CIS 4130 CMWA

ML Project - Cover Page

Austin Shum

Project Milestone #1:

[Dataset Link]

Introduction:

This data set includes the data from 514 individual stocks with almost 1300 columns of extra data from indicators given by the data. I think that this dataset would be interesting to create a machine learning model because I have always been interested in the world of investments and finances. I am not too sure if creating a machine learning model would help in generating wealth through investing in these companies, but it would be great to try.

Data Set Attributes:

High Price – This is the highest price of a certain stock on a certain date.

Low Price – This is the lowest price of a certain stock on a certain date.

Open Price – This is the price of the stock on a certain date when the market opens at 9:30 AM.

Close Price – This is the price of the stock on a certain date when the market closes at 4:00 PM.

Volume – This is the amount of shares which have been traded on a certain date or over a period of time.

Indicator – These are the mathematical calculations which are made by using price, volume or interest information. I will select a few, including the moving average (MA), momentum (MOM), on-balance volume (OBV) and bollinger bands (BB). These bands are lines charted on a graph, plotted above and below a standard deviation from the MA. By using the upper and lower bands, we can predict the relative high or low prices of a certain stock.

What Will I Predict?

I will try to use the data from indicators and the attributes above in order to aid in predicting the price of all different stocks included within this dataset on the next day's closing price.

Project Milestone #2:

Summary: The goal of this milestone was to download and collect the data into a bucket on Google Cloud Storage. Firstly, a bucket was created in GCS and the necessary folders were created. Next, I set up the virtual machine by using several commands that would install python, install kaggle, and import the files from the Kaggle API into the bucket I created. It would store all the files into the landing folder within the bucket.

Project Milestone #3:

Summary: By using Python, I loaded the data set from GCS and created some descriptive statistics and EDA about the data. Firstly, I imported the necessary modules and imported the data into the code by calling the source bucket and its contents. Next, I performed the EDA by using several functions such as the shape, the number of null values, the various columns within the data, and extracted dates. As I would need to reference the dates due to the need of predicting the following days' closing price, this was an important step. Lastly, I created an extra column labeled ticker_symbol, so the various data could be assigned to their relative ticker symbols.

