Mining_Project

May 11, 2022

Name:

Abhay Kumar M.V.V.S

Mohith Krishna Behata

Email:

smgxy@mst.edu

mbm2b@mst.edu

Course:

CS 5402

Assignment:

Semester Project-Markdown

Tittle

Stock Analysis and Prediction

Date:

2022-05-11

GitHublink:

https://git-classes.mst.edu/smgxy/mining_project

https://git-classes.mst.edu/mbm2b/mining_project

[1]:
```
import pandas as pd
import numpy as np
import math
import datetime as dt
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪explained_variance_score, r2_score
import matplotlib.pyplot as plt
```

0.1 Concept description

An attempt to forecast the future value of a particular stock, a sector, the market, or the market as a whole is called a stock market prediction. Generally, these forecasts are based on fundamental

analysis of a company or economy, technical analysis of charts, or both.
The goal of the project is to conduct an analysis of the performance of stocks of a few big-name tech companies like Apple, Samsung, and Pixel. Once we have performed the analysis, we also want to see if we can use the obtained knowledge and apply it to predict the future trends for the stocks.

0.2 Data collection

Top 10 SmartPhone Companies Stock Price is the data we are using from 2016-2021, and all the data is verified and accurate. As it is the Top 10 SmartPhone Companies Stock Price from 2016-2021, it has data for 5 years, starting on 23 August and ending on 23 August 2021. Data is collected from Yahoo Finance. I appreciate their assistance. We will only use the Samsung, Apple, and Google pixel data from the available dataset.The data was downloaded from kaggle. The link from which the data was downloaded is,

https://www.kaggle.com/datasets/meetnagadia/stock-price-of-top-10-smartphone-company-20162021

0.3 Example Description

Date The Date is an attribute that describes the date on which the trading has taken place. As we are having 3 different data sets the range of values for Date attribute in each dataset has been describled below, The values for Date attribute ranges from, The range of values for Pixel dataset is 2016-08-23 to 2021-08-23. The range of values for Apple dataset is 2016-08-23 to 2021-08-20. The range of values for Pixel dataset is 2016-08-23 to 2021-08-23. Open
The Open is an attribute that describes the starting period of trading on a securities exchange or organized over-the-counter market,As we are having 3 different data sets the range of values for Open attribute in each dataset has been describled below, The values for Open attribute ranges from, The range of values for Pixel dataset is 1.92 to 6.95. The range of values for Apple dataset is 25.6625 to 150.229. The range of values for Pixel dataset is 29800 to 90300. High The High is an attribute that describes the highest price at which a stock is traded during a period, As we are having 3 different data sets the range of values for High attribute in each dataset has been describled below, The values for High attribute ranges from, The range of values for Pixel dataset is 2.05 to 7.05. The range of values for Apple dataset is 26.43 to 151.67. The range of values for Pixel dataset is 30120 to 96800. low The Low is an attribute that describes the Lowest price at which a stock is traded during a period, As we are having 3 different data sets the range of values for Low attribute in each dataset has been describled below, The values for Low attribute ranges from, The range of values for Pixel dataset is 1.85 to 6.66. The range of values for Apple dataset is 25.6325 to 149.089. The range of values for Pixel dataset is 29120 to 89500. Close The Close is an attribute that describes the end of a trading session in the financial markets, As we are having 3 different data sets the range of values for Close attribute in each dataset has been describled below, The values for Close attribute ranges from, The range of values for Pixel dataset is 1.92 to 6.93. The range of values for Apple dataset is 25.782 to 151.119. The range of values for Pixel dataset is 29300 to 91000. Adj Close The Adjusted Close is an attribute that describes the closing price after adjustments for all applicable splits and dividend distributions, As we are having 3 different data sets the range of values for Adj Close attribute in each dataset has been describled below, The values for Adj Close attribute ranges from, The range of values for Pixel dataset is 1.92 to 6.93. The range of values for Apple dataset is 24.179 to 151.119. The range of values for Pixel dataset is 25270.40625 to 90198.078125. Volume The Volume is an attribute that describes Volume in stocks

refers to the total number of shares traded, As we are having 3 different data sets the range of values for Volume attribute in each dataset has been described below, The values for Volume attribute ranges from, The range of values for Pixel dataset is 12400 to 11558400. The range of values for Apple dataset is 45448000 to 447940000 . The range of values for Pixel dataset is 0 to 90306177.

Attributes values level of measurement

- Date : Nominal.
- Open : Nominal/Ratio.
- High : Nominal/Ratio.
- Low : Nominal/Ratio.
- Close : Nominal/Ratio.
- AdjClose : Nominal/Ratio.
- Volume : Nominal/Ratio.

0.4 Data Importing and wrangling

[2]:
```
pix = pd.read_csv("pixel.csv")
app = pd.read_csv("apple.csv")
sam = pd.read_csv("samsung.csv")
```

[3]:
```
pix.head()
```

[3]:
```
         Date  Open  High   Low  Close  Adj Close  Volume
0  2016-08-23  3.17  3.29  3.11   3.23       3.23  351000
1  2016-08-24  3.21  3.22  2.90   2.97       2.97  233300
2  2016-08-25  2.93  3.01  2.75   2.97       2.97   84100
3  2016-08-26  2.96  2.99  2.82   2.87       2.87  122100
4  2016-08-29  2.89  2.96  2.81   2.89       2.89   50700
```

[4]:
```
app.head()
```

[4]:
```
         Date       Open       High        Low      Close  Adj Close  \
0  2016-08-23  27.147499  27.330000  27.132500  27.212500  25.520983
1  2016-08-24  27.142500  27.187500  26.920000  27.007500  25.328730
2  2016-08-25  26.847500  26.969999  26.670000  26.892500  25.220877
3  2016-08-26  26.852501  26.987499  26.577499  26.735001  25.073166
4  2016-08-29  26.655001  26.860001  26.572500  26.705000  25.045031

      Volume
0   85030800
1   94700400
2  100344800
3  111065200
4   99881200
```

[5]:
```
sam.head()
```

[5]:

```
         Date     Open     High      Low    Close     Adj Close      Volume
0  2016-08-23  33300.0  33880.0  33140.0  33740.0  29099.779297  11545700.0
1  2016-08-24  33600.0  33640.0  32720.0  33060.0  28513.298828  15938550.0
2  2016-08-25  32600.0  33180.0  32440.0  32780.0  28271.812500  14173800.0
3  2016-08-26  32120.0  32460.0  32060.0  32240.0  27806.070313  12058000.0
4  2016-08-29  32040.0  32800.0  31940.0  32800.0  28289.056641   8925750.0
```

[6]:

```
pix.tail()
```

[6]:

```
            Date  Open  High   Low  Close  Adj Close    Volume
1253  2021-08-16  3.58  3.60  3.41   3.43       3.43    450600
1254  2021-08-17  3.43  3.51  3.22   3.23       3.23    829000
1255  2021-08-18  3.24  3.66  3.22   3.24       3.24   1264500
1256  2021-08-19  3.20  3.74  3.20   3.56       3.56   3096200
1257  2021-08-20  3.69  4.48  3.69   4.45       4.45  11558400
```

[7]:

```
app.tail()
```

[7]:

```
            Date        Open        High         Low       Close   Adj Close  \
1253  2021-08-16  148.539993  151.190002  146.470001  151.119995  151.119995
1254  2021-08-17  150.229996  151.679993  149.089996  150.190002  150.190002
1255  2021-08-18  149.800003  150.720001  146.149994  146.360001  146.360001
1256  2021-08-19  145.029999  148.000000  144.500000  146.699997  146.699997
1257  2021-08-20  147.440002  148.500000  146.779999  148.190002  148.190002

         Volume
1253  103296000
1254   92229700
1255   86326000
1256   86960300
1257   59947400
```

[8]:

```
sam.tail()
```

[8]:

```
            Date     Open     High      Low    Close  Adj Close      Volume
1223  2021-08-17  74000.0  75100.0  74000.0  74200.0    74200.0  30944847.0
1224  2021-08-18  73900.0  74600.0  73100.0  73900.0    73900.0  29192631.0
1225  2021-08-19  73500.0  74400.0  73100.0  73100.0    73100.0  22166298.0
1226  2021-08-20  73500.0  73900.0  72500.0  72700.0    72700.0  22364803.0
1227  2021-08-23  73300.0  74000.0  73000.0  73300.0    73300.0  19271114.0
```

0.5 Exploratory Data Analysis

0.5.1 Values for each attribute

Determining the possible values or range of the values for each attribute

For all the data we have the minimum and maximum values would be provided as those values fall within the range.

