CIS 4130
Mahir Hoque
mahir.hoque@baruchmail.cuny.edu

PROPOSAL

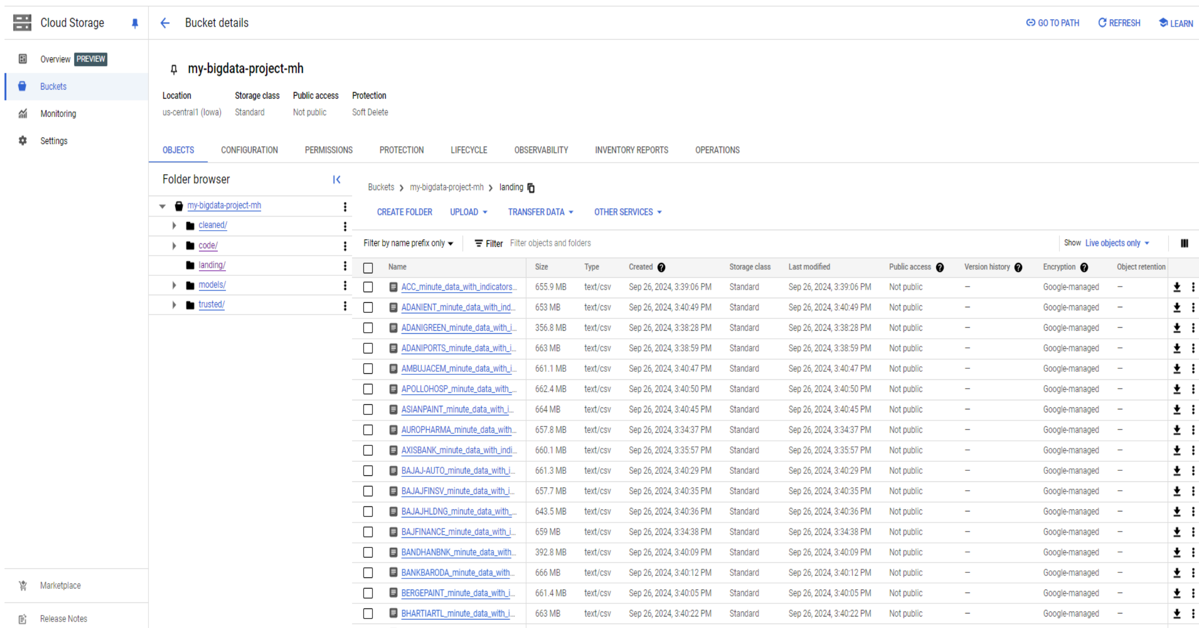
For the big data machine learning project, I intend to use a dataset about stocks from kaggle called "Stock Market Data - Nifty 100 Stocks (1 min) data". This is a dataset that includes 100 stocks data from the Nifty 100 index and two indices data on Nifty 50. For each stock there are 55 technical indicators that include: date, open, high, low, volume, and close price which serve as the data set attribute. There are 102 csv files in this dataset that equal 66.14 GB of data. The url is:

https://www.kaggle.com/datasets/debashis74017/stock-market-data-nifty-50-stocks-1-min-data/data.

I want to predict the stock's next day's closing price using the technical indicators. I will predict the "close" column which will be the Y variable and use the other columns as my X variable. I will use linear regression since I am predicting numerical values, closing stock prices, based on various data points. Also, closing stock price is a continuous outcome variable so linear regression will be a good fit for a predictive model.

DATA ACQUISITION

<u>Output:</u>



First, I modified a VM instance, I changed the machine type to n2d-standard-2 and increased the persistent disk size to 150 gb on Google Cloud. I then created a kaggle API token and downloaded the kaggle.json file from the Kaggle website.. I then opened an SSH common terminal and created a .kaggle directory there (Appendix A). I uploaded the kaggle json file, moved it to the directory, then secured it with the correct permissions. I installed all the necessary software packages such as zip and python3 to create a python environment. I activated the python environment and installed the kaggle library to handle the datasets I will load in. I used the Kaggle API to download the stock-market-data-nifty-50-stocks-1-min-data dataset which contained all the stock data. After downloading the dataset, I unzipped the file using unzip and checked to see if everything downloaded using ls -1. After ensuring everything is there, I had to authenticate the VM with Google Cloud Platform by using gcloud auth login then following the link and entering the authorization code. I made a GC storage bucket called my-big data-project-mh and manually created certain folders within it such as: landing, cleaned, trusted, code, and models. I uploaded the CSV files from the dataset into the landing file of the project bucket and used the recursive command to get all the files ended in CSV into the landing file. Finally, I confirmed all the files wee in the folder using gcloud storage ls -1.

EXPLORATORY DATA ANALYSIS AND DATA CLEANING

<u>Output:</u>

Bajaj Example

```
landing/BAJAJHLDNG_minute_data_with_indicators.csv Number of duplicate records: 0
landing/BAJAJHLDNG_minute_data_with_indicators.csv Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 666472 entries, 0 to 666471
Data columns (total 60 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   date          666472 non-null  object
 1   close         666472 non-null  float64
 2   high          666472 non-null  float64
 3   low           666472 non-null  float64
 4   open          666472 non-null  float64
 5   volume        666472 non-null  int64
 6   sma5          666472 non-null  float64
 7   sma10         666472 non-null  float64
 8   sma15         666472 non-null  float64
 9   sma20         666472 non-null  float64
 10  ema5          666472 non-null  float64
 11  ema10         666472 non-null  float64
 12  ema15         666472 non-null  float64
 13  ema20         666472 non-null  float64
 14  upperband     666472 non-null  float64
 15  middleband    666472 non-null  float64
 16  lowerband     666472 non-null  float64
 17  HT_TRENDLINE  666472 non-null  float64
 18  KAMA10        666472 non-null  float64
 19  KAMA20        666472 non-null  float64
 20  KAMA30        666472 non-null  float64
 21  SAR           666472 non-null  float64
 22  TRIMA5        666472 non-null  float64
 23  TRIMA10       666472 non-null  float64
 24  TRIMA20       666472 non-null  float64
 25  ADX5          666472 non-null  float64
 26  ADX10         666472 non-null  float64
 27  ADX20         666472 non-null  float64
 28  APO           666472 non-null  float64
 29  CCI5          666472 non-null  float64
 30  CCI10         666472 non-null  float64
 31  CCI15         666472 non-null  float64
 32  macd510       666472 non-null  float64
 33  macd520       666472 non-null  float64
 34  macd1020      666472 non-null  float64
 35  macd1520      666472 non-null  float64
 36  macd1226      666472 non-null  float64
 37  MFI           666472 non-null  float64
 38  MOM10         666472 non-null  float64
 39  MOM15         666472 non-null  float64
 40  MOM20         666472 non-null  float64
 41  ROC5          666472 non-null  float64
```

Bajaj Holding Data type

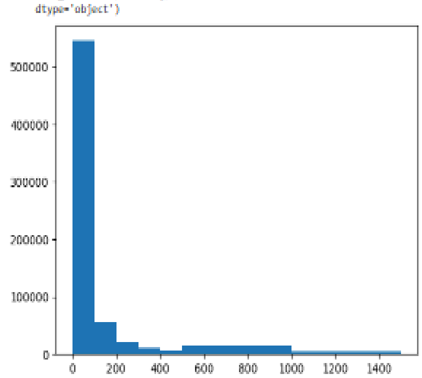
```
File landing/BAJAJHLDNG_minute_data_with_indicators.csv with size 674758787 bytes
landing/BAJAJHLDNG_minute_data_with_indicators.csv Number of records:
date            666472
close           666472
high            666472
low             666472
open            666472
volume          666472
sma5            666472
sma10           666472
sma15           666472
sma20           666472
ema5            666472
ema10           666472
ema15           666472
ema20           666472
upperband       666472
middleband      666472
lowerband       666472
HT_TRENDLINE    666472
KAMA10          666472
KAMA20          666472
KAMA30          666472
SAR             666472
TRIMA5          666472
TRIMA10         666472
TRIMA20         666472
ADX5            666472
ADX10           666472
ADX20           666472
APO             666472
CCI5            666472
CCI10           666472
CCI15           666472
macd510         666472
macd520         666472
macd1020        666472
macd1520        666472
macd1226        666472
MFI             666472
MOM10           666472
MOM15           666472
MOM20           666472
ROC5            666472
ROC10           666472
ROC20           666472
PPO             666472
RSI14           666472
RSI8            666472
```

Bajaj Record Count

```
landing/BAJAJHLDNG_minute_data_with_indicators.csv Describe
             close           high            low           open  \
count  666472.000000  666472.000000  666472.000000  666472.000000
mean     2940.054690    2941.451268    2938.622887    2940.067841
std      1224.481442    1225.335903    1223.556650    1224.490046
min         0.000000       0.000000       0.000000       0.000000
25%      1940.000000    1941.250000    1939.550000    1940.000000
50%      2840.000000    2840.150000    2838.950000    2840.000000
75%      3507.100000    3509.000000    3505.150000    3507.300000
max      7309.600000    7378.400000    7294.000000    7315.000000

            volume          sma5          sma10          sma15  \
count  666472.000000  6.664720e+05  666472.000000  666472.000000
mean      119.795291  2.940039e+03    2940.020529    2940.001592
std      2164.293285  1.224449e+03    1224.421523    1224.397571
min         0.000000 -1.000444e-11     389.725000     692.930000
25%         0.000000  1.940067e+03    1939.901250    1939.793333
50%        12.000000  2.839840e+03    2839.680000    2839.506667
75%        61.000000  3.507210e+03    3507.171250    3507.155000
max    711488.000000  7.299130e+03    7296.370000    7291.163333

             sma20           ema5  ...         fastd        fastksr  \
count  666472.000000  666472.000000  ...  6.664720e+05  666472.000000
mean     2939.982657    2940.039501  ...  4.590825e+01      48.469370
std      1224.375353    1224.444719  ...  3.298942e+01      44.329689
min       844.537500      76.046087  ... -9.598248e-12       0.000000
25%      1939.671875    1939.995552  ...  1.636998e+01       0.000000
50%      2839.461250    2839.703216  ...  4.445714e+01      45.572823
75%      3507.121875    3507.229280  ...  7.205292e+01     100.000000
max      7281.765000    7295.957784  ...  1.000000e+02     100.000000

            fastdsr         ULTOSC          WILLR           ATR  \
count  6.664720e+05  666472.000000  666472.000000  6.664720e+05
mean   4.846932e+01      48.526604     -50.096070  3.044422e+00
std    3.379921e+01      21.082986      35.514484  3.897657e+00
min   -1.929938e-12       0.000000    -100.000000  5.693988e-07
25%    1.977844e+01      35.085636     -83.399209  9.593643e-01
50%    4.754063e+01      49.100084     -50.000000  1.986247e+00
75%    7.445785e+01      62.399581     -16.666667  3.829782e+00
max    1.000000e+02     100.000000      -0.000000  1.870054e+02

              Trange       TYPPRICE    HT_DCPERIOD           BETA
count  666472.000000  666472.000000  666472.000000  666472.000000
mean        3.044574    2940.042948      19.927951       0.138484
std         9.412656    1224.455916       4.802888       1.882702
min         0.000000       0.000000       7.108812    -179.555103
25%         0.000000    1940.133333      16.485117      -0.124140
50%         1.300000    2839.950000      19.252032       0.000000
75%         4.050000    3507.066667      22.644207       0.415753
max      1354.000000    7301.583333      45.515438     123.868920

[8 rows x 59 columns]
landing/BAJAJHLDNG_minute_data_with_indicators.csv Columns with null values
[]
landing/BAJAJHLDNG_minute_data_with_indicators.csv Number of Rows with null values: 0
landing/BAJAJHLDNG_minute_data_with_indicators.csv Integer data type columns: Index(['volume'], dtype='object')
landing/BAJAJHLDNG_minute_data_with_indicators.csv Float data type columns: Index(['close', 'high', 'low', 'open', 'sma5', 'sma10', 'sma15', 'sma20',
       'ema5', 'ema10', 'ema15', 'ema20', 'upperband', 'middleband',
```

Bajaj Statistics

```
[8 rows x 59 columns]
landing/BAJAJHLDNG_minute_data_with_indicators.csv Columns with null values
[]
landing/BAJAJHLDNG_minute_data_with_indicators.csv Number of Rows with null values: 0
landing/BAJAJHLDNG_minute_data_with_indicators.csv Integer data type columns: Index(['volume'], dtype='object')
landing/BAJAJHLDNG_minute_data_with_indicators.csv Float data type columns: Index(['close', 'high', 'low', 'open', 'sma5', 'sma10', 'sma15', 'sma20',
       'ema5', 'ema10', 'ema15', 'ema20', 'upperband', 'middleband',
       'lowerband', 'HT_TRENDLINE', 'KAMA10', 'KAMA20', 'KAMA30', 'SAR',
       'TRIMA5', 'TRIMA10', 'TRIMA20', 'ADX5', 'ADX10', 'ADX20', 'APO', 'CCI5',
       'CCI10', 'CCI15', 'macd510', 'macd520', 'macd1020', 'macd1520',
       'macd1226', 'MFI', 'MOM10', 'MOM15', 'MOM20', 'ROC5', 'ROC10', 'ROC20',
       'PPO', 'RSI14', 'RSI8', 'slowk', 'slowd', 'fastk', 'fastd', 'fastksr',
       'fastdsr', 'ULTOSC', 'WILLR', 'ATR', 'Trange', 'TYPPRICE',
       'HT_DCPERIOD', 'BETA'],
      dtype='object')
```



Bajaj Null Values

and Histogram

Yes Bank Example

```
landing/YESBANK_minute_data_with_indicators.csv Number of duplicate records: 0
landing/YESBANK_minute_data_with_indicators.csv Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 671013 entries, 0 to 671012
Data columns (total 60 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   date          671013 non-null  object
 1   close         671013 non-null  float64
 2   high          671013 non-null  float64
 3   low           671013 non-null  float64
 4   open          671013 non-null  float64
 5   volume        671013 non-null  int64
 6   sma5          671013 non-null  float64
 7   sma10         671013 non-null  float64
 8   sma15         671013 non-null  float64
 9   sma20         671013 non-null  float64
 10  ema5          671013 non-null  float64
 11  ema10         671013 non-null  float64
 12  ema15         671013 non-null  float64
 13  ema20         671013 non-null  float64
 14  upperband     671013 non-null  float64
 15  middleband    671013 non-null  float64
 16  lowerband     671013 non-null  float64
 17  HT_TRENDLINE  671013 non-null  float64
 18  KAMA10        671013 non-null  float64
 19  KAMA20        671013 non-null  float64
 20  KAMA30        671013 non-null  float64
 21  SAR           671013 non-null  float64
 22  TRIMA5        671013 non-null  float64
 23  TRIMA10       671013 non-null  float64
 24  TRIMA20       671013 non-null  float64
 25  ADX5          671013 non-null  float64
 26  ADX10         671013 non-null  float64
 27  ADX20         671013 non-null  float64
 28  APO           671013 non-null  float64
 29  CCI5          671013 non-null  float64
 30  CCI10         671013 non-null  float64
 31  CCI15         671013 non-null  float64
 32  macd510       671013 non-null  float64
 33  macd520       671013 non-null  float64
 34  macd1020      671013 non-null  float64
 35  macd1520      671013 non-null  float64
 36  macd1226      671013 non-null  float64
 37  MFI           671013 non-null  float64
 38  MOM10         671013 non-null  float64
 39  MOM15         671013 non-null  float64
 40  MOM20         671013 non-null  float64
 41  ROC5          671013 non-null  float64
 42  ROC10         671013 non-null  float64
 43  ROC20         671013 non-null  float64
 44  PPO           671013 non-null  float64
```

Yes Bank Data Type

```
File landing/YESBANK_minute_data_with_indicators.csv with size 695765993 bytes
landing/YESBANK_minute_data_with_indicators.csv Number of records:
date            671013
close           671013
high            671013
low             671013
open            671013
volume          671013
sma5            671013
sma10           671013
sma15           671013
sma20           671013
ema5            671013
ema10           671013
ema15           671013
ema20           671013
upperband       671013
middleband      671013
lowerband       671013
HT_TRENDLINE    671013
KAMA10          671013
KAMA20          671013
KAMA30          671013
SAR             671013
TRIMA5          671013
TRIMA10         671013
TRIMA20         671013
ADX5            671013
ADX10           671013
ADX20           671013
APO             671013
CCI5            671013
CCI10           671013
CCI15           671013
macd510         671013
macd520         671013
macd1020        671013
macd1520        671013
macd1226        671013
MFI             671013
MOM10           671013
MOM15           671013
MOM20           671013
ROC5            671013
ROC10           671013
ROC20           671013
PPO             671013
RSI14           671013
RSI8            671013
slowk           671013
slowd           671013
fastk           671013
```

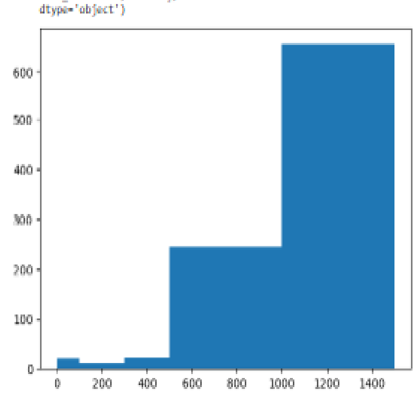
Yes Bank Record Count

```
landing/YESBANK_minute_data_with_indicators.csv Describe
               close           high            low           open  \
count  671013.000000  671013.000000  671013.000000  671013.000000
mean      148.327325     148.429565     148.219338     148.328312
std       122.664404     122.723782     122.607838     122.664864
min         6.000000       8.500000       5.750000       6.000000
25%        16.250000      16.300000      16.250000      16.250000
50%       151.190000     151.270000     151.100000     151.190000
75%       252.320000     252.490000     252.170000     252.320000
max       403.700000     403.950000     403.250000     403.800000

             volume           sma5          sma10          sma15  \
count  6.710130e+05  671013.000000  671013.000000  671013.000000
mean   2.031313e+05     148.327779     148.328348     148.328918
std    7.764663e+05     122.663967     122.663466     122.662975
min    0.000000e+00       8.810000       9.160000      10.030000
25%    1.927300e+04      16.250000      16.250000      16.253333
50%    4.731500e+04     151.182000     151.184000     151.186000
75%    1.340640e+05     252.302000     252.292000     252.300000
max    1.610966e+08     403.490000     403.280000     403.020000

               sma20           ema5  ...         fastd        fastksr  \
count  671013.000000  671013.000000  ...  6.710130e+05  671013.000000
mean      148.329487     148.327780  ...  5.257384e+01      50.638115
std       122.662487     122.663855  ...  2.677320e+01      42.891584
min        10.735000       9.552175  ... -4.151938e-12       0.000000
25%        16.250000      16.251464  ...  3.333333e+01       0.000000
50%       151.188000     151.181697  ...  5.277778e+01      51.536692
75%       252.307500     252.289750  ...  7.178494e+01     100.000000
max       402.830000     403.337980  ...  1.000000e+02     100.000000

            fastdsr         ULTOSC          WILLR           ATR  \
count  6.710130e+05  671013.000000  671013.000000  6.710130e+05
mean   5.063821e+01      53.386089     -47.520397  2.143557e-01
std    3.150872e+01      14.807704      32.899850  1.919231e-01
min   -4.979128e-12       0.000000    -100.000000  3.955791e-14
25%    2.633600e+01      44.107324     -75.000000  5.960529e-02
50%    5.155356e+01      52.954082     -50.000000  1.789467e-01
75%    7.466976e+01      62.406731     -18.750000  2.926636e-01
max    1.000000e+02     100.000000      -0.000000  7.515128e+00

              Trange       TYPPRICE    HT_DCPERIOD           BETA
count  671013.000000  671013.000000  671013.000000  671013.000000
mean        0.214352     148.325409      19.651323       0.319659
std         0.303408     122.665268       5.634615       0.677969
min         0.000000       6.750000       6.001490     -28.823327
25%         0.050000      16.250000      15.939119       0.000000
50%         0.150000     151.180000      19.043305       0.161941
75%         0.280000     252.323333      22.728644       0.615890
max        60.150000     403.600000      49.999995      35.829661

[8 rows x 59 columns]
landing/YESBANK_minute_data_with_indicators.csv Columns with null values
[]
landing/YESBANK_minute_data_with_indicators.csv Number of Rows with null values: 0
landing/YESBANK_minute_data_with_indicators.csv Integer data type columns: Index(['volume'], dtype='object')
landing/YESBANK_minute_data_with_indicators.csv Float data type columns: Index(['close', 'high', 'low', 'open', 'sma5', 'sma10', 'sma15', 'sma20',
       'ema5', 'ema10', 'ema15', 'ema20', 'upperband', 'middleband',
```

Yes Bank Statistics


```
[8 rows x 59 columns]
landing/YESBANK_minute_data_with_indicators.csv Columns with null values
[]
landing/YESBANK_minute_data_with_indicators.csv Number of Rows with null values: 0
landing/YESBANK_minute_data_with_indicators.csv Integer data type columns: Index(['volume'], dtype='object')
landing/YESBANK_minute_data_with_indicators.csv Float data type columns: Index(['close', 'high', 'low', 'open', 'sma5', 'sma10', 'sma15', 'sma20',
       'ema5', 'ema10', 'ema15', 'ema20', 'upperband', 'middleband',
```


```
       'lowerband', 'HT_TRENDLINE', 'KAMA10', 'KAMA20', 'KAMA30', 'SAR',
       'TRIMA5', 'TRIMA10', 'TRIMA20', 'ADX5', 'ADX10', 'ADX20', 'APO', 'CCI5',
       'CCI10', 'CCI15', 'macd510', 'macd520', 'macd1020', 'macd1520',
       'macd1226', 'MFI', 'MOM10', 'MOM15', 'MOM20', 'ROC5', 'ROC10', 'ROC20',
       'PPO', 'RSI14', 'RSI8', 'slowk', 'slowd', 'fastk', 'fastd', 'fastksr',
       'fastdsr', 'ULTOSC', 'WILLR', 'ATR', 'Trange', 'TYPPRICE',
       'HT_DCPERIOD', 'BETA'],
      dtype='object')
```



Yes Bank Null Values

and Histogram

<u>Summary of Output:</u>

I performed EDA using a function (Appendix B) that finds the previous information. Only one stock file called "TECHM" only had one row of missing null values. This is good considering that most stock files have around 600,000 records. The data types of the stock files have consistent data types which means that all the indicators have numerical data types like int64 or float64 the besides date variable which was object. The spread of the volume of each stock file varies differently which is good because it shows a wide range of market activity of each stock. Volume is connected to volatility and it is a huge indicator of price changes, so seeing stock have volume means as little as 119.7 like Bajaj Holding to high volume means as high as 2.031313e+05 like YES Bank shows a varied price movements. The histograms of the volumes show multiple left skewed and right skewed stocks which is predicted as some stocks are popular than others and the outliers are some are expected as there could be volatility in the changes. The variability is also proven by the descriptive statistics like the maximum, mean, and minimum of closing price for each stock which shows high and low price spread for each stock. This is good for my model as the different trends and the information from the technical indicators could capture an accurate response. The data now is complete with no nulls and with a varied distribution of each performance indicator due to performing cleaning (Appendix C).

I conducted EDA on my data by searching for numerous things: number of records, number of duplicate records, data type for each variable, statistics for each variable such as mean, the number of rows with null values, the columns with null values, integer data type columns, and float data type columns, as well as made histograms of the volume features. I created a dataproc cluster using N2-Standard-8 machine type with 32 GB memory and used Python 3 Jupyter Notebook environment. I read in all the CSV files and called the EDA function which consists of pandas operations and matplot chart function. ,I conducted data cleaning by going through each file in my landing folder, reading it as a dataframe, dropping any records with missing values (which was just one record), adding a ticker symbol, and saving the files as parquet files in my cleaned folder (Appendix C). Going forward, some challenges in feature engineering that I may have with my data would be first deciding to create time based patterns based on the date. I haven't converted that data type to date time yet as I am deciding if I want to utilize it in my model since I am finding per minute stock prices however creating patterns based on months could be interesting as stock prices are affected by external factors. Other challenges could be scaling or normalization data due to the volatility of the stocks which have an impact on the indicators. I plan to create a model with and without those aspects to potentially see the impacts.

FEATURE ENGINEERING AND MODELING

<u>Feature Engineering/Normalization Planning Chart</u>

*next_day_close is created from close as target variable

Features	**Data Type**	**WindowSpec**	**StringIndex**	**OneHotEncoder**	**VectorAssembler (Numeric Values for Scaler)**	**MinMaxScaler**	**VectorAssembler (All features)**
date	Object (Late Date Type)	WindowSpec (Order)					
Close (next_day_close is created)	Date	WindowSpec (Lead)					
high	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
low	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
open	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
volume	int64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
sma5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
sma10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
sma15	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
sma20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ema5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ema10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ema15	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ema20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
upperband	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
middleband	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
lowerband	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
HT_TRENDLINE	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
KAMA10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
KAMA20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
KAMA30	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
SAR	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
TRIMA5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
TRIMA10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
TRIMA20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ADX5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ADX10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ADX20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
APO	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
CCI5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler

CCI10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
CCI15	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
macd510	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
macd520	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
macd1020	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
macd1520	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
macd1226	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
MFI	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
MOM10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
MOM15	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
MOM20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ROC5	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ROC10	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ROC20	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
PPO	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
RSI14	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
RSI8	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
slowk	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
slowd	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
fastk	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
fastd	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
fastksr	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
fastdsr	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ULTOSC	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
WILLR	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ATR	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
Trange	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
TYPPRICE	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
HT_DCPERIOD	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
BETA	float64				VectorAssembler (Scaler)	MinMaxScaler	VectorAssembler
ticker_symbol	float64	WindowSpec (Partition)	StringIndexer	OneHotEncoder			VectorAssembler

Output:

Features after Next_Day_Close Created

```
+-------------+----------+-----+-----+--------------+
|ticker_symbol|date      |open |close|next_day_close|
+-------------+----------+-----+-----+--------------+
|SAIL         |2015-02-02|76.2 |76.25|76.25         |
|SAIL         |2015-02-02|76.25|76.25|76.15         |
|SAIL         |2015-02-02|76.25|76.15|76.2          |
|SAIL         |2015-02-02|76.15|76.2 |76.1          |
|SAIL         |2015-02-02|76.2 |76.1 |76.15         |
|SAIL         |2015-02-02|76.1 |76.15|76.15         |
|SAIL         |2015-02-02|76.1 |76.15|76.2          |
|SAIL         |2015-02-02|76.2 |76.2 |76.2          |
|SAIL         |2015-02-02|76.2 |76.2 |76.25         |
|SAIL         |2015-02-02|76.2 |76.25|76.25         |
|SAIL         |2015-02-02|76.25|76.25|76.25         |
|SAIL         |2015-02-02|76.25|76.25|76.2          |
```

Features after Ticker symbol Index and Encoder

```
+-------------+-----------+---------------+
|ticker_symbol|tickerIndex|tickerVector   |
+-------------+-----------+---------------+
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
|SAIL         |59.0       |(99,[59],[1.0])|
+-------------+-----------+---------------+
```

Features after numeric features vector created

```
|numeric_features_vector
|
+----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
---------------------------------+
|[76.2,76.3,76.2,7223.0,76.21999999999998,76.34500000000001,76.38666666666667,76.4225,76.25835221526272,76.3114596333582,
76.34776561967712,76.36942002130897,76.2689897951651,76.21999999999998,76.17101020483487,76.44159783549784,76.35923478382
323,76.42494751516341,76.43990799965027,77.0,76.21666666666682,76.31999999999978,76.44954545454544,31.406659068753942,13.
715605427413688,6.068086871591404,-0.0743589743589723,37.03703703704142,-58.93719806762911,-89.94708994708749,-0.05310741
80889703,-0.1110678057325174,-0.0579582702221728,-0.0217299611652634,-0.0558735927547076,44.82421744718789,-0.20000000000
00028,-0.4000000000000057,-0.2999999999999971,-0.130975769482633,-0.2616088947024253,-0.3919007184846501,-0.0972868704658
84,41.56038198198949,37.72416874749003,26.984126984125727,19.04761904761817,33.33333333333176,26.984126984125727,99.99999
999999956,51.74151805656957,36.88332767956419,-84.6153846153853,0.1731559622682564,0.0999999999999943,76.25,32.0374618184
```

Features after final assembler

```
|scaled_features_vector
|tickerVector    |features
|
+--------------------------------------
--------------------------------------
```

```
|[0.00238125,0.0023823871956836014,0.0023886361107236286,4.483644381636041E-5,0.0023889605075619194,0.002111322657145812
3,0.0020879839757953333,0.0020672514870638606,0.0020929956695703296,0.002073143508873769,0.0020651591812963265,0.00206755
24439798293,0.002378383311087232,0.0023889605075619194,0.11970683934741044,0.0020680757199872755,0.0021050184805223287,0.
0021137622686242937,0.002115570445740995,0.0027511839916821348,0.002387996202436818,0.002104640976852552,0.00208353416211
73323,0.31366605200976866,0.136007734819495,0.059105595594393205,0.5387386086373679,0.5555555555555621,0.4557971014492781
5,0.6067019399111218,0.7409985058613777,0.7909542149500991,0.7529545931292629,0.7374020267877887,0.737899505741309,0.4482
421740810718,0.49866892413633185,0.49990167673933844,0.4994183503892242,0.2486910960381275,0.24692093872977025,0.24694147
04796524,0.47617694309179703,0.41560381981989486,0.37724168747490033,0.2698412698411424,0.19047619047616018,0.33333333333
33176,0.2698412698411424,0.9999999999999957,0.5174151805656863,0.36883327674486355,0.15384615384614705,1.080512250250899E
-4,8.72710770559925E-6,0.0023863958224771887,0.5917467380032653,0.3937073043842544]        |(99,[59],[1.0])|(157,[0,1,2,
3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,4
5,46,47,48,49,50,51,52,53,54,55,56,57,117],[0.00238125,0.0023823871956836014,0.0023886361107236286,4.483644381636041E-5,
0.0023889605075619194,0.0021113226571458123,0.0020879839757953333,0.0020672514870638606,0.0020929956695703296,0.002073143
508873769,0.0020651591812963265,0.0020675524439798293,0.002378383311087232,0.0023889605075619194,0.11970683934741044,0.00
20680757199872755,0.0021050184805223287,0.0021137622686242937,0.002115570445740995,0.0027511839916821348,0.00238799620243
6818,0.002104640976852552,0.0020835341621173323,0.31366605200976866,0.136007734819495,0.059105595594393205,0.538738608637
```

Best Model Coefficients/Intercept/Hyperparamters

```
[ ]  # bestModel.coeff. and int
     coefficients = bestModel.stages[5].coefficients
     print("bestModel coefficients", coefficients)
     intercept = bestModel.stages[5].intercept
     print("bestModel intercept", intercept)
```

```
bestModel coefficients [1456.7957334169641,1464.2498543376298,1531.823775635574,-18
bestModel intercept -5725.1534802806555
```

```
# bestModel.hyperparamters
reg_param = bestModel.stages[5].getRegParam()
print("bestModel reg param", reg_param)
elastic_net_param = bestModel.stages[5].getElasticNetParam()
print("bestModel elastic net param", elastic_net_param)
```

```
bestModel reg param 0.5
bestModel elastic net param 0.0
```

```
for i in range(len(testData.columns)-1):
  print(testData.columns[i],coefficients[i])
```

```
date 1456.7957334169641
close 1464.2498543376298
high 1531.823775635574
low -18.28374874031694
open 1479.6585584541654
volume 1489.7591023947352
sma5 1486.2282345247759
sma10 1487.2865004419207
sma15 1489.9811261140978
sma20 1487.0589182152105
ema5 1485.3655373204658
ema10 1484.2695635834184
ema15 1657.049204893177
ema20 1479.6585585197522
upperband 1480.8727419511538
middleband 1486.045293167919
lowerband 1304.5512601121584
HT_TRENDLINE 1178.2562097444857
KAMA10 1124.879098163754
KAMA20 0.15142958524671568
KAMA30 1473.9662142594502
```

Best Model Test predictions/RMSE/RSquared + Average CV Metrics for all Models

```
+------------+----------+-------------+-----+-----+-----+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------
|ticker_symbol|date      |next_day_close|open |high |low  |volume|sma5             |sma10            |sma15            |sma20            |ema5             |ema10
+------------+----------+-------------+-----+-----+-----+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------
|SAIL        |2015-02-02|75.25        |75.4 |75.4 |75.25|15613 |75.41000000000005|75.51499999999999|75.57333333333334|75.625           |75.41968110791045|75.4914
|SAIL        |2015-02-02|75.5         |75.55|75.55|75.35|52825 |75.57000000000005|75.61999999999999|75.67            |75.70749999999998|75.5340106087966 |75.6022
|SAIL        |2015-02-02|75.4         |75.6 |75.6 |75.5 |11845 |75.62000000000005|75.655           |75.69666666666666|75.72999999999998|75.6010159131949 |75.6471
|SAIL        |2015-02-02|75.5         |75.7 |75.7 |75.5 |84171 |75.65000000000005|75.67999999999999|75.71666666666665|75.74499999999998|75.65152386979234|75.6798
|SAIL        |2015-02-02|75.65        |75.65|75.75|75.6 |19256 |75.69000000000003|75.73499999999999|75.76666666666662|75.76749999999996|75.68583959082373|75.7203
|SAIL        |2015-02-02|75.65        |75.7 |75.7 |75.6 |12368 |75.74000000000004|75.76999999999998|75.78333333333327|75.78249999999994|75.7306390793534 |75.7550
|SAIL        |2015-02-02|75.6         |75.65|75.7 |75.65|12640 |75.66000000000004|75.69999999999999|75.73333333333332|75.75249999999997|75.67728580468852|75.6976
|SAIL        |2015-02-02|75.8         |75.75|75.8 |75.7 |9295  |75.86000000000004|75.95499999999997|76.00333333333327|76.02749999999995|75.83833562302394|75.9163
|SAIL        |2015-02-02|75.75        |75.75|75.8 |75.75|5090  |75.78000000000006|75.79999999999997|75.79999999999993|75.80249999999992|75.77424011863727|75.7906
|SAIL        |2015-02-02|75.75        |75.8 |75.85|75.7 |7175  |75.79000000000005|75.80999999999997|75.80333333333326|75.80499999999992|75.7863601779559 |75.7997
|SAIL        |2015-02-02|75.75        |75.8 |75.85|75.75|6024  |75.78000000000004|75.80499999999998|75.79333333333327|75.79749999999993|75.77840689281771|75.7894
|SAIL        |2015-02-02|75.75        |75.75|75.8 |75.75|10683 |75.79000000000005|75.80999999999997|75.7966666666666 |75.80249999999992|75.79261033922657|75.7982
|SAIL        |2015-02-02|75.75        |75.8 |75.8 |75.75|2315  |75.82000000000004|75.80499999999998|75.79999999999993|75.80749999999992|75.80837326325978|75.8084
|SAIL        |2015-02-02|75.8         |75.8 |75.8 |75.75|3105  |75.82000000000004|75.79999999999998|75.79999999999993|75.80999999999992|75.81255989488967|75.8102
|SAIL        |2015-02-02|75.7         |75.8 |75.85|75.75|11980 |75.81000000000004|75.81499999999997|75.8866666666666 |75.93999999999994|75.81077294907206|75.8409
|SAIL        |2015-02-02|75.85        |75.85|75.85|75.75|11999 |75.79000000000005|75.78999999999996|75.80333333333326|75.79999999999993|75.80108446787897|75.7982
|SAIL        |2015-02-02|75.8         |75.8 |75.85|75.8 |1514  |75.82000000000004|75.80999999999997|75.80999999999992|75.81249999999993|75.81532340090179|75.8196
|SAIL        |2015-02-02|75.85        |75.8 |75.85|75.8 |8239  |75.78000000000004|75.87999999999997|75.93999999999993|75.98499999999994|75.8113587031182 |75.8637
|SAIL        |2015-02-02|75.8         |75.85|75.85|75.8 |6100  |75.81000000000004|75.83499999999997|75.9066666666666 |75.95499999999994|75.81615942360808|75.8501
|SAIL        |2015-02-02|75.8         |75.85|75.85|75.8 |10975 |75.81000000000004|75.81999999999996|75.80999999999992|75.80249999999992|75.80681040040079|75.8131
+------------+----------+-------------+-----+-----+-----+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------
only showing top 20 rows

RMSE: 7.4824094473582 R-squared: 0.999994735293215
Average metric [7.960931984249167, 7.9609320809837785, 7.797832097041481, 8.055319407086197, 7.829515673572518, 8.177735124626418]
```

Summary of Output:

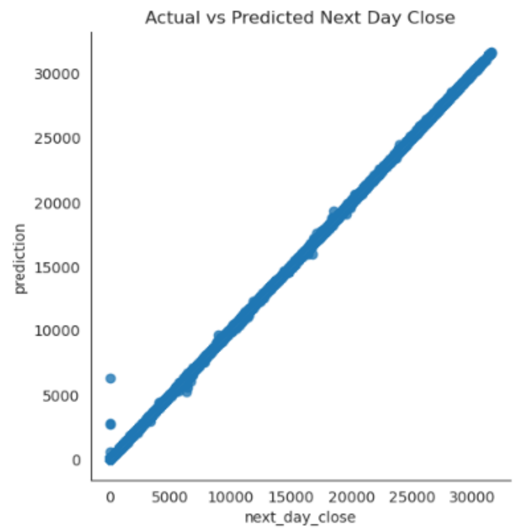
The best model's R squared score is 0.999994735293215, RMSE score is 7.4824094473582, and cross validation average metric score is 7.797832097041481. The R squared score is close to 1 which could suggest that the model is effective and that the predictor variables have a high percentage of explaining the variation in next day closing prices. While it could also mean overfitting, there have been numerous steps to ensure accurate modeling and mitigating the risks of overfitting. The numeric values have been scaled, the ticker symbols have been indexed and encoded, and there has been a cross validator on the training data (Appendix D). In the outputs you can see the transformation of the data which will be trained and tested on. I also included in my model cross validation and hyperparameters for overfitting reduction and model optimization. For the cross validation metric scores, the scores are relatively low as most of them were almost 8 units off the actual next-day closing price. The test data RMSE is also low with being 8 units off, which shows high accuracy. As for hyperparameters, the best model uses regParam of 0.5 which shows a moderate regularization and has an elastic net param of 0 which indicates ridge regression. This means this model has the configuration of L2 regularization. The model's coefficients were adjusted to reduce overfitting leading to smaller coefficients.

First, I imported all relevant pyspark functions and modules that are needed for feature engineering, modeling, and evaluation. After reading in my cleaned files, I created a dataframe. Next, I had to configure pyspark time policy in order to change the datatype of date to date type. I then created the target variable, next day close price, by using the windowspec function and shifting over the close prices using the lead function. Since the last day will have no value, I have to drop null values. I then indexed and encoded the ticker symbol because it is a categorical variable, since different stocks will have different patterns, it will be incorporated in the model. I created a list for numerical variables and scaled them using MinMaxScaler. After putting the features into an assembly, I uploaded the data to the trusted folder. I then loaded it in, and split the data into training and test sets. I created a linear regression estimator as well as an evaluator. Next, I created the pipeline that incorporated all the previous steps including the encoded ticker symbol and scaled features. I also built the grid for parameters for model optimization. After training the model and creating the cross validator, I used the best model to predict the data and calculated the metric scores. One challenge I had was with scaling the numeric features. I originally had to manually scale the dataset using the MinMaxScaler but since I also put the numeric feature assembler and scaler in the pipeline, there was an error. I had to adjust the numeric feature scaling and didn't transform before the pipeline. I also had a problem with uploading and loading the parquet file to the trusted dataset in which I had to manually delete a file in that folder and retry again.

DATA VISUALIZATIONS

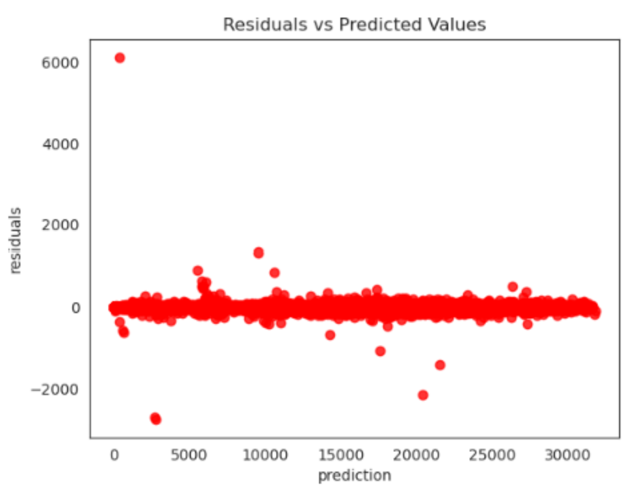
<u>Outputs:</u>

1) <u>Actual vs Predicted Next Day Close Price Scatter Plot</u> - This is a scatter plot that shows the relationship between the next day close actual prices and the prediction of next day close prices from the test model. This graph shows high accuracy as the actual prices match with the prediction prices creating a straight diagonal line. There are a few outliers but this graph enforces accuracy of model and R squared score.

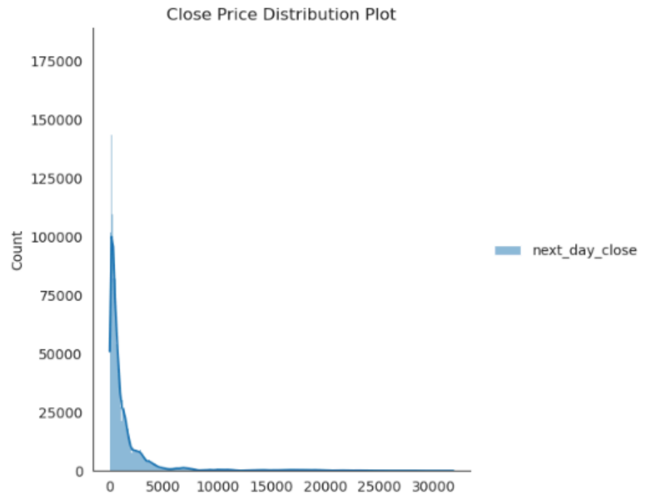


<Figure size 640x480 with 0 Axes>

2) <u>Residuals vs Predicted Values Graph</u>- This residual graph shows that most residuals are close to the value of 0 and that it creates a straight line on 0. There are no other patterns and there are few outliers. This shows the accuracy of the model and that there's no overfitting, enforcing the validation of the model too.



3) Next Day Close Price Distribution Chart- This close price distribution chart shows a right skewed, unimodal distribution which shows most values being less than 2500. This graph shows that skewness affects the model's prediction performance and high R squared score on concentrated low values. There are some high outliers that could affect a model's performance but the model's metrics show that it is outlier resistant and is accurate.

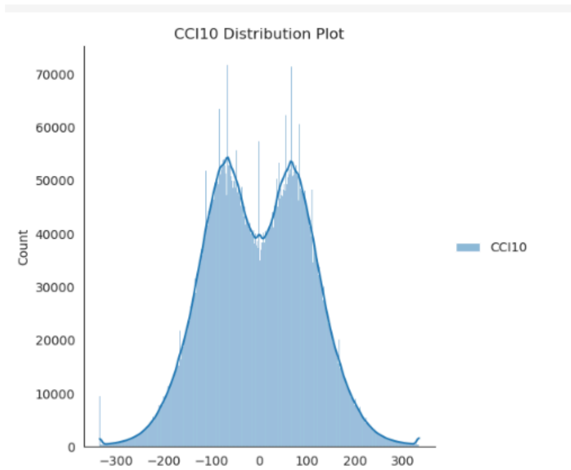



```
<Figure size 640x480 with 0 Axes>
```

4) CCI10 Distribution Plot / Most Important Features - After find the coefficients of each features, the top 10 most important features based on coefficients were: CCI10, macd510, macd520, CCI15, MFI, ema15, ATR, ADX10, WILLR, and macd1226. Based on this, I did a distribution plot of the CCI10 feature (commodity channel index). The distribution plot is a symmetrical, bimodal graph. This shows a balanced and variable feature graph which could explain why it had a high coefficient and why CCI10 is a reliable feature.

```
for i in range(len(testData.columns)-1):
  print(testData.columns[i],coefficients[i])
```

```
date 1456.7957334169641
close 1464.2498543376298
high 1531.823775635574
low -18.28374874031694
open 1479.6585584541654
volume 1489.7591023947352
sma5 1486.2282345247759
sma10 1487.2865004419207
sma15 1489.9811261140978
sma20 1487.0589182152105
ema5 1485.3655373204658
ema10 1484.2695635834184
ema15 1657.049204893177
ema20 1479.6585585197522
upperband 1480.8727419511538
middleband 1486.045293167919
lowerband 1304.5512601121584
HT_TRENDLINE 1178.2562097444857
KAMA10 1124.879098163754
KAMA20 0.15142958524671568
KAMA30 1473.9662142594502
SAR 1492.044002172654
TRIMA5 1489.5956067935292
TRIMA10 -0.015123907704899945
TRIMA20 0.30235335171517114
ADX5 -0.5868214199715767
ADX10 1290.445197824833
ADX20 1.154923251944645
APO 0.24676067279735128
CCI5 1.2869849507988649
CCI10 15022.484087067367
CCI15 -6152.590381945153
```




```
<Figure size 640x480 with 0 Axes>
```

SUMMARY AND CONCLUSIONS

The purpose of this project was to create a linear regression machine learning pipeline to predict the next day's closing price based on the Nifty 100 dataset which included more than 55 important data features on one hundred stocks. The best model had an R squared score of 0.999994735293215, RMSE score of 7.4824094473582, and the cross validation average metric score was 7.797832097041481. The top 10 most important features based on coefficients were CCI10, macd510, macd520, CCI15, MFI, ema15, ATR, ADX10, WILLR, and macd1226. I utilized Google Cloud Platform and PySpark to perform this project which included these following steps: Data Acquisition, EDA, cleaning, feature engineering, modeling, and visualizations. I first sourced the data from Kaggle to GC's storage bucket using a virtual instance on GC through an SSH command terminal and put the data in a landing file. Next, I used JupyterLab through a dataproc cluster to do EDA. I remove null values as well as clear any irrelevant data. I also created a ticker symbol column. After uploading the cleaned data to the cleaned folder in my bucket I did feature engineering by creating the next day closing price as my target variable, indexed and encoded the ticker symbol, scaled the numeric features, then put everything into a vector assembler. I used cross validation and hyperparameters for model optimization and used the best model to create metrics and visualizations. The model used regParam of 0.5 and elastic net param of 0 which correlates to L2 regularization. I put the features after the F.E. phase in the landing folder, the model in the model folder, and visualizations in its own folder of the GC bucket.

In conclusion, this project resulted in a successful linear regression model that has a high R squared score and low RMSE score. This predicting variables model explained almost all the variation in the next day's closing prices This pipeline included Google Cloud and PySpark to handle almost 66 gigabytes of data and transform the data to create the model. This pipeline model ensured to lessen overfitting using three fold cross validation as well as hyperparameters to adjust coefficients. The EDA showed the spread of the data of the stocks which had a great variety in indicator metrics. The visualizations after the model proved the accuracy of the model as well as the spread of impact coefficients compared to target variables. All in all, as the stock market is known to be volatile, this ML pipeline to predict next day closing prices ultimately proved to have consistent performance through cross validation and the metrics of the model enforces the strength of this model.

Github Link: https://github.com/MahirH21/Big-Data-Stock-Prediction-ML-Pipeline

CODE APPENDICES

Appendix A: Code for Data Acquisition

1) Modified VM instance - changed machine type to n2d-standard-2 and increased persistent disk size to 150 GB

2) Created Kaggle API token and downloaded kaggle.json file

3) Opened SSH command terminal, made directory for Kaggle then checked if its there

```
mkdir .kaggle
ls -la
```

4) Uploaded kaggle json file, moved it to kaggle directory, then secured the file

```
ls -l
mv kaggle.json .kaggle/
ls -l .kaggle
chmod 600 .kaggle/kaggle.json
ls -l .kaggle
```

5) Installed software packages and set up a python environment

```
sudo apt -y install zip
sudo apt -y install python3-pip python3.11-venv
python3 -m venv pythondev
cd pythondev
source bin/activate
pip3 install kaggle
kaggle datasets list
```

6) Get Kaggle API command and download kaggle dataset

```
kaggle datasets download -d debashis74017/stock-market-data-nifty-50-stocks-1-min-data
```

7) Unzipped file and see extracted contents

```
unzip stock-market-data-nifty-50-stocks-1-min-data.zip
ls -l
```

8) Authenticated virtual machine, followed link and pasted authorization code after inserting code

```
gcloud auth login
```

9) Created bucket in cloud storage using command. Manually created bucket files (landing, cleaned, trusted, code, models) in cloud console

gcloud storage buckets create gs://my-bigdata-project-mh --project=evocative-bus-433103-s6

--default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access

10) Copied files into the landing file in the project bucket. Used recursive to copy all csv files

gcloud storage cp --recursive *.csv gs://my-bigdata-project-mh/landing/

11) See all the files in the landing folder in the project bucket. Also, double checked manually in the bucket tab on cloud console

gcloud storage ls -l gs://my-bigdata-project-mh/landing/

Appendix B: Code for EDA

```
#Import libraries/modules
from google.cloud import storage
from io import StringIO
import pandas as pd

#EDA Function
def perform_EDA(df: pd.DataFrame, filename: str):

    print(f"{filename} Number of records:")
    print(df.count())
    print(f"{filename} Number of duplicate records: {
len(df)-len(df.drop_duplicates())}" )
    print(f"{filename} Info")
    print(df.info())
    print(f"{filename} Describe")
    print(df.describe())
    print(f"{filename} Columns with null values")
```

```
    print(df.columns[df.isnull().any()].tolist())
    rows_with_null_values = df.isnull().any(axis=1).sum()
    print(f"{filename} Number of Rows with null values:
{rows_with_null_values}" )
    integer_column_list = df.select_dtypes(include='int64').columns
    print(f"{filename} Integer data type columns: {integer_column_list}")
    float_column_list = df.select_dtypes(include='float64').columns
    print(f"{filename} Float data type columns: {float_column_list}")

    # Basic graphs/plots with Matplotlib
    import matplotlib.pyplot as plt
    # Plot a histogram from the volume data
    plt.hist(df['volume'], bins = [0,100,200,300,400,500,1000,1500])
    # Show the plot
    plt.show()

#Point to bucket
source_bucket_name="my-bigdata-project-mh"

#Create a client object that points to GCS
storage_client = storage.Client()

#Get a list of the 'blobs' (objects or files) in the bucket
blobs = storage_client.list_blobs(source_bucket_name, prefix="landing/")

#Make a list
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.csv')]

#Go through each file and do an EDA on each file
for blob in filtered_blobs:
    print(f"File {blob.name} with size {blob.size} bytes")
    source_file_path=f"gs://{source_bucket_name}/landing/{blob.name}"
    df = pd.read_csv(StringIO(blob.download_as_text()), header=0, sep=",")
    perform_EDA(df, blob.name)
```

Appendix C: Code for Data Cleaning

```
#Import libraries/modules
from google.cloud import storage
import pandas as pd
from io import StringIO

#Create a client object that points to GCS
storage_client = storage.Client()

#Point to bucket
source_bucket_name="my-bigdata-project-mh"
bucket=storage_client.get_bucket(source_bucket_name)

# Get a list of the 'blobs' (objects or files) in the landing folder of
the source bucket
blobs = storage_client.list_blobs(source_bucket_name, prefix="landing/")

#Make list
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.csv')]

#Go through each file and perform cleaning
for blob in filtered_blobs:
    print(f"Processing file {blob.name} with size {blob.size} bytes")
    source_file_path=f"gs://{source_bucket_name}/landing/{blob.name}"
    df = pd.read_csv(StringIO(blob.download_as_text()), header=0, sep=",")

    #Remove rows with missing values
    df= df.dropna()

    #Add ticker column
    filename = blob.name.replace('landing/', '')
    filename_parts = filename.split('_')
    ticker_symbol = filename_parts[0]
    df['ticker_symbol'] = ticker_symbol

    #Save the Pandas dataframe to a parquet file and store in a buffer
called filedata
    filedata = df.to_parquet(index=False)
```

```
    #Create a blob with the associated file name
    #Source
code:https://stackoverflow.com/questions/47141291/upload-file-to-google-cl
oud-storage-bucket-sub-directory-using-pythons (for cleaned folder path)

blob2=bucket.blob(f"cleaned/{ticker_symbol}_minute_data_with_indicators.pa
rquet")

    #Upload the filedata buffer to the file blob in the GCS bucket
    blob2.upload_from_string(filedata,
content_type='application/octet-stream')
```