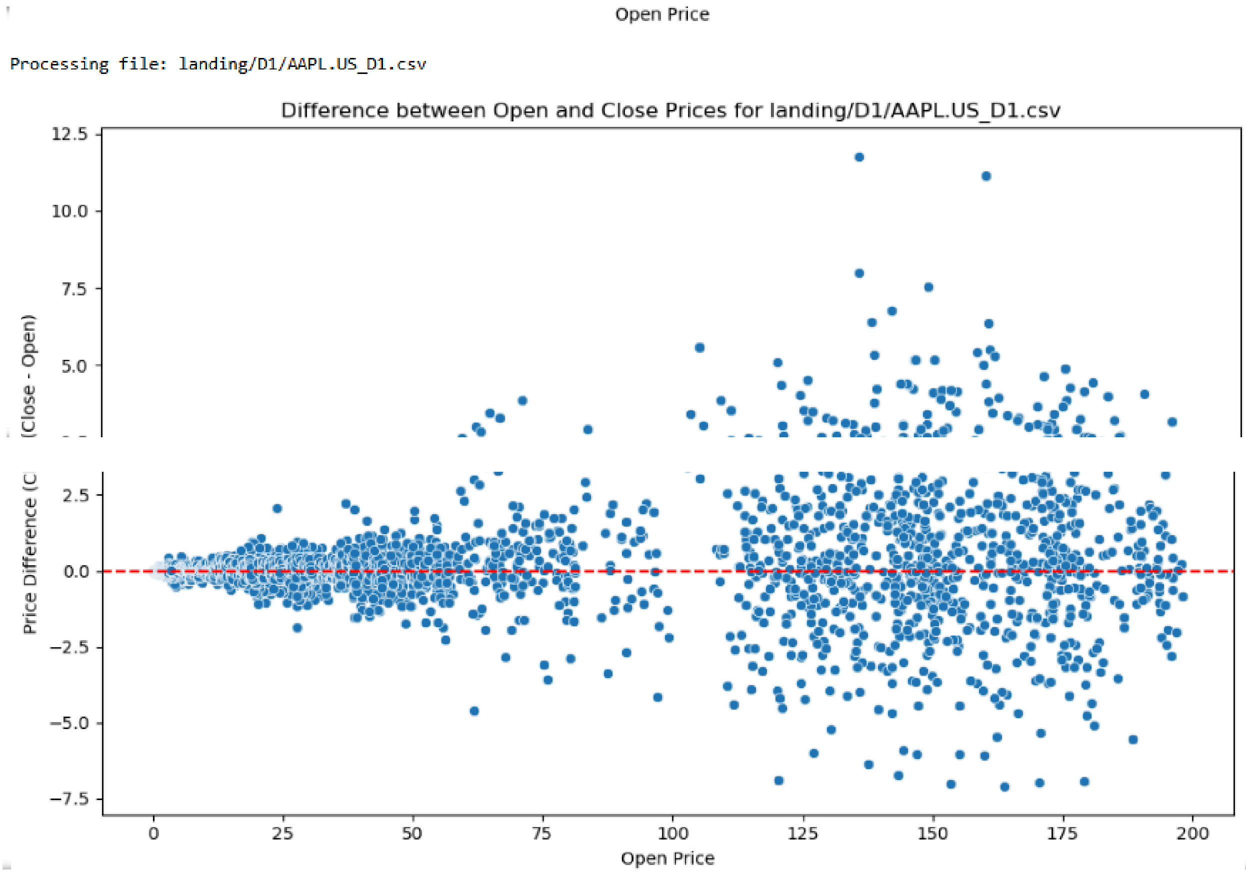
```
[8 rows x 1297 columns]
File landing/D1/CZR.US_D1.csv with size 21245108 bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2888 entries, 0 to 2887
Columns: 1299 entries, datetime to ticker_symbol
dtypes: float64(1296), int64(1), object(2)
memory usage: 28.6+ MB
Number of observations: 2888
Number of missing values in each field:
bbands_3_upperband       2
bbands_3_middleband      2
bbands_3_lowerband       2
bbands_4_upperband       3
bbands_4_middleband      3
                        ..
willr_60                59
willr_70                69
willr_80                79
willr_90                89
```

These are some examples of the outputs I received after running the code:

```
Summary statistics for numerical columns:
             open         high          low        close        volume  \
count  2888.000000  2888.000000  2888.000000  2888.00000  2.888000e+03
mean     28.242926    28.886028    27.592822    28.22973  3.546342e+06
std      28.215217    28.800944    27.582910    28.18175  6.519859e+06
min       4.570000     4.940000     3.300000     4.55000  8.100000e+02
25%       9.340000     9.510000     9.150000     9.35750  6.522150e+05
50%      12.935000    13.140000    12.700000    12.93500  1.437434e+06
75%      44.667500    45.472500    43.550000    44.64000  3.378964e+06
max     119.160000   119.810000   116.600000   119.49000  1.366555e+08
```

```
Length: 1284, dtype: int64
List of variables: ['datetime', 'open', 'high', 'low', 'close', 'volume', 'bbands_3_upperband', 'bbands_3_middleband', 'bband
s_3_lowerband', 'bbands_4_upperband', 'bbands_4_middleband', 'bbands_4_lowerband', 'bbands_5_upperband', 'bbands_5_middleban
d', 'bbands_5_lowerband', 'bbands_6_upperband', 'bbands_6_middleband', 'bbands_6_lowerband', 'bbands_7_upperband', 'bbands_7_
middleband', 'bbands_7_lowerband', 'bbands_8_upperband', 'bbands_8_middleband', 'bbands_8_lowerband', 'bbands_9_upperband',
'bbands_9_middleband', 'bbands_9_lowerband', 'bbands_10_upperband', 'bbands_10_middleband', 'bbands_10_lowerband', 'bbands_12
_upperband', 'bbands_12_middleband', 'bbands_12_lowerband', 'bbands_14_upperband', 'bbands_14_middleband', 'bbands_14_lowerba
nd', 'bbands_16_upperband', 'bbands_16_middleband', 'bbands_16_lowerband', 'bbands_18_upperband', 'bbands_18_middleband', 'bb
ands_18_lowerband', 'bbands_20_upperband', 'bbands_20_middleband', 'bbands_20_lowerband', 'bbands_25_upperband', 'bbands_25_m
```