[9]:
```python
dfs = [pix,app,sam]
for i in range(0,3):
    if i ==0:
        print('Pixel data ')
        for c in dfs[i].columns:
            print(f'{c}: min: {dfs[i][c].min()}, max: {dfs[i][c].max()}')

    elif i==1:
        print("------------------------------------------------------")
        print('Apple data')
        for c in dfs[i].columns:
            print(f'{c}: min: {dfs[i][c].min()}, max: {dfs[i][c].max()}')

    else:
        print("------------------------------------------------------")
        print('Samsung data')
        for c in dfs[i].columns:
            print(f'{c}: min: {dfs[i][c].min()}, max: {dfs[i][c].max()}')
```

```
Pixel data
Date: min: 2016-08-23, max: 2021-08-20
Open: min: 1.92, max: 6.95
High: min: 2.05, max: 7.05
Low: min: 1.85, max: 6.66
Close: min: 1.92, max: 6.93
Adj Close: min: 1.92, max: 6.93
Volume: min: 12400, max: 11558400
------------------------------------------------------
Apple data
Date: min: 2016-08-23, max: 2021-08-20
Open: min: 25.6625, max: 150.229996
High: min: 26.43, max: 151.679993
Low: min: 25.6325, max: 149.089996
Close: min: 25.782499, max: 151.119995
Adj Close: min: 24.179869, max: 151.119995
Volume: min: 45448000, max: 447940000
------------------------------------------------------
Samsung data
Date: min: 2016-08-23, max: 2021-08-23
Open: min: 29800.0, max: 90300.0
High: min: 30120.0, max: 96800.0
Low: min: 29120.0, max: 89500.0
Close: min: 29300.0, max: 91000.0
Adj Close: min: 25270.40625, max: 90198.078125
Volume: min: 0.0, max: 90306177.0
```

0.5.2 Finding the datatype

As all the datasets share some attribute we find the data type of attribute using one dataset

[10]:
```
for c in pix.columns:
    print(type(pix[c].iloc[0]))
```

```
<class 'str'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.int64'>
```

0.5.3 Data preprocessing - Converting the date format

[11]:
```
pix['Date'] = pd.to_datetime(pix.Date)
pix.sort_values(by='Date', inplace=True)

app['Date'] = pd.to_datetime(app.Date)
app.sort_values(by='Date', inplace=True)

sam['Date'] = pd.to_datetime(sam.Date)
sam.sort_values(by='Date', inplace=True)
```

0.5.4 Checking for missing values

[12]:
```
pix.isnull().sum()
app.isnull().sum()
sam.isnull().sum()
```

[12]:
```
Date         0
Open         5
High         5
Low          5
Close        5
Adj Close    5
Volume       5
dtype: int64
```

[13]:
```
pix.isna().sum()
app.isna().sum()
sam.isna().sum()
```

[13]:
```
Date         0
Open         5
High         5
```

```
Low          5
Close        5
Adj Close    5
Volume       5
dtype: int64
```

As the number of missing values are very few compared to the entire data, we just dropped the rows with missing values as the amount of information loss is very less.

0.5.5 Dropping the missing value rows

[14]:
```
pix.dropna(inplace=True)
pix.isna().any()
app.dropna(inplace=True)
app.isna().any()
sam.dropna(inplace=True)
sam.isna().any()
```

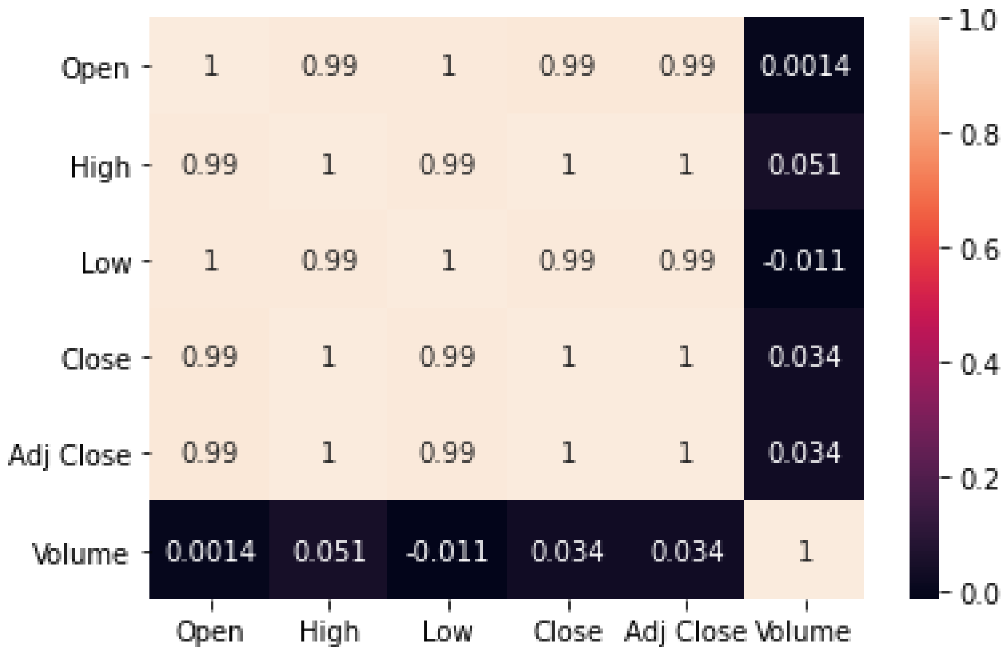
[14]:
```
Date          False
Open          False
High          False
Low           False
Close         False
Adj Close     False
Volume        False
dtype: bool
```

0.5.6 Correlation Analysis

Since we plan on performing a prediction of future closing prices, we need to understand the relationships among the attributes to further allow careful feature selection.

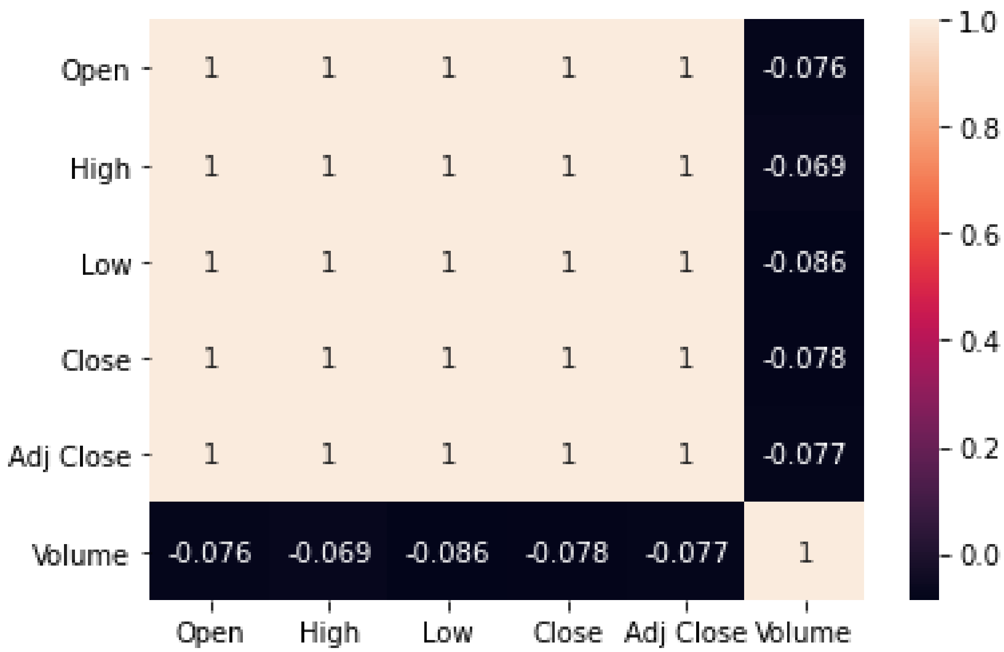
[15]:
```
import seaborn as sns
pixcr = pix[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
pcr = pixcr.corr()
sns.heatmap(pcr,annot=True)
```

[15]: <AxesSubplot:>



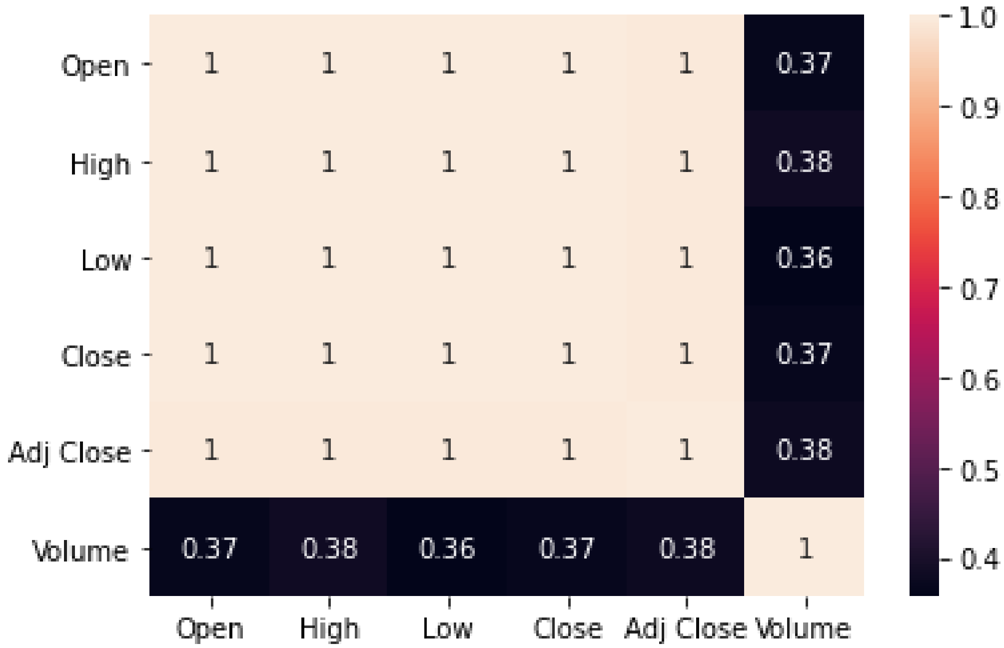
[16]:
```
appcr = app[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
acr = appcr.corr()
sns.heatmap(acr,annot=True)
```

[16]: <AxesSubplot:>



[17]:
```
samcr = sam[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
scr = samcr.corr()
sns.heatmap(scr,annot=True)
```

[17]: <AxesSubplot:>



From the heatmaps we can establish that all the features are strongly correlation with each other, expect the "Volume" feature.

0.6 Mining and Analytics

0.6.1 Monthwise comparision between Stock actual, open and close price

Extracting the month and year from the date

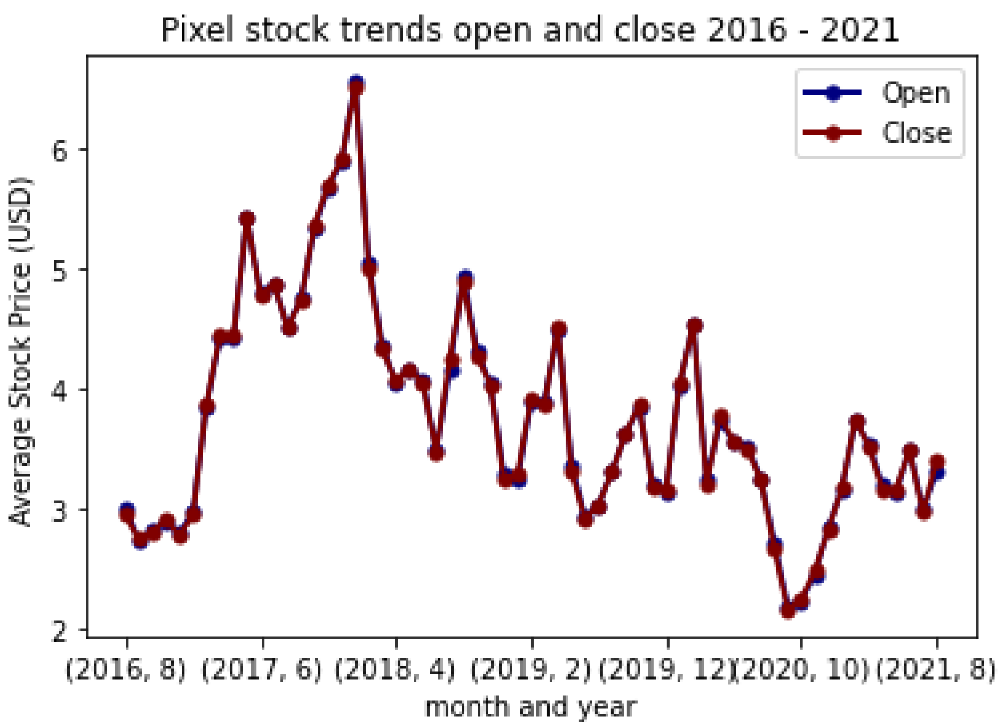
[18]:
```
pix['month'] = pd.DatetimeIndex(pix['Date']).month
pix['year'] = pd.DatetimeIndex(pix['Date']).year

app['month'] = pd.DatetimeIndex(app['Date']).month
app['year'] = pd.DatetimeIndex(app['Date']).year


sam['month'] = pd.DatetimeIndex(sam['Date']).month
sam['year'] = pd.DatetimeIndex(sam['Date']).year
```

Analysis Opening and closing prices per month We divide the columns by month and year, and calculate the average of opening and closing statistics for each stock in each month.

[19]:
```
pixmonth = pix.groupby(['year','month'])[['Open','Close']].mean()
appmonth = app.groupby(['year','month'])[['Open','Close']].mean()
sammonth = sam.groupby(['year','month'])[['Open','Close']].mean()
#print(pixmonth.head(),appmonth.head(),sammonth.head())
```

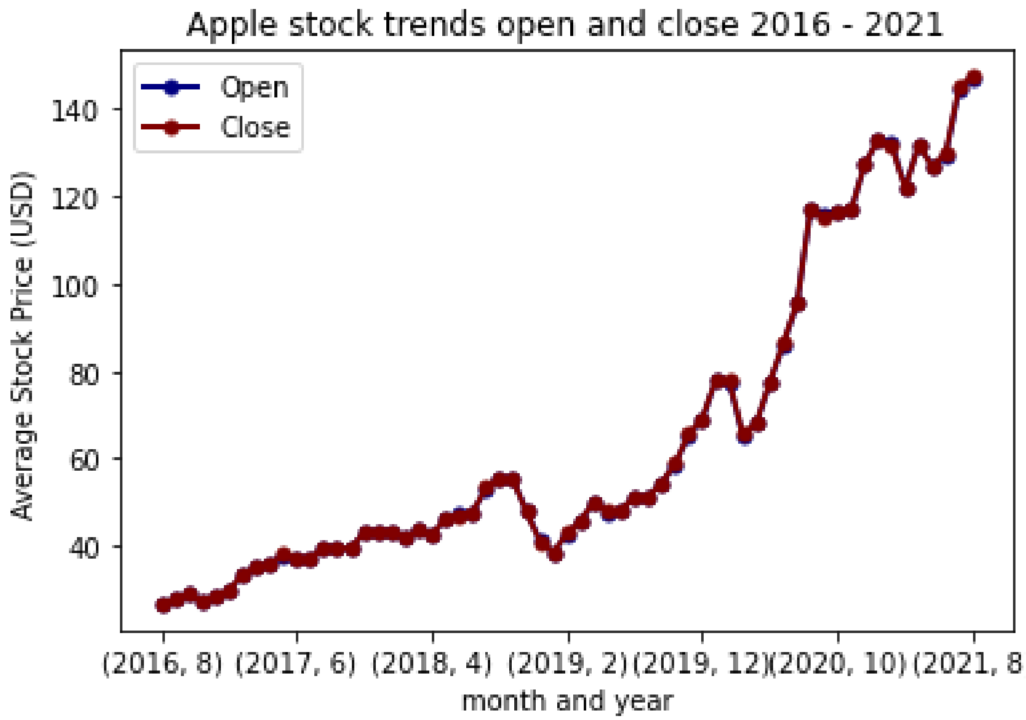
[20]:
```
ax = pixmonth.plot(lw=2, colormap='jet', marker='.', markersize=10,␣
↪title='Pixel stock trends open and close 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (USD)")
```

[20]: Text(0, 0.5, 'Average Stock Price (USD)')



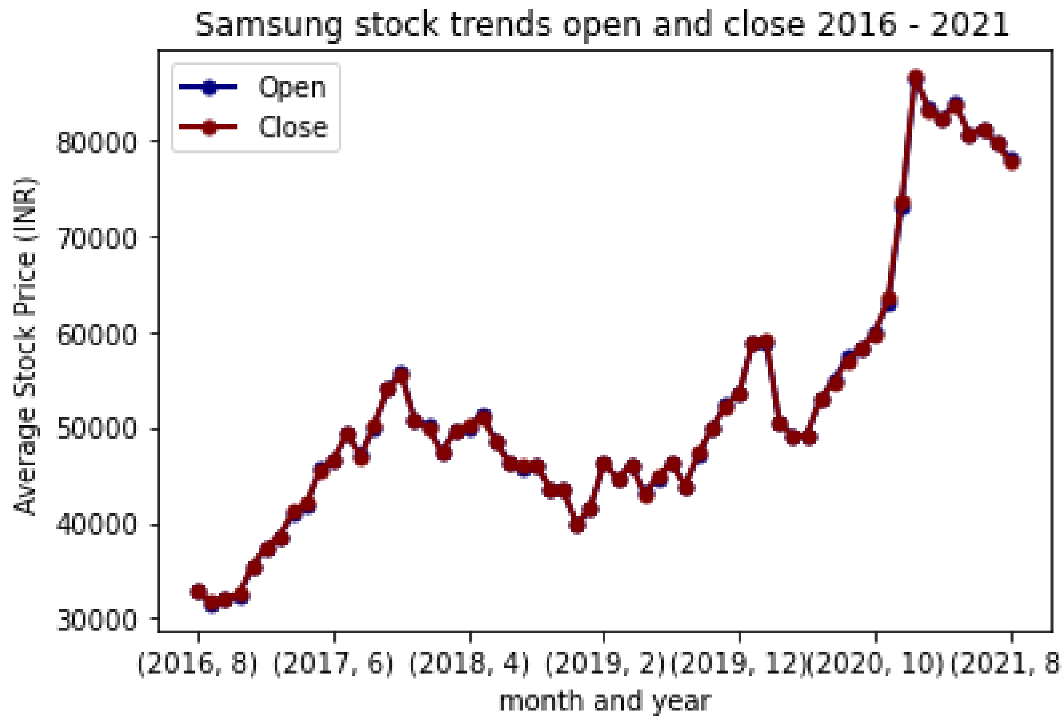
[21]:
```
ax = appmonth.plot(lw=2, colormap='jet', marker='.', markersize=10,␣
↪title='Apple stock trends open and close 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (USD)")
```

[21]: Text(0, 0.5, 'Average Stock Price (USD)')



[22]:
```
ax = sammonth.plot(lw=2, colormap='jet', marker='.', markersize=10,␣
 ↪title='Samsung stock trends open and close 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (INR)")
```

[22]: Text(0, 0.5, 'Average Stock Price (INR)')



From these initial plots, we can see that Apple and Samsung have fared well over the years, whereas, Pixel has an overall decreasing trend. Also we can see a drastic decrease in 2020 in all three companies. This can be easily explained with how covid-19 had an immense impact on all the sectors and all the companies had losses due to the same. There was less demand to buy stocks and more and more people were getting rid of stocks to offset expenses which they could not otherwise as a lot of people lost jobs.

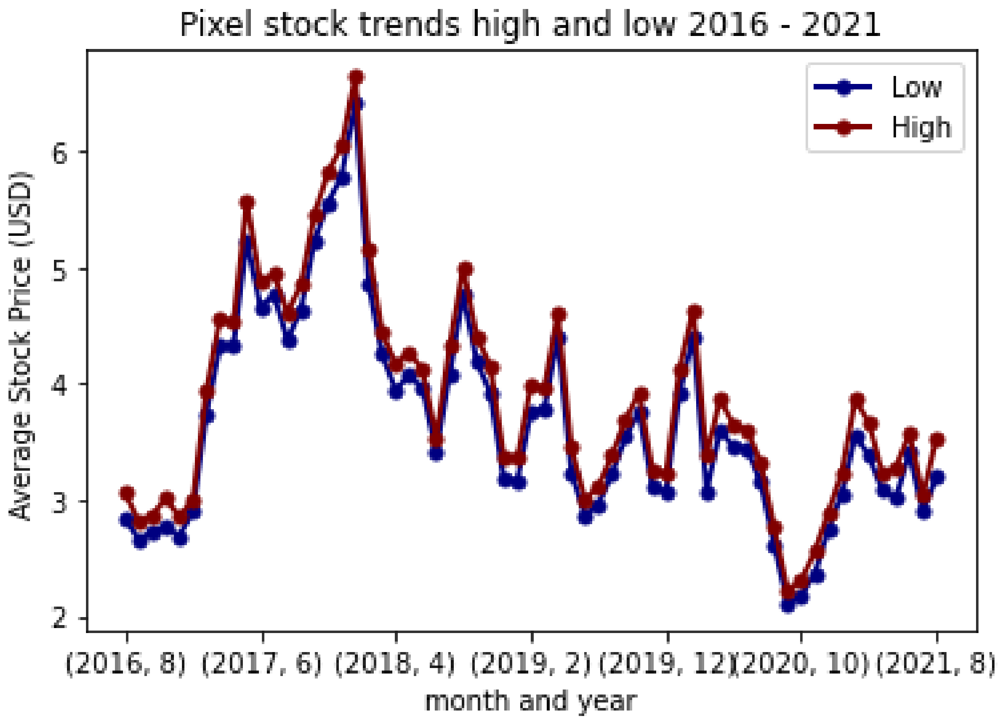
0.6.2 Analysis of High and low prices per month

Here we group the stock prices by year and month and analyse the trends for different companies

[23]:
```
pixmonth1 = pix.groupby(['year','month'])[['Low','High']].mean()
appmonth1 = app.groupby(['year','month'])[['Low','High']].mean()
sammonth1 = sam.groupby(['year','month'])[['Low','High']].mean()
#print(pixmonth1.head(),appmonth1.head(),sammonth1.head())
```

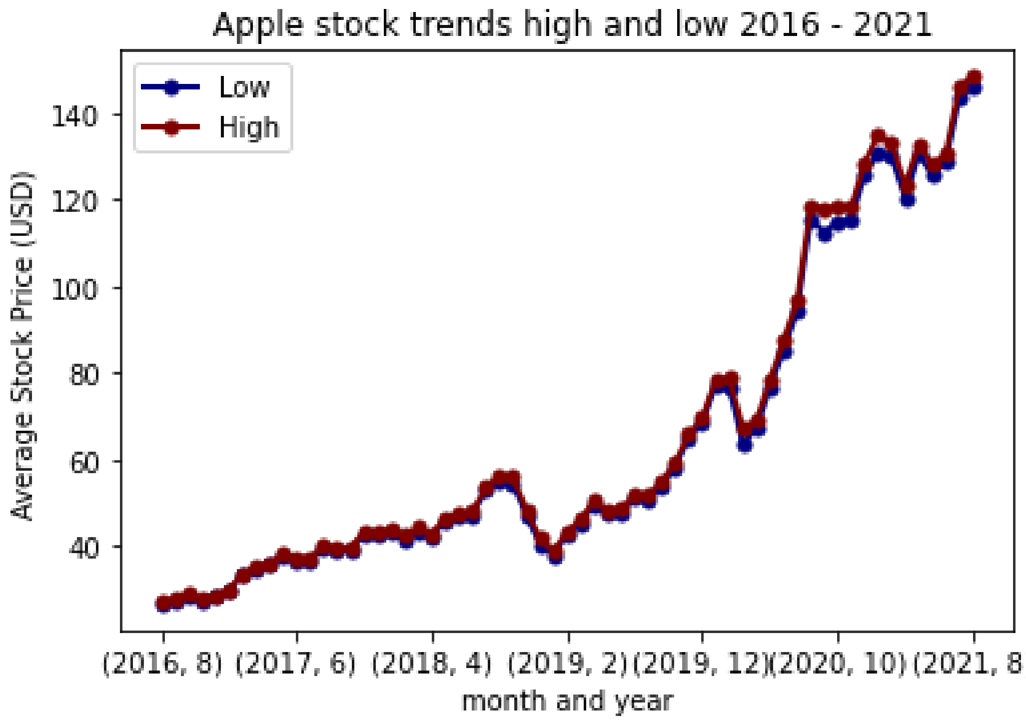
[24]:
```
ax = pixmonth1.plot(lw=2, colormap='jet', marker='.', markersize=10,␣
 ↪title='Pixel stock trends high and low 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (USD)")
```

[24]:
```
Text(0, 0.5, 'Average Stock Price (USD)')
```



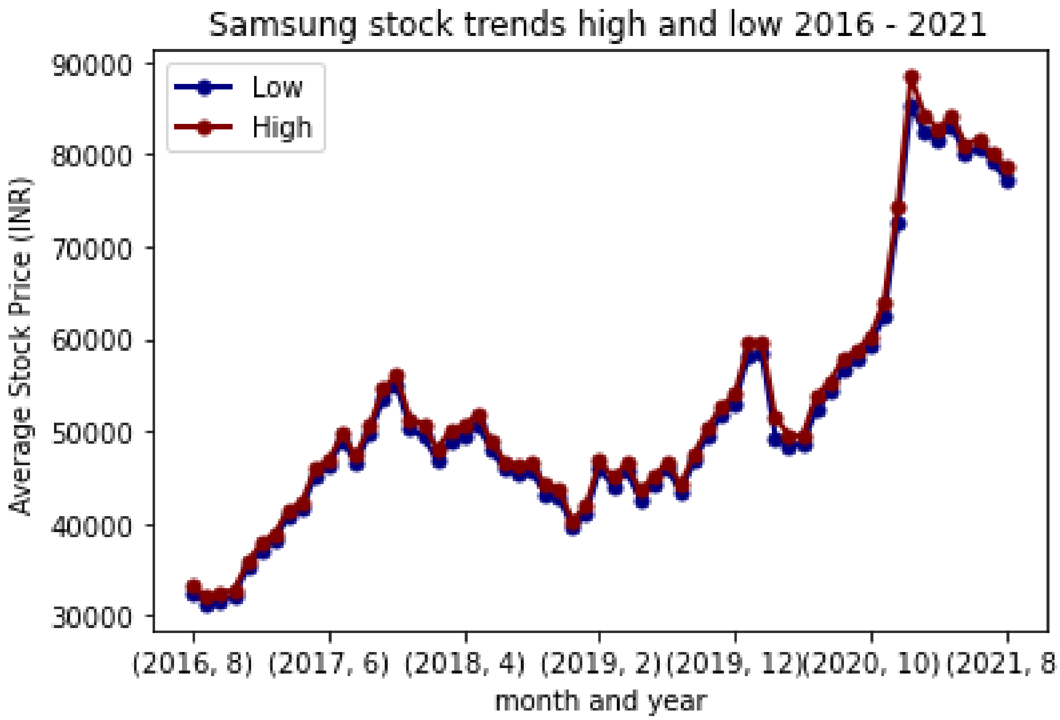
[25]:
```
ax = appmonth1.plot(lw=2, colormap='jet', marker='.', markersize=10, title='␣
 ↪Apple stock trends high and low 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (USD)")
```

[25]: Text(0, 0.5, 'Average Stock Price (USD)')



[26]:
```
ax = sammonth1.plot(lw=2, colormap='jet', marker='.', markersize=10,␣
 ↪title='Samsung stock trends high and low 2016 - 2021')
ax.set_xlabel("month and year")
ax.set_ylabel("Average Stock Price (INR)")
```

[26]: Text(0, 0.5, 'Average Stock Price (INR)')



From the above plots we can see that there is not much deviation between the high and low prices of the stocks at any point. Indicating that these stocks are non-volatile and relatively stable

0.6.3 Preparing the data

Here, we are going to do the following : 1. Separate the X and Y for each stock data, our Y is going to be the closing date 2. Normalizing the values using standardization and storing those scalers for retaining the original information 3. Getting the test train splits

splitting data into x and y

[27]:
```
pix_x = pixcr[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
pix_y = pixcr['Close']
pix_yr = np.array(pixcr['Close']).reshape(-1,1)
app_x = appcr[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
app_y = appcr['Close']
app_yr = np.array(appcr['Close']).reshape(-1,1)
sam_x = samcr[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
sam_y = samcr['Close']
sam_yr = np.array(samcr['Close']).reshape(-1,1)
```

Normalizing the data

[28]:
```
pix_nx=(pix_x-pix_x.mean())/pix_x.std()
pix_ny=np.array((pix_y-pix_y.mean())/pix_y.std()).reshape(-1,1)
```

```
app_nx=(app_x-app_x.mean())/app_x.std()
app_ny=np.array((app_y-app_y.mean())/app_y.std()).reshape(-1,1)
sam_nx=(sam_x-sam_x.mean())/sam_x.std()
sam_ny=np.array((sam_y-sam_y.mean())/sam_y.std()).reshape(-1,1)
```

Train and test splits As using test train split from sklearn will shuflle the data, which will lose the temporal correlation, we go with the simple split of first 80% data for train and the next 20% for test

[29]:
```
pixsplit = int(0.8*len(pix_nx))

pix_trainx, pix_trainy, pix_testx, pix_testy = pix_nx.iloc[0:pixsplit,:
 ↪],pix_ny[0:pixsplit,:],pix_nx.iloc[pixsplit:,:],pix_ny[pixsplit:,:]

print(pix_trainx.shape, pix_trainy.shape, pix_testx.shape,pix_testy.shape)
```

```
(1006, 5) (1006, 1) (252, 5) (252, 1)
```

[30]:
```
appsplit = int(0.8*len(app_nx))

app_trainx, app_trainy, app_testx, app_testy = app_nx.iloc[0:appsplit,:
 ↪],app_ny[0:appsplit,:],app_nx.iloc[appsplit:,:],app_ny[appsplit:,:]

print(app_trainx.shape, app_trainy.shape, app_testx.shape,app_testy.shape)
```

```
(1006, 5) (1006, 1) (252, 5) (252, 1)
```

[31]:
```
samsplit = int(0.8*len(sam_nx))

sam_trainx, sam_trainy, sam_testx, sam_testy = sam_nx.iloc[0:samsplit,:
 ↪],sam_ny[0:samsplit,:],sam_nx.iloc[samsplit:,:],sam_ny[samsplit:,:]

print(sam_trainx.shape, sam_trainy.shape, sam_testx.shape,sam_testy.shape)
```

```
(978, 5) (978, 1) (245, 5) (245, 1)
```

0.7 Evaluation

In this section we will train different models and evaluate them on the data

Linear regression models

[32]:
```
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix, accuracy_score
```

[33]:
```
pixreg = LinearRegression()
pixreg.fit(pix_trainx, pix_trainy)
```

```
pixreg_confidence = pixreg.score(pix_testx, pix_testy)
print("linear regression confidence: ", pixreg_confidence)
```

```
linear regression confidence:  1.0
```

[34]:
```
pixpredicted = pixreg.predict(pix_testx)
pixlr_rmse=np.sqrt(metrics.mean_squared_error(pix_testy, pixpredicted))
pixlr_acc = (pix_testy.mean())/(pixpredicted.mean()) *100
print(pixlr_acc)
```

```
100.00000000000007
```

[35]:
```
appreg = LinearRegression()
appreg.fit(app_trainx, app_trainy)
appreg_confidence = appreg.score(app_testx, app_testy)
print("linear regression confidence: ", appreg_confidence)
apppredicted =appreg.predict(app_testx)
applr_rmse=np.sqrt(metrics.mean_squared_error(app_testy, apppredicted))
applr_acc = (app_testy.mean())/(apppredicted.mean()) *100
print(applr_acc)
```

```
linear regression confidence:  0.9977731096391257
100.40624964309268
```

[36]:
```
samreg = LinearRegression()
samreg.fit(sam_trainx, sam_trainy)
samreg_confidence = samreg.score(sam_testx, sam_testy)
print("linear regression confidence: ", samreg_confidence)
sampredicted =samreg.predict(sam_testx)
samlr_rmse=np.sqrt(metrics.mean_squared_error(sam_testy, sampredicted))
samlr_acc = (sam_testy.mean())/(sampredicted.mean()) *100
print(samlr_acc)
```

```
linear regression confidence:  0.9975013775513606
99.01608187490807
```

As we can see that the accuracy is too high with normalized data, it might indiciate that the model might not learning trends, so moving forward we used original data instead of normalized data

splitting data into train and test using the original data

[37]:
```
pixsplit = int(0.8*len(pix_nx))

pix_trainx, pix_trainy, pix_testx, pix_testy = pix_nx.iloc[0:pixsplit,:
 ↪],pix_yr[0:pixsplit],pix_nx.iloc[pixsplit:,:],pix_yr[pixsplit:]

print(pix_trainx.shape, pix_trainy.shape, pix_testx.shape,pix_testy.shape)

appsplit = int(0.8*len(app_nx))
```

```
app_trainx, app_trainy, app_testx, app_testy = app_nx.iloc[0:appsplit,:
↪],app_yr[0:appsplit,:],app_nx.iloc[appsplit:,:],app_yr[appsplit:,:]

print(app_trainx.shape, app_trainy.shape, app_testx.shape,app_testy.shape)

samsplit = int(0.8*len(sam_nx))

sam_trainx, sam_trainy, sam_testx, sam_testy = sam_nx.iloc[0:samsplit,:
↪],sam_yr[0:samsplit,:],sam_nx.iloc[samsplit:,:],sam_yr[samsplit:,:]

print(sam_trainx.shape, sam_trainy.shape, sam_testx.shape,sam_testy.shape)
```

```
(1006, 5) (1006, 1) (252, 5) (252, 1)
(1006, 5) (1006, 1) (252, 5) (252, 1)
(978, 5) (978, 1) (245, 5) (245, 1)
```

Note on models built The given task was to build three models, where we had to use two models that were taught in class and one model which was not taught in class. But as were are performing prediction on continuous data and the majority of models taught in class were classification models with the permission of our instructor Mr. Perry Koob we have built 3 models where two were not taught in class and one was taught in class.

The models that were built are, 1. Multilinear regression model. 2. Ridge regression model. 3. Lasso regression model.

Multiple Linear Regression model

[38]:
```
pixreg = LinearRegression()
pixreg.fit(pix_trainx, pix_trainy)
pixreg_confidence = pixreg.score(pix_testx, pix_testy)
print("pixel: linear regression confidence: ", pixreg_confidence)
pixpredicted = pixreg.predict(pix_testx)
pixlr_rmse=np.sqrt(metrics.mean_squared_error(pix_testy, pixpredicted))
pixlrr2 = metrics.r2_score(pixpredicted,pix_testy)
pixlr_acc = (pixpredicted.mean()/pix_testy.mean())*100
print("pixel linear regression RMSE: ", pixlr_rmse)

print("pixel linear regression r2",pixlrr2)
```

```
pixel: linear regression confidence:  1.0
pixel linear regression RMSE:  5.533464473916797e-16
pixel linear regression r2 1.0
```

[39]:
```
appreg = LinearRegression()
appreg.fit(app_trainx, app_trainy)
appreg_confidence = appreg.score(app_testx, app_testy)
print("apple linear regression confidence: ", appreg_confidence)
```

```
apppredicted =appreg.predict(app_testx)
applr_rmse=np.sqrt(metrics.mean_squared_error(app_testy, apppredicted))
applrr2 = metrics.r2_score(apppredicted,app_testy)

applr_acc = (apppredicted.mean())/(app_testy.mean()) *100
print("apple linear regression RMSE: ", applr_rmse)
print("apple linear regression r2",applrr2)
```

```
apple linear regression confidence:  0.9977731096391258
apple linear regression RMSE:  0.48616032802597053
apple linear regression r2 0.9977669647462513
```

[40]:
```
samreg = LinearRegression()
samreg.fit(sam_trainx, sam_trainy)
samreg_confidence = samreg.score(sam_testx, sam_testy)
print("samsung linear regression confidence: ", samreg_confidence)
sampredicted =samreg.predict(sam_testx)
samlr_rmse=np.sqrt(metrics.mean_squared_error(sam_testy, sampredicted))
samlrr2 = metrics.r2_score(sampredicted,sam_testy)
#samlr_acc = (sam_testy.mean())/(sampredicted.mean()) *100
print("samsung linear regression RMSE: ", samlr_rmse)
print("samsung linear regression R2 ",samlrr2)
```

```
samsung linear regression confidence:  0.9975013775513606
samsung linear regression RMSE:  494.108834332737
samsung linear regression R2  0.9975432651688001
```

Ridge Regression Here, we consider the parameter alpha = 1.0

[41]:
```
from sklearn.linear_model import Ridge

pixrr = Ridge(alpha=1.0)
pixrr.fit(pix_trainx,pix_trainy)
pixrrpred=pixrr.predict(pix_testx)
pixrrsme = np.sqrt(metrics.mean_squared_error(pixrrpred,pix_testy))
pixrrr2 = metrics.r2_score(pixrrpred,pix_testy)
print("Pixel ridge regression RMSE: ",pixrrsme)
print("Pixel ridge regression R2: ",pixrrr2)
```

```
Pixel ridge regression RMSE:  0.009588359393736862
Pixel ridge regression R2:  0.9996856139825767
```

[42]:
```
apprr = Ridge(alpha=1.0)
apprr.fit(app_trainx, app_trainy)
apprrpred =apprr.predict(app_testx)
apprr_rmse=np.sqrt(metrics.mean_squared_error(app_testy, apprrpred))
apprrr2 = metrics.r2_score(apprrpred,app_testy)
```

```
print("apple ridge regression RMSE: ", apprr_rmse)
print("apple ridge regression r2",apprrr2)
```

```
apple ridge regression RMSE:  0.8591063177911388
apple ridge regression r2 0.9929202257097802
```

[43]:
```
samrr = Ridge(alpha=1.0)
samrr.fit(sam_trainx, sam_trainy)
samrrpred =samrr.predict(sam_testx)
samrr_rmse=np.sqrt(metrics.mean_squared_error(sam_testy, samrrpred))
samrrr2 = metrics.r2_score(samrrpred,sam_testy)
#samlr_acc = (sam_testy.mean())/(sampredicted.mean()) *100
print("samsung ridge regression RMSE: ", samrr_rmse)
print("samsung ridge regression R2 ",samrrr2)
```

```
samsung ridge regression RMSE:  542.7415648547875
samsung ridge regression R2  0.9970138115513776
```

Lasso model For the lasso model we have take alpha = 0.1 and maximum iterations as 10000, to ensure convergence to the optimal solution

[44]:
```
from sklearn.linear_model import Lasso
```

1. Pixel

[45]:
```
pixl = Lasso(alpha=0.1,max_iter=10000)
pixl.fit(pix_trainx,pix_trainy)
pixlpred = pixl.predict(pix_testx)
pixlpred.reshape(-1,1)
pixlrmse = np.sqrt(metrics.mean_squared_error(pixlpred,pix_testy))
pixlr2 = metrics.r2_score(pixlpred,pix_testy)
print("Pixel Lasso regression RMSE: ", pixlrmse)
print("Pixel Lasso regression r2: ", pixlr2)
```

```
Pixel Lasso regression RMSE:  0.12462733707147777
Pixel Lasso regression r2:  0.9330905105630616
```

2. Apple

[46]:
```
appl = Lasso(alpha=0.1,max_iter=10000)
appl.fit(app_trainx, app_trainy)
applpred =appl.predict(app_testx)
applpred = applpred.reshape(-1,1)
applrmse=np.sqrt(metrics.mean_squared_error(app_testy, applpred))
applr2 = metrics.r2_score(applpred,app_testy)

print("apple lasso regression RMSE: ", applrmse)
print("apple lasso regression r2",applr2)
```

```
apple lasso regression RMSE:  1.2221283161890288
apple lasso regression r2 0.9855248816931225
```

3. Samsung

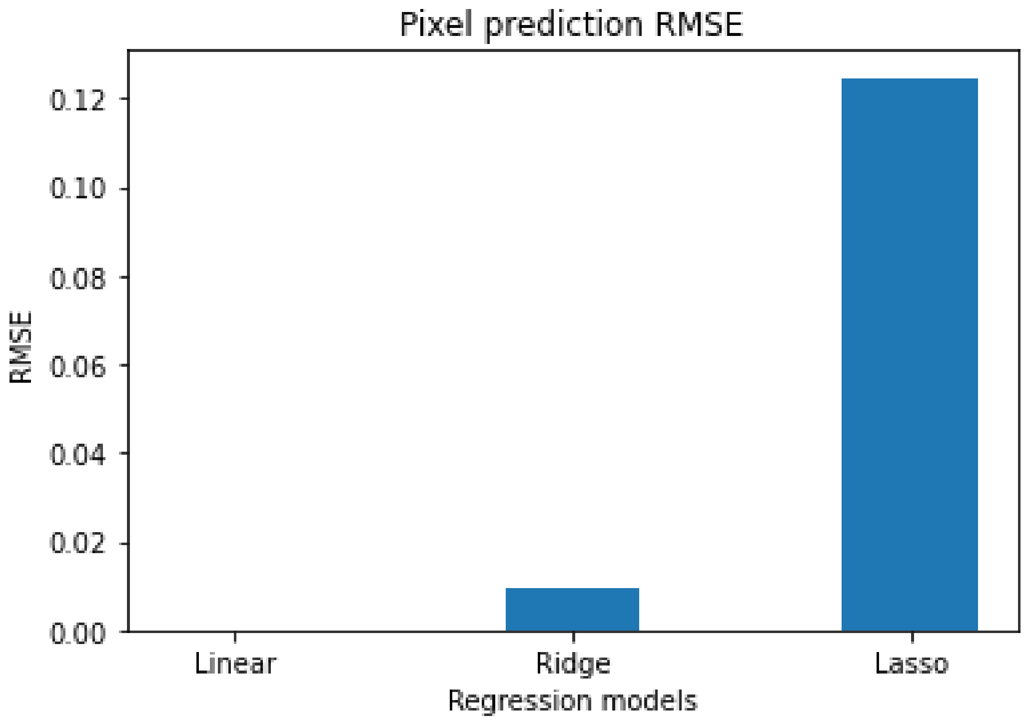
[47]:
```
saml = Lasso(alpha=0.1,max_iter=10000)
saml.fit(sam_trainx, sam_trainy)
samlpred =saml.predict(sam_testx)
samlpred = samlpred.reshape(-1,1)
samlrmse=np.sqrt(metrics.mean_squared_error(sam_testy, samlpred))
samlr2 = metrics.r2_score(samlpred,sam_testy)

print("samsung lasso regression RMSE: ", samlrmse)
print("samsung lasoo regression r2",samlr2)
```

```
samsung lasso regression RMSE:  490.94446885758396
samsung lasoo regression r2 0.9975744574723566
```

[48]:
```
pixel_rmses =[pixlr_rmse,pixrrsme,pixlrmse]
pixel_r2 =[pixlrr2,pixrrr2,pixlr2]
pixelcosts=[np.array(pix_testy),pixpredicted,pixrrpred,pixlpred]

apple_rmses =[applr_rmse,apprr_rmse,applrmse]
apple_r2 =[applrr2,apprrr2,applr2]
applecosts=[np.array(app_testy),apppredicted,apprrpred,applpred]

sam_rmses =[samlr_rmse,samrr_rmse,samlrmse]
sam_r2 =[samlrr2,samrrr2,samlr2]
samcosts=[np.array(sam_testy),sampredicted,samrrpred,samlpred]
```

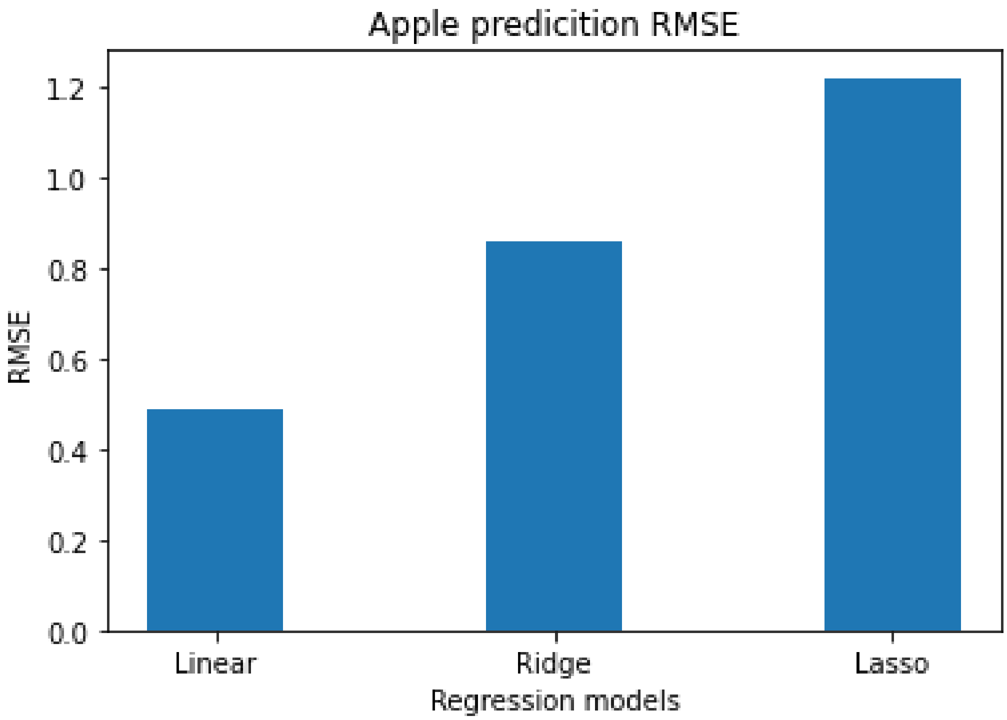
Pixel data prediction RMSE analysis

[49]:
```
plt.bar(['Linear','Ridge','Lasso'],pixel_rmses,width=0.4)

# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('RMSE')
plt.title('Pixel prediction RMSE')

plt.show()
```



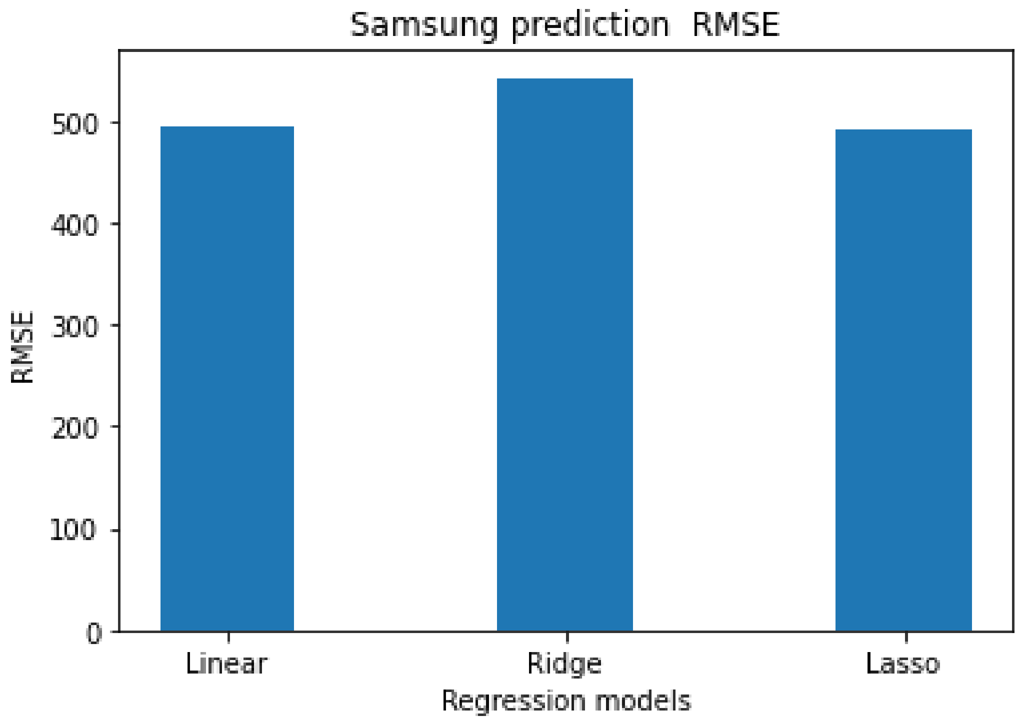
Apple data prediction RMSE analysis

[50]:
```
plt.bar(['Linear','Ridge','Lasso'],apple_rmses,width=0.4)



# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('RMSE')
plt.title('Apple predicition RMSE')

plt.show()
```



