/8/21  34550  34550  32142  32788         NaN
189  7/9/21  32788  33667  32302  33508         NaN

[190 rows x 6 columns]
```

[21]:
```
#Creates independent data for predicting#
X= np.array(BTC[['Close']])
X= X[:-futuredays]
```

[22]:
```
#Creates dependent variable for predicting#
y= BTC['Prediction'].values
y=y[:-futuredays]
```

[23]:
```
#Splits for training purposes#
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = .05)
```

[24]:
```
#Creates modelling for training#
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train, y_train)
```

[24]:
```
LinearRegression()
```

[25]:
```
#Test confidence to see if it is good#
LR_Conf= LR.score(x_test, y_test)
LR_Conf
```

[25]:
```
0.624061023610036
```

[26]:
```
#Creating projections for the last 60 days#
x_projection = np.array(BTC[['Close']]) [-futuredays:]
x_projection
```

[26]:
```
array([[56942],
       [54540],
       [49253],
       [49351],
       [48255],
       [44062],
       [44843],
       [43318],
       [38390],
       [40108],
       [35263],
       [38086],
       [33791],
       [38999],
       [37656],
       [38731],
       [38476],
       [35245],
       [34266],
       [36115],
       [36677],
       [36330],
       [37739],
```

```
       [38693],
       [37071],
       [35826],
       [35897],
       [34471],
       [33638],
       [36406],
       [36410],
       [37331],
       [36041],
       [38830],
       [39809],
       [39949],
       [38550],
       [37752],
       [35493],
       [36193],
       [35853],
       [32622],
       [32895],
       [33066],
       [34857],
       [32212],
       [31329],
       [32648],
       [34721],
       [36156],
       [34585],
       [33411],
       [33124],
       [34715],
       [35542],
       [33847],
       [33941],
       [34550],
       [32788],
       [33508]])
```

[27]:
```
#Prints prediction for the next 60 days#
#Based off our machine learning model we can see what is expect for the next 60␣
 ↪days from 7/9
LR_pred= LR.predict(x_projection)
LR_pred
```

[27]:
```
array([42494.91235752, 44058.58222569, 47500.34850088, 47436.55181184,
       48150.03315052, 50879.62006028, 50371.1995078 , 51363.9541077 ,
       54572.01618528, 53453.62116716, 56607.65135504, 54769.91611864,
```

```
       57565.90366393, 54175.56533194, 55049.84036648, 54350.02974688,
       54516.03133571, 56619.36911425, 57256.68501806, 56053.01075234,
       55687.15627028, 55913.04862842, 54995.80847678, 54374.76723855,
       55430.66754087, 56241.14588636, 56194.92583613, 57123.23276037,
       57665.50461723, 55863.57364508, 55860.96969859, 55261.41101892,
       56101.18376244, 54285.58207121, 53648.26616741, 53557.1280402 ,
       54467.85832562, 54987.34565068, 56457.92443178, 56002.23379576,
       56223.56924754, 58326.90702608, 58149.18767803, 58037.86896552,
       56871.95192393, 58593.81154146, 59168.63272947, 58309.98137389,
       56960.48610465, 56026.32030081, 57049.02028536, 57813.27858062,
       58000.11174139, 56964.39202438, 56426.02608726, 57529.44841305,
       57468.2556705 , 57071.80481716, 58218.84324668, 57750.1328782 ])
```

[31]:
```
#Saves predictions as CSV, I find it easier to work with#
from numpy import asarray
from numpy import savetxt
savetxt('BitcoinPred.csv', LR_pred, delimiter=',')
```

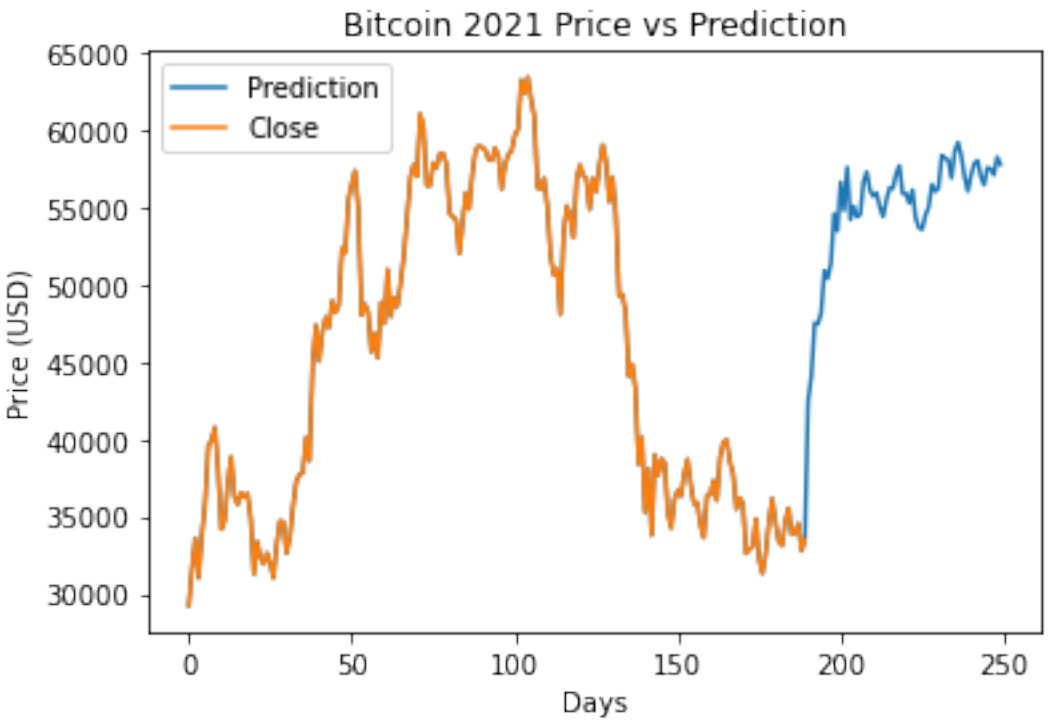
[33]:
```
BTCPred= pd.read_csv('/Users/Brett/Desktop/BitcoinPred.csv')
BTCPred
```

[33]:

	Date	Close
0	1/1/21	29259
1	1/2/21	31690
2	1/3/21	33581
3	1/4/21	31027
4	1/5/21	33788
..	…	…
245	9/3/21	57529
246	9/4/21	57468
247	9/5/21	57072
248	9/6/21	58219
249	9/7/21	57750

[250 rows x 2 columns]

[34]:
```
#This plots the prediction vs the actual close. Showing what we are predicting␣
 ↪is going to happen#
plt.figure()
plt.plot(BTCPred['Close'])
plt.plot(BTC["Close"])
plt.title('Bitcoin 2021 Price vs Prediction')
plt.ylabel('Price (USD)')
plt.xlabel('Days')
plt.legend(['Prediction','Close'], loc='upper left')

plt.show()
```


Bitcoin 2021 Price vs Prediction
Prediction
Close
Price (USD)
65000
60000
55000
50000
45000
40000
35000
30000
0
50
100
150
200
250
Days

[ ]: