gnl8yhb8y

March 11, 2024

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[6]:
```
df = pd.read_csv('/content/Twitter Stock Market Dataset.csv')
print(df)
```

```
            Date       Open       High        Low      Close  Adj Close  \
0     2013-11-07  45.099998  50.090000  44.000000  44.900002  44.900002
1     2013-11-08  45.930000  46.939999  40.685001  41.650002  41.650002
2     2013-11-11  40.500000  43.000000  39.400002  42.900002  42.900002
3     2013-11-12  43.660000  43.779999  41.830002  41.900002  41.900002
4     2013-11-13  41.029999  42.869999  40.759998  42.599998  42.599998
...          ...        ...        ...        ...        ...        ...
2259  2022-10-28        NaN        NaN        NaN        NaN        NaN
2260  2022-10-31        NaN        NaN        NaN        NaN        NaN
2261  2022-11-01        NaN        NaN        NaN        NaN        NaN
2262  2022-11-02        NaN        NaN        NaN        NaN        NaN
2263  2022-11-03        NaN        NaN        NaN        NaN        NaN

          Volume
0     117701670.0
1      27925307.0
2      16113941.0
3       6316755.0
4       8688325.0
...           ...
2259          NaN
2260          NaN
2261          NaN
2262          NaN
2263          NaN

[2264 rows x 7 columns]
```

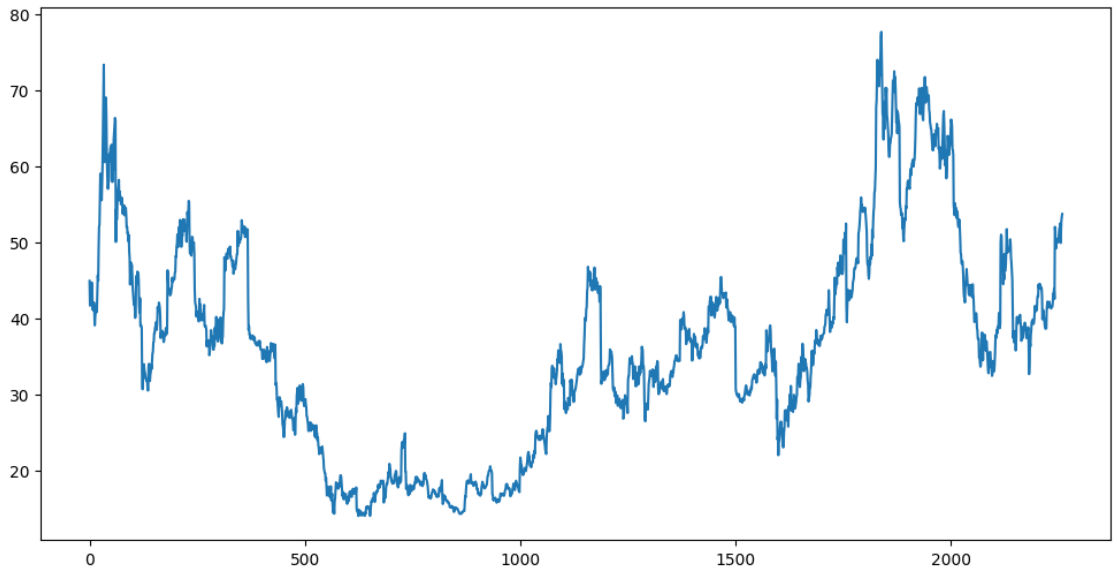
[7]:
```
df.head(10)
```

[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-11-07	45.099998	50.090000	44.000000	44.900002	44.900002	117701670.0
1	2013-11-08	45.930000	46.939999	40.685001	41.650002	41.650002	27925307.0
2	2013-11-11	40.500000	43.000000	39.400002	42.900002	42.900002	16113941.0
3	2013-11-12	43.660000	43.779999	41.830002	41.900002	41.900002	6316755.0
4	2013-11-13	41.029999	42.869999	40.759998	42.599998	42.599998	8688325.0
5	2013-11-14	42.340000	45.669998	42.240002	44.689999	44.689999	11099433.0
6	2013-11-15	45.250000	45.270000	43.430000	43.980000	43.980000	8010663.0
7	2013-11-18	43.500000	43.950001	40.849998	41.139999	41.139999	12810624.0
8	2013-11-19	41.389999	41.900002	40.000000	41.750000	41.750000	7436616.0
9	2013-11-20	41.400002	41.750000	40.509998	41.049999	41.049999	5767325.0

[8]:
```
plt.figure(figsize=(12,6))
plt.plot(df['Close'], label='Closing Price')
```

[8]: [<matplotlib.lines.Line2D at 0x7b82b7741180>]