After reading the data, I created a chart to show the distribution of the data. I imported the necessary modules and referenced the file within the bucket. I calculated the difference in opening and closing columns and created an example with ticker symbol 'AAPL' stock.



The next and last step for this milestone was cleaning the data. In order to clean the data, I read the files within the same bucket, landing folder, and created a function that would run the cleaning operations. This includes keeping columns which are needed within the project, adding a ticker symbol column after referencing the file name, dropping any null values, and saving the newly cleaned data within a different folder in the bucket.

<u>Difficulties</u>:

Creating the main part of this milestone which was the EDA after parsing through the files proved to be quite difficult. It was difficult to be able to parse through all of the CSV files within my landing folder so that the EDA could be performed, but after that, analyzing the data seemed to be easy.

The cleaned dataset for this stock market data includes attributes such as the open, close, high prices, volume and indicators like the momentum, moving average, and on-balance volume. I will use these attributes to create the ML model and make predictions about future prices using this information. I believe one of the most difficult parts of implementing the data into a model would be including and analyzing the indicators and splitting the data into test and training data.

Project Milestone #4

Summary: The point of this milestone was to feature engineering and create the modeling in order to predict the stock price of next days' closing. For the feature engineering portion of the milestone, I first used a window specification to review the following day's closing price for model reference. I had difficulty at first, gathering the datetime information in the DF as it was not previously included within the cleaned parquet files. However, I fixed it and proceeded with the indexing (for ticker symbol data), scaling (features such as high, open, volume) and assembling (combining all features together) into a vector. I created a pipeline with all features prior to the model and saved this.

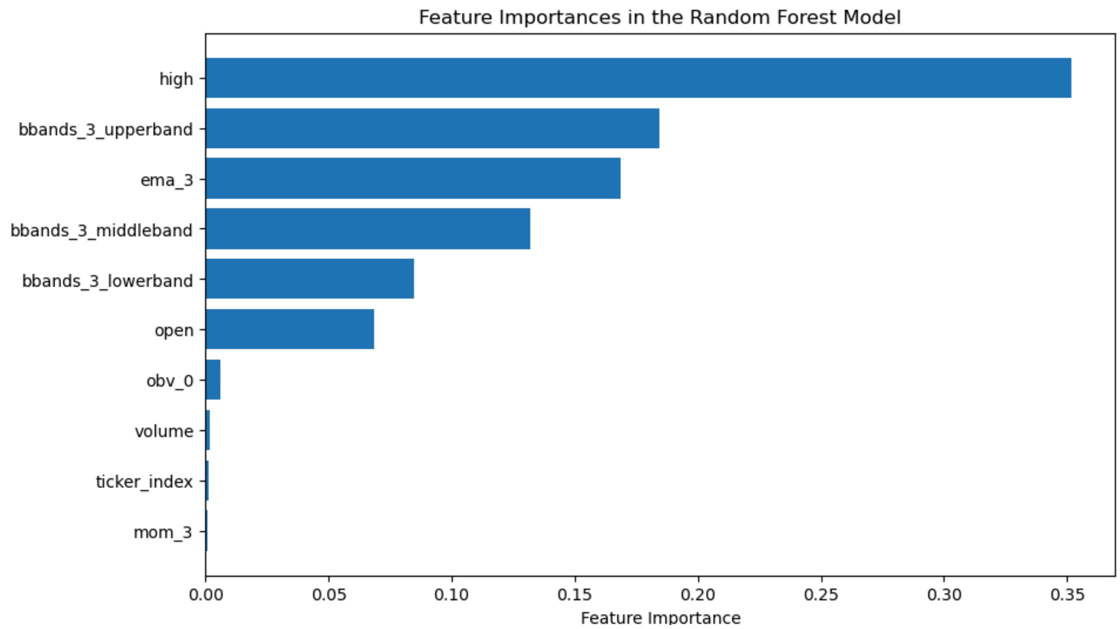
Secondly, I moved onto the modeling portion by defining the random forest model I was going to use for my data and splitting the data into test and training sets. I was able to include the cross-validator over 3 hours but it seems like the model had less accurate predictions. Lastly, I fit the data into the model, first using the training data and then the testing data and followed with RMSE, MAE, and R^2 evaluations to ensure the model was working and was as accurate as intended.