Appendix D: Code for Feature Engineering and Modeling

```
#Import functions modules, regression models, window, evaluation modules,
tuning modules
from pyspark.sql.functions import *
from pyspark.sql import Window
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler, MinMaxScaler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression,
GeneralizedLinearRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator,
RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.tuning import *
from pyspark.ml.evaluation import *


#Create source bucket name for paths
```

```
source_bucket_name = "my-bigdata-project-mh"


#Read parquet files from cleaned bucket and create dataframe
sdf = spark.read.parquet(f"gs://{source_bucket_name}/cleaned")


#Configure time policy settings to allow for date parsing with timezones
#Source code:
https://stackoverflow.com/questions/74984049/inconsistent-behavior-cross-version-parse-datetime-by-new-parser
spark.conf.set("spark.sql.parquet.int96RebaseModeInWrite", "CORRECTED")
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")


# Convert the date column to an actual date data type
sdf = sdf.withColumn("date", to_date(sdf.date, "yyyy-MM-dd"))


# Create a window specification where each stock is within its own
partition and then the data is ordered by Date
windowSpec = Window.partitionBy("ticker_symbol").orderBy("date")


# Use the 'lead' function to "look ahead" one day and create the
next_day_close column
sdf = sdf.withColumn("next_day_close", lead("close", 1).over(windowSpec))


# Remove records with nulls
sdf = sdf.na.drop()


# Create an indexer for the ticker symbol
indexer = StringIndexer(inputCol="ticker_symbol", outputCol="tickerIndex",
handleInvalid="keep")


# Create an encoder for ticker index
```

```python
encoder = OneHotEncoder(inputCol="tickerIndex", outputCol="tickerVector",
dropLast=True, handleInvalid="keep")


# List of numeric columns for scaling and assembling
numeric_columns_list = ["open", "high", "low", "volume", "sma5", "sma10",
"sma15", "sma20", "ema5", "ema10","ema15", "ema20",
                   "upperband", "middleband", "lowerband", "HT_TRENDLINE",
"KAMA10", "KAMA20", "KAMA30", "SAR", "TRIMA5",
                   "TRIMA10", "TRIMA20", "ADX5", "ADX10", "ADX20", "APO",
"CCI5", "CCI10", "CCI15", "macd510", "macd520",
                   "macd1020", "macd1520", "macd1226", "MFI", "MOM10",
"MOM15", "MOM20", "ROC5", "ROC10", "ROC20", "PPO",
                   "RSI14", "RSI8", "slowk", "slowd", "fastk", "fastd",
"fastksr", "fastdsr", "ULTOSC",  "WILLR", "ATR",
                   "Trange", "TYPPRICE", "HT_DCPERIOD", "BETA"]


# Assemble the numeric features into their own feature vector
numeric_assembler = VectorAssembler(inputCols=numeric_columns_list,
outputCol="numeric_features_vector")


# Create a scaler over the numeric_features_vector
scaler = MinMaxScaler(inputCol="numeric_features_vector",
outputCol="scaled_features_vector")


# Create an assembler for scaled features vector and ticker vector
assembler = VectorAssembler(inputCols=["scaled_features_vector",
"tickerVector"], outputCol="features")


# Save dataset to trusted folder
sdf.write.parquet(f"gs://{source_bucket_name}/trusted/features_data_5")


# Load trusted folder
```

```python
sdf =
spark.read.parquet(f"gs://{source_bucket_name}/trusted/features_data_5")


# Split the data into training and test sets
trainingData, testData = sdf.randomSplit([0.7, 0.3], seed=42)


# Create a Linear Regression Estimator
linear_reg = LinearRegression(labelCol="next_day_close")


# Create a regression evaluator (to get RMSE, R2, RME, etc.)
evaluator = RegressionEvaluator(labelCol="next_day_close")


# Create the pipeline
pipeline = Pipeline(stages=[indexer, encoder, numeric_assembler, scaler,
assembler, linear_reg])


# Create a grid to hold hyperparameters
grid = ParamGridBuilder()
grid = grid.addGrid(linear_reg.regParam, [0.0, 0.5, 1.0])
grid = grid.addGrid(linear_reg.elasticNetParam, [0, 1])


# Build the parameter grid
grid = grid.build()

# How many models to be tested
print('Number of models to be tested: ', len(grid))

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid,
evaluator=evaluator, numFolds=3)

# Train the models
all_models  = cv.fit(trainingData)
```

```
# Get the best model from all of the models trained
bestModel = all_models.bestModel


# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(testData)


# Show the predictions for stock prices
test_results.select(
    "ticker_symbol", "date", "next_day_close", "open", "high", "low",
"volume", "sma5", "sma10", "sma15", "sma20",
    "ema5", "ema10","ema15", "ema20", "upperband", "middleband",
"lowerband", "HT_TRENDLINE", "KAMA10", "KAMA20", "KAMA30", "SAR",
"TRIMA5",
    "TRIMA10", "TRIMA20", "ADX5", "ADX10", "ADX20", "APO", "CCI5",
"CCI10", "CCI15", "macd510", "macd520", "macd1020", "macd1520",
"macd1226",
    "MFI", "MOM10", "MOM15", "MOM20", "ROC5", "ROC10", "ROC20", "PPO",
"RSI14", "RSI8", "slowk", "slowd", "fastk", "fastd", "fastksr", "fastdsr",
    "ULTOSC",  "WILLR", "ATR", "Trange", "TYPPRICE", "HT_DCPERIOD",
"BETA", "prediction").show(truncate=False)

# Save model to model folder
bestModel.write().overwrite().save(f"gs://{source_bucket_name}/models/stoc
k_price_model5")

# Calculate RMSE and R2 on test data
rmse = evaluator.evaluate(test_results, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(test_results, {evaluator.metricName: "r2"})
print(f"RMSE: {rmse} R-squared: {r2}")

# Show the average performance over the three folds
print(f"Average metric {all_models.avgMetrics}")

# bestModel.coeff. and int
coefficients = bestModel.stages[5].coefficients
```

```
print("bestModel coefficients", coefficients)
intercept = bestModel.stages[5].intercept
print("bestModel intercept", intercept)

# bestModel.hyperparamters
reg_param = bestModel.stages[5].getRegParam()
print("bestModel reg param", reg_param)
elastic_net_param = bestModel.stages[5].getElasticNetParam()
print("bestModel elastic net param", elastic_net_param)

for i in range(len(testData.columns)-1):
  print(testData.columns[i],coefficients[i])
```

Appendix E: Code for Data Visualization

```
#Actual vs Predicted Values Plot
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = test_results.select('next_day_close', 'prediction').sample(False,
0.15).toPandas()
sns.set_style("white")
sns.lmplot(x='next_day_close', y='prediction', data=df)
plt.title("Actual vs Predicted Next Day Close")
plt.show()


import io
from google.cloud import storage
# Create a memory buffer named img_data to hold the figure
img_data = io.BytesIO()
# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)
```

```
# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket(source_bucket_name)
# Create a blob to hold the data. Give it a file name
blob = bucket.blob("actualvspredictedplot.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)
```

```
#Residual Plot
df =
test_results.select('next_day_close','prediction').sample(False,0.15).toPa
ndas()
df['residuals'] = df['next_day_close'] - df['prediction']

# Set the style for Seaborn plots
sns.set_style("white")

plt.title("Residuals vs Predicted Values")

sns.regplot(x = 'prediction', y = 'residuals', data = df, scatter = True,
color = 'red')

# Create a memory buffer named img_data to hold the figure
img_data = io.BytesIO()
# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)

# Connect to Google Cloud Storage
storage_client = storage.Client()
```

```
# Point to the bucket
bucket = storage_client.get_bucket(source_bucket_name)
# Create a blob to hold the data. Give it a file name
blob = bucket.blob("residualvsprediction.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)
```

```
#Close Price Distribution Plot

df = test_results.select('next_day_close').sample(False,0.55).toPandas()
sns.set_style("white")
sns.displot(df, kde=True, color='green')
plt.title('Close Price Distribution Plot')
plt.show()

# Create a memory buffer named img_data to hold the figure
img_data = io.BytesIO()
# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)

# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket(source_bucket_name)
# Create a blob to hold the data. Give it a file name
blob = bucket.blob("closepricedistribution.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)
```

```
#CCI10 Distribution Plot
df = test_results.select('CCI10').sample(False,0.55).toPandas()
sns.set_style("white")
sns.displot(df, kde=True, color='green')
plt.title('CCI10 Distribution Plot')
plt.show()
```

```python
# Create a memory buffer named img_data to hold the figure
img_data = io.BytesIO()
# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)

# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket(source_bucket_name)
# Create a blob to hold the data. Give it a file name
blob = bucket.blob("CCI10pricedistribution.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)
```