Classification problem: Buy(+1) or sell(-1) the stock

[10]:
```
df['Open - Close'] = df['Open'] - df['Close']
df['High - Low'] = df['High'] - df['Low']
df = df.dropna()
print(df)
```

```
            Date       Open       High        Low      Close  Adj Close  \
0     2013-11-07  45.099998  50.090000  44.000000  44.900002  44.900002
1     2013-11-08  45.930000  46.939999  40.685001  41.650002  41.650002
2     2013-11-11  40.500000  43.000000  39.400002  42.900002  42.900002
3     2013-11-12  43.660000  43.779999  41.830002  41.900002  41.900002
4     2013-11-13  41.029999  42.869999  40.759998  42.599998  42.599998
…            …          …          …          …          …          …
2254  2022-10-21  50.000000  50.750000  49.549999  49.889999  49.889999
2255  2022-10-24  50.709999  51.860001  50.520000  51.520000  51.520000
2256  2022-10-25  52.415001  53.180000  52.200001  52.779999  52.779999
2257  2022-10-26  52.950001  53.500000  52.770000  53.349998  53.349998
2258  2022-10-27  53.910000  54.000000  53.700001  53.700001  53.700001

           Volume  Open - Close  High - Low
0     117701670.0      0.199996    6.090000
1      27925307.0      4.279998    6.254998
2      16113941.0     -2.400002    3.599998
3       6316755.0      1.759998    1.949997
4       8688325.0     -1.569999    2.110001
…             …             …           …
2254   51209029.0      0.110001    1.200001
2255   22987553.0     -0.810001    1.340001
2256   35077848.0     -0.364998    0.979999
2257   28064973.0     -0.399997    0.730000
2258  136345128.0      0.209999    0.299999

[2259 rows x 9 columns]
```

input Features to predict whether customer should buy or sell the stock

[11]:
```
a = df[['Open - Close','High - Low']]
a.head()
```

[11]:

	Open - Close	High - Low
0	0.199996	6.090000
1	4.279998	6.254998
2	-2.400002	3.599998
3	1.759998	1.949997
4	-1.569999	2.110001

intention is to store +1 for the buy signal and -1 for the sell signal. The target variable is “b” for classification task.

[12]:
```
b = np.where(df['Close'].shift(-1)>df['Close'],1,-1) #shift function is shift␣
 ↪the data up and down
print(b)
```

```
[-1  1 -1 …  1  1 -1]
```

[13]:
```
from sklearn.model_selection import train_test_split
a_train, a_test, b_train, b_test = train_test_split(a, b,test_size=0.25,␣
 ↪random_state=44)
```

Implementation of KNN Classifier.

[14]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

[15]:
```
#Using gridsearch to find the best parameter
prm = {'n_neighbors' :[2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
knn = neighbors.KNeighborsClassifier()
model = GridSearchCV(knn, prm, cv=5)

# fit the model
model.fit(a_train, b_train)

# Accuracy Score
accuracy_train = accuracy_score(b_train, model.predict(a_train))
accuracy_test = accuracy_score(b_test, model.predict(a_test))

print('Train_data Accuracy: %.2f' %accuracy_train)
print('Test_data Accuracy: %.2f' %accuracy_test)
```

```
Train_data Accuracy: 0.76
Test_data Accuracy: 0.48
```

[16]:
```
predictions_classification = model.predict(a_test)
```

[17]:
```
actual_predicted_data = pd.DataFrame({'Actual Class' :b_test, 'Predicted Class'␣
 ↪:predictions_classification})
```

[18]:
```
actual_predicted_data.head(10)
```

[18]:

	Actual Class	Predicted Class
0	-1	-1
1	1	1
2	-1	-1
3	-1	-1
4	1	1

```
5              1               -1
6             -1                1
7             -1                1
8              1                1
9              1               -1
```

[19]:
```
b = df['Close']
print(b)
```

```
0       44.900002
1       41.650002
2       42.900002
3       41.900002
4       42.599998
           ...
2254    49.889999
2255    51.520000
2256    52.779999
2257    53.349998
2258    53.700001
Name: Close, Length: 2259, dtype: float64
```

Implementation of KNN regression

[20]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn import neighbors
```

[24]:
```
a_train_reg, a_test_reg, b_train_reg, b_test_reg = train_test_split(a, b,␣
 ↪test_size=0.25, random_state= 44)

#using gridsearch to find the best parameter
prm = {'n_neighbors' :[2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
knn_reg = neighbors.KNeighborsRegressor()
model_reg = GridSearchCV(knn_reg, prm,cv=5)