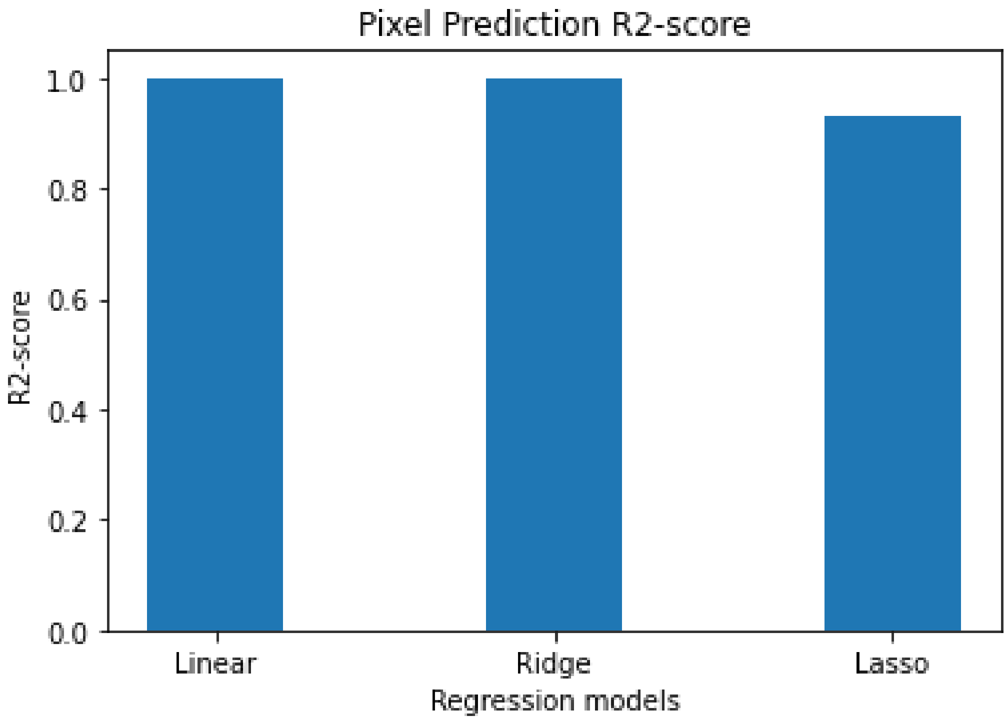
Samsung data prediction RMSE analysis

[51]:
```
plt.bar(['Linear','Ridge','Lasso'],sam_rmses,width=0.4)

# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('RMSE')
plt.title('Samsung prediction  RMSE')

plt.show()
```



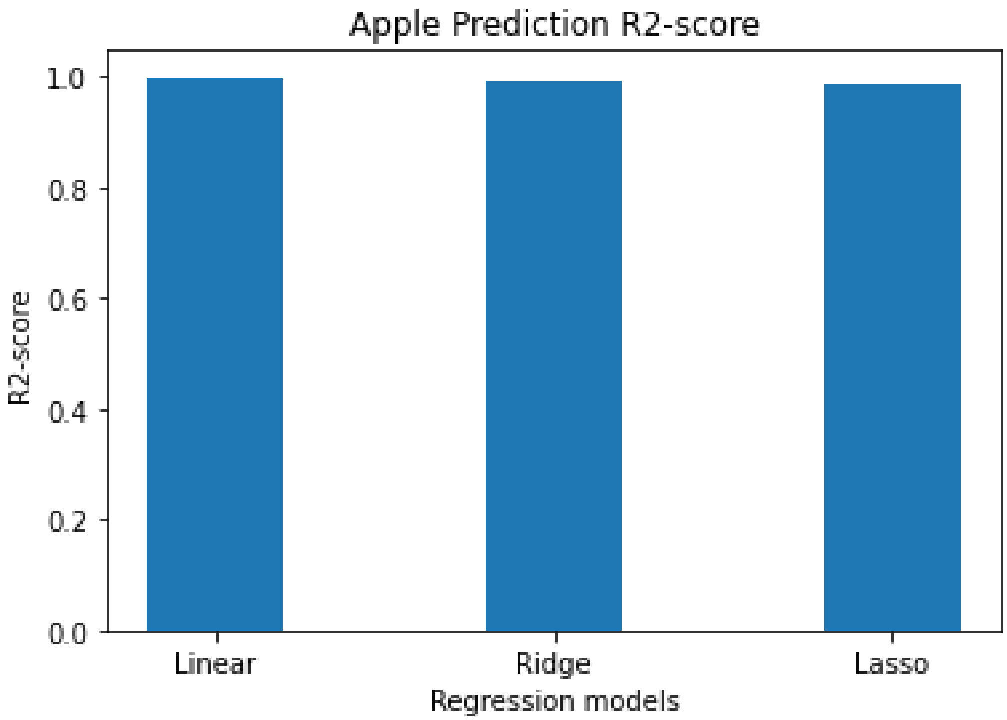
Pixel data prediction R2-score analysis

[52]:
```
plt.bar(['Linear','Ridge','Lasso'],pixel_r2,width=0.4)



# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('R2-score')
plt.title('Pixel Prediction R2-score')

plt.show()
```



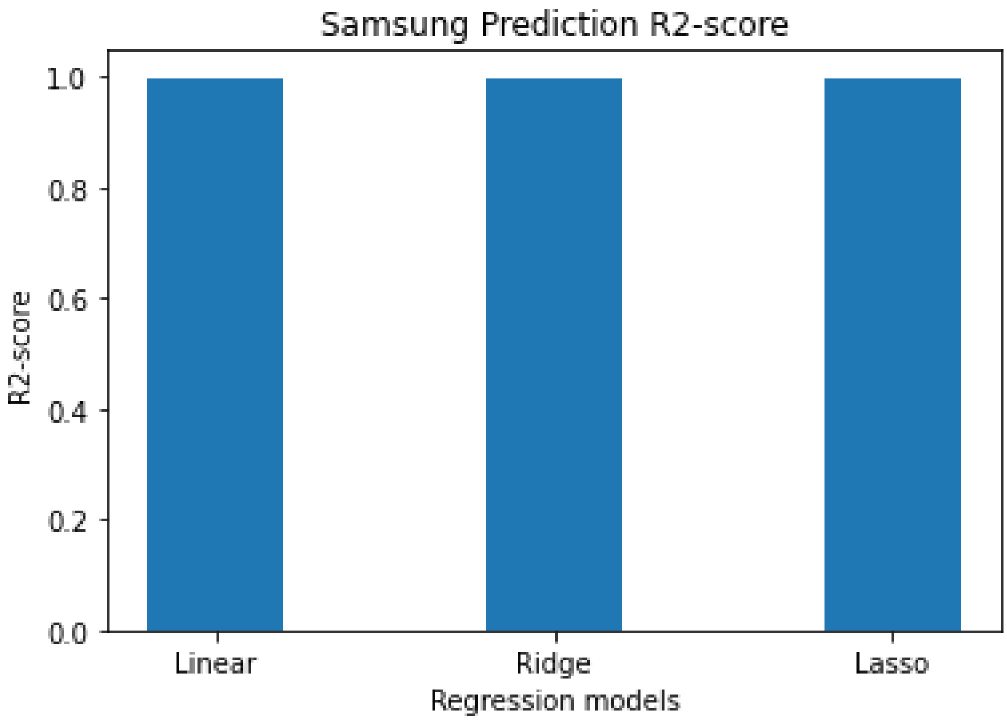
Apple data prediction R2-score analysis

[53]:
```
plt.bar(['Linear','Ridge','Lasso'],apple_r2,width=0.4)



# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('R2-score')
plt.title('Apple Prediction R2-score')

plt.show()
```



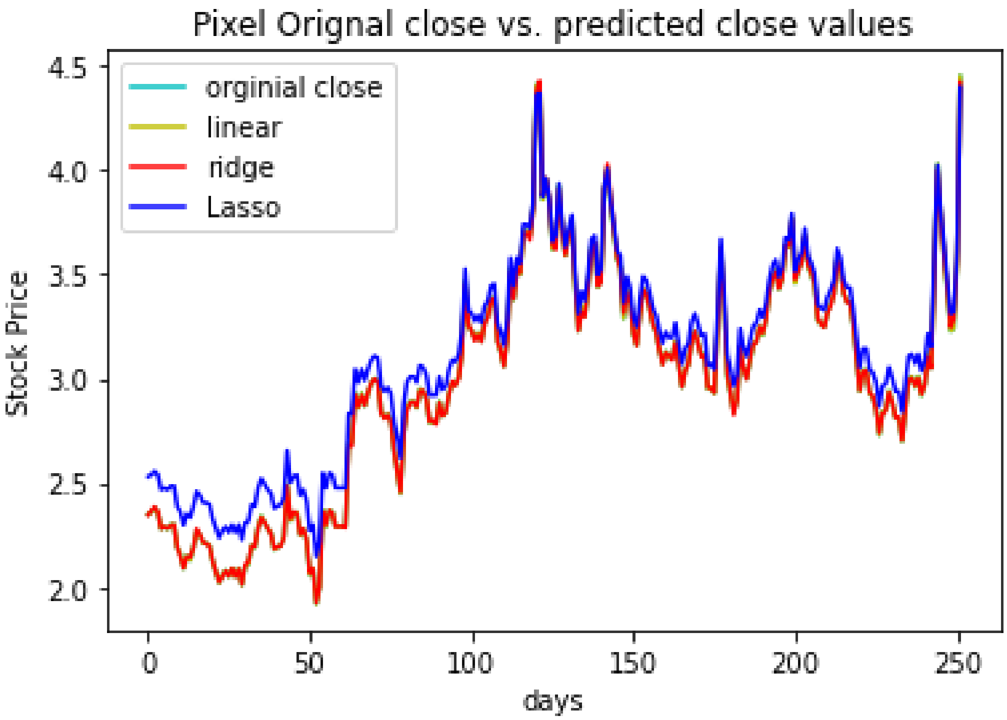
Samsung data prediction R2-score analysis