#fit the model and make prediction
model_reg.fit(a_train_reg,b_train_reg)
prediction = model_reg.predict(a_test_reg)
```

[25]:
```
print(prediction)
```

```
[39.49066693 24.0700002  32.74533313 38.43666673 48.33733287 35.09666653
 29.8426666  16.8086666  39.54200027 49.7653332  35.10199987 15.9866668
 39.575333   51.894      43.42133367 19.33199993 43.05133327 36.97133327
 27.93333313 20.615333   27.93399993 38.80666693 34.87133313 33.64666667
 39.74400027 42.95733347 53.3246666  53.83999987 28.72933367 38.73333333
 39.14600033 41.01266667 34.01866693 39.2233332  38.2420002  51.97666713
 50.79866647 42.77000047 51.4966664  30.19800007 29.277333   16.27533313
```

40.80466713 47.27733293 40.23666687 22.2673334 37.51533353 42.05266727
39.65933313 38.69466633 50.22533307 45.85066653 27.126667   57.50933333
46.68800007 20.0673332 44.04400007 32.90333367 36.5653338 41.96933347
46.2880008 41.3913334 23.33666667 29.16466687 39.41999973 37.31933353
23.16333313 36.0566668 22.90333333 23.67266633 24.8780002 38.58533333
49.01266687 33.61933333 34.7839998 24.54466627 52.8479994 26.70066693
42.14866693 17.6853334 35.09666653 36.50200013 36.25133307 29.4
28.24266653 53.3679994 42.34200007 53.42666673 40.42800013 41.3520002
39.8813336 43.4826668 42.36866627 51.90799993 46.88866693 45.47933313
27.83333373 17.36400013 38.06799993 29.23199993 33.47466653 26.61200007
22.52933327 46.94000053 26.8719998 44.74066667 18.66933313 32.6633334
26.48866713 41.63533287 33.5406666 53.42666607 29.1886666 51.138
31.7966672 38.06466653 22.25533267 46.3219998 49.4899998 42.05599987
41.0706668 42.4893332 16.828   44.69666647 47.5513334 31.63266633
42.0219992 34.45266653 42.76333313 45.55599967 36.194   33.475333
41.2966666 32.85999993 55.02066647 37.77266673 36.80466667 17.88733347
24.40066693 49.2926664 52.0386666 41.29800033 19.19266667 37.9020004
33.49266667 22.2626668 18.52599993 21.60199993 39.64866693 41.95133327
45.7359998 33.17066713 37.35599967 23.1633336 29.20800013 18.43866673
32.92266713 25.988   30.4613334 26.24533347 23.01066667 29.59599987
25.30333327 31.94599993 17.2700002 31.69066667 48.3406668 42.9753332
37.0399998 39.0426668 26.35333333 42.37866693 36.63799967 41.97333347
35.5326668 54.43133367 22.06399987 44.69666653 48.9179998 36.79466687
48.39333307 36.27999993 23.6319998 43.63733353 41.27933353 20.55599993
18.22400007 16.67333313 28.1640004 35.0453336 48.42799967 23.7106662
30.42600033 37.65466653 27.8386664 37.578   22.06399987 49.03666627
20.8380002 30.8740006 36.91266667 36.72533333 30.85933347 50.978
34.169333   37.88333347 35.70333333 39.04533273 30.9760002 40.982
27.02466707 36.63933353 23.256   55.71399987 32.33533313 38.1333336
50.74133327 27.9026664 28.14666667 16.90799993 44.93266647 39.99533327
42.76666673 42.189334   19.198   49.11600013 36.03600013 31.25799967
49.28333387 38.57800027 39.93733353 49.60399993 31.97600007 31.50800013
25.30333327 25.12066653 46.48599993 36.24000007 37.8606664 40.19066647
36.92999993 53.2473328 31.57133333 34.3266668 30.2306672 48.94666707
29.59133327 37.05333307 45.18666693 43.91399973 50.35266693 34.7839998
49.9113332 31.1733334 38.19266647 38.2426666 29.54733367 25.20000027
45.3986668 38.00199987 30.19133333 37.52000033 41.62133327 40.03266693
41.43199967 40.18533367 39.71133333 33.52533347 35.874667   42.29666633
47.14933293 35.25800007 46.069333   29.6153334 31.60466687 38.53000013
17.87133327 48.40600027 56.57266613 38.0106668 34.3613332 35.74266687
37.49399967 25.22599993 22.34133333 22.04133333 34.72066707 39.933333
44.93266647 39.89866713 47.90666647 41.54533307 29.66733353 42.22266667
39.04799973 20.4366668 43.30600033 31.81666693 27.36599993 21.70133333
31.00666673 39.21466693 44.2359998 23.94733313 46.824   38.83400033
44.11600013 40.27466693 35.27266647 31.11666687 33.99599993 28.4360002
32.66066693 55.62799973 41.37999953 34.61399987 20.60599953 29.79066667
49.56999973 41.60199987 31.47799947 23.00466687 41.06066687 37.8373328
35.01666687 55.1   25.1253338 35.0353332 20.9646666 34.252

```
24.37466647 22.11466667 50.22533307 27.30266687 21.29466653 32.89133313
41.34533333 33.85466653 32.49933307 28.55066687 22.50200007 20.75200013
40.73199993 47.519334   16.24733333 35.62666667 41.7740002  38.29799973
41.39533347 45.1053336  36.9493328  50.6393334  36.27999993 35.2053336
21.5193334  37.36866687 40.02200007 37.63933307 56.6080002  16.78333327
44.5793334  42.10133333 41.39533287 38.58266667 49.98266687 48.15666693
33.16733327 43.86066627 39.17200027 22.54133307 35.9673334  36.51466627
49.06933387 30.2093336  18.22400007 24.2253334  31.5213334  34.1806668
30.76533313 38.12599993 48.57933353 15.87999993 34.05533353 41.72866667
35.05399973 31.1033336  17.3673334  33.85466653 53.1859996  42.29799913
43.34933393 16.902      31.794      26.91066713 35.24666687 29.19866633
33.3126664  39.51866653 31.26333333 41.0286666  45.98599973 47.42800053
32.7279996  26.48333353 46.6326662  41.7246664  32.74533313 31.89733273
43.272667   22.8320002  25.35533327 25.20200013 31.77733387 39.15066653
19.2279998  47.04799993 44.03799993 25.7900002  42.35666687 25.3566668
37.9579998  38.29199973 38.37533327 42.02666667 35.77000007 41.01933353
33.4680002  42.61999987 42.6866668  47.37933373 39.29533307 46.8546668
33.35999947 39.6479998  23.19199993 40.54333307 28.996      20.1186666
38.18       24.47533333 17.58333367 20.6593332  46.53600033 30.44533327
45.75733313 18.118      35.68866667 28.         26.8559998  40.27866687
34.19066667 52.13266673 41.30800087 36.28266687 15.91866673 37.1093332
40.4520006  16.6086666  46.7673332  46.86400067 41.36799993 33.06933347
41.4159998  48.17600007 22.25533353 35.10199987 26.57666667 17.54266667
41.8840002  53.5006666  40.24266707 44.2893334  41.69199993 44.14866707
17.89733353 50.19666607 23.33600013 37.65933327 34.92466647 24.8846666
52.53866713 31.2353336  45.731333   31.44866667 26.1060002  17.1299998
31.3653332  34.594      46.37133313 43.83200047 26.6033334  19.0513332
18.95999987 18.41666687 41.67400007 48.3806666  37.73399987 35.16266653
37.64999973 32.84199987 21.52533307 44.858667   25.11666687 29.57533313
36.838      53.97266673 48.23466633 15.53199993 37.51599987 42.7786666
35.9953332  17.44533347 42.1586668  28.3806666  44.35733333 27.89733347
37.5486668  39.5206666  24.42199993 22.82466673 42.30533333 40.25333327
37.080667   36.5446668  41.3000002  30.35933333 36.4960002  39.6859992
22.11533327 40.39400067 20.3620002  34.79333313 18.87199973 39.6720002
37.44533307 38.69400007 19.4953334  49.9220004  30.723333   23.37600007
37.52866627 32.14333327 39.6926668  33.2533334  52.09333293 47.50333327
22.3600002  28.55533333 37.08733327 40.9760002  48.71199987 28.78066707
33.3279998  53.2699996  20.10933313 53.02466633 17.7866666  22.62733313
19.13733327 40.95333367 39.39866673 37.27199993 53.82466613 36.89066647
33.7040004 ]
```

[27]:
```
# rmse(root mean square error)
rms = np.sqrt(np.mean(np.power((np.array(b_test)-np.array(prediction)),2)))
print(rms)
```

```
36.96522469487019
```

[28]:
```
valid = pd.DataFrame({'Actual Close' :b_test_reg, 'Predicted Close Valur' :
↪prediction})
```

[29]:
```
valid.head(10)
```

[29]:

	Actual Close	Predicted Close Valur
1350	33.020000	39.490667
559	16.799999	24.070000
1337	31.030001	32.745333
417	34.759998	38.436667
1862	64.239998	48.337333
1358	34.380001	35.096667
357	51.619999	29.842667
649	14.950000	16.808667
1732	43.840000	39.542000
66	56.470001	49.765333

[ ]:
```

```