[54]:
```
plt.bar(['Linear','Ridge','Lasso'],sam_r2,width=0.4)

# The following commands add labels to our figure.
plt.xlabel('Regression models')
plt.ylabel('R2-score')
plt.title('Samsung Prediction R2-score')

plt.show()
```



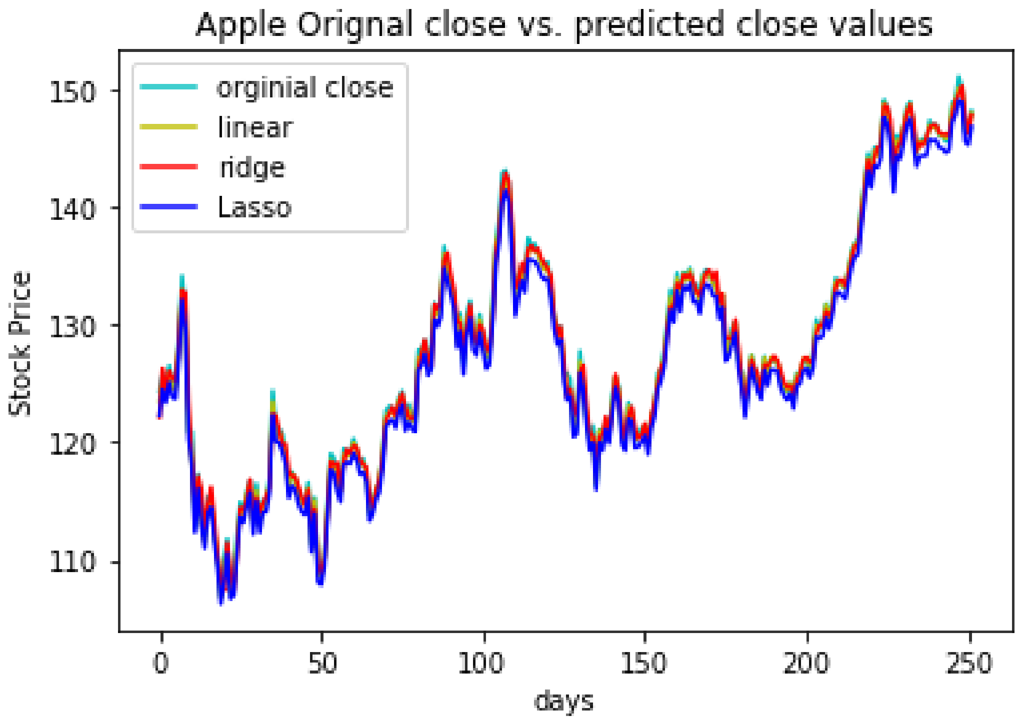
Pixel data original "close" vs. predicted "close"

[55]:
```
x = [i for i in range(0,len(pix_testy))]
y1 = pixelcosts[0]
y2 = pixelcosts[1]
y3 = pixelcosts[2]
y4 = pixelcosts[3]


# Plot a simple line chart
plt.plot(x, y1, 'c')
plt.plot(x, y2, 'y')
plt.plot(x, y3, 'r')
plt.plot(x, y4, 'b')
plt.ylabel("Stock Price")
plt.xlabel("days")

plt.legend(["orginial close", "linear","ridge","Lasso"], loc ="upper left")
plt.title("Pixel Orignal close vs. predicted close values")

plt.show()
```



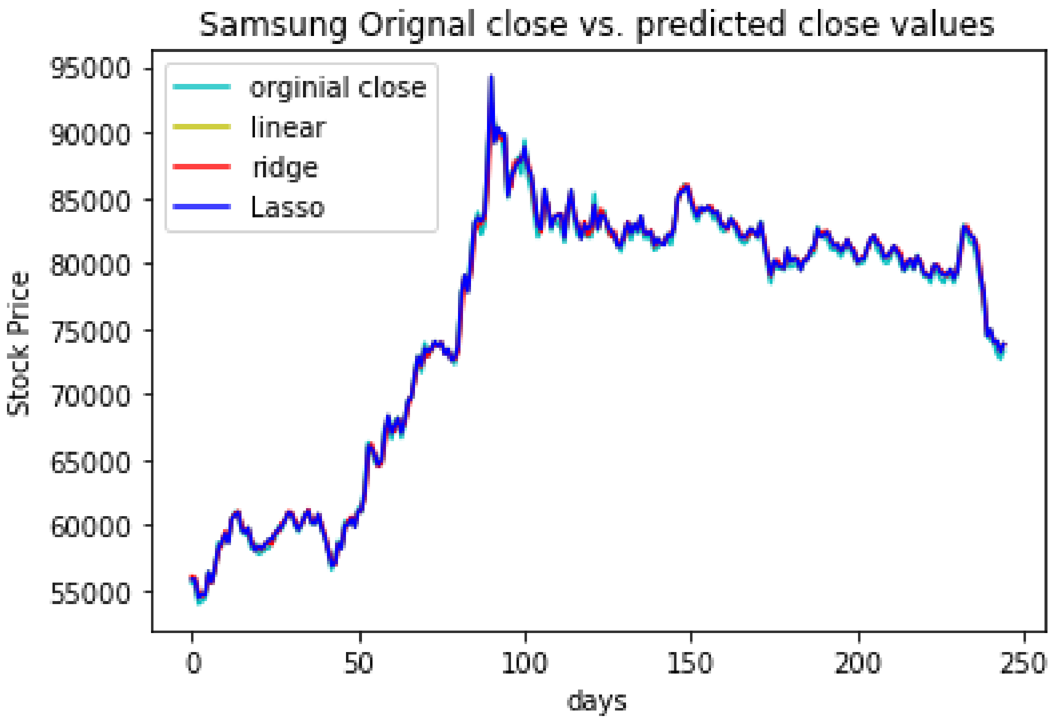
Apple data original "close" vs. predicted "close"

[56]:
```
x = [i for i in range(0,len(app_testy))]
y1 = applecosts[0]
y2 = applecosts[1]
y3 = applecosts[2]
y4 = applecosts[3]


# Plot a simple line chart
plt.plot(x, y1, 'c')
plt.plot(x, y2, 'y')
plt.plot(x, y3, 'r')
plt.plot(x, y4, 'b')
plt.ylabel("Stock Price")
plt.xlabel("days")
plt.legend(["orginial close", "linear","ridge","Lasso"], loc ="upper left")
plt.title("Apple Orignal close vs. predicted close values")

plt.show()
```



Samsung data original “close” vs. predicted “close”

[57]:
```
x = [i for i in range(0,len(sam_testy))]
y1 = samcosts[0]
y2 = samcosts[1]
y3 = samcosts[2]
y4 = samcosts[3]


# Plot a simple line chart
plt.plot(x, y1, 'c')
plt.plot(x, y2, 'y')
plt.plot(x, y3, 'r')
plt.plot(x, y4, 'b')
plt.ylabel("Stock Price")
plt.xlabel("days")

plt.legend(["orginial close", "linear","ridge","Lasso"], loc ="upper left")
plt.title("Samsung Orignal close vs. predicted close values")

plt.show()
```



0.8 Results

1. Since the data we are dealing with is continuous data, we cannot simply apply any regression models like logistic regression, as they perform classification at the end of the day. So we went with regression models that would allow us to work with continuous data : Linear, Ridge and Lasso.
2. For pixel and Apple we have low RMSE, indiciating that our predicted close values are almost identical to the original close values.
3. For pixel and apple we have a R2-score close to 1, indicating that the model is able to learn the trends and make accurate prediction.
4. For samsung, we see that the RMSE is high, this is due to the a lot if variation in the data itself and how the predicted values are on this data. But, the R2-scores for all models also indicate that these models are able to identify the trends
5. From our experiments we can conclude that linear regression has the highest performance among all the models, indicating that the underlying relationship among all the variable is a simple linear combination.

0.9 Refrences

iloc.Retrieved (2022, February 17) from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.htm

Justify the text in jupyter.Retrieved (2022, February 17) from https://stackoverflow.com/questions/35077507/how-to-right-align-and-justify-align-in-markdown

NumPy.Retrieved (2022, February 17) from https://numpy.org/doc/stable/index.html

Pandas Package. Retrieved (2022, February 17) from https://pandas.pydata.org/

Matplotlib.Retrieved (2022, February 17) from https://matplotlib.org/

Paper on Stock prediction using Multi-Linear Regression.Retrieved (2022, May 10) from https://ijesi.org/papers/Vol(7)i10/Version-2/D0710022933.pdf

Paper on Stock prediction using regresssion Analysis.Retrieved (2022, May 10) from https://www.ijser.org/researchpaper/Stock-Price-Prediction-Using-Regression-Analysis.pdf

Paper on Stock forecasting using lasso regresssion model.Retrieved (2022, May 10) from https://www.researchgate.net/publication/278660757_Stock_Market_Forecasting_Using_LASSO_Linear_Reg