Based on the following results, feature importances, the feature of most importance was moving average price indicator, while of the lower importance was open price. The RMSE and MAE indicate that the model had some error predicting the prices when comparing it to actual closing price. Lastly, the R^2 of 0.984 would mean the model was an okay fit.

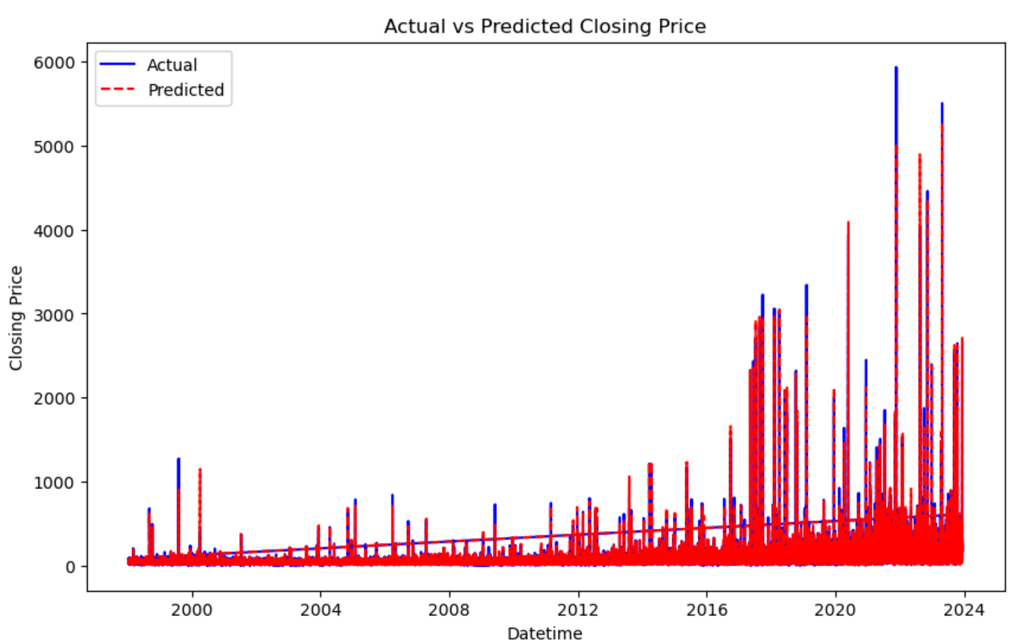
```
Root Mean Squared Error (RMSE) = 21.115899648183415

Mean Absolute Error (MAE) = 8.1807257765551

R-Squared (R2) = 0.9835247834224002
Feature Importances:
datetime: 0.3520112845975745
ticker_symbol: 0.06829989467833736
high: 0.0016100864946969075
close: 0.006082477935932407
open: 0.0008128456852295622
volume: 0.16883962746515507
obv_0: 0.18445926616383362
mom_3: 0.1318217733920169
ema_3: 0.08457997615903919
bbands_3_upperband: 0.0014827674281844242
```

Project Milestone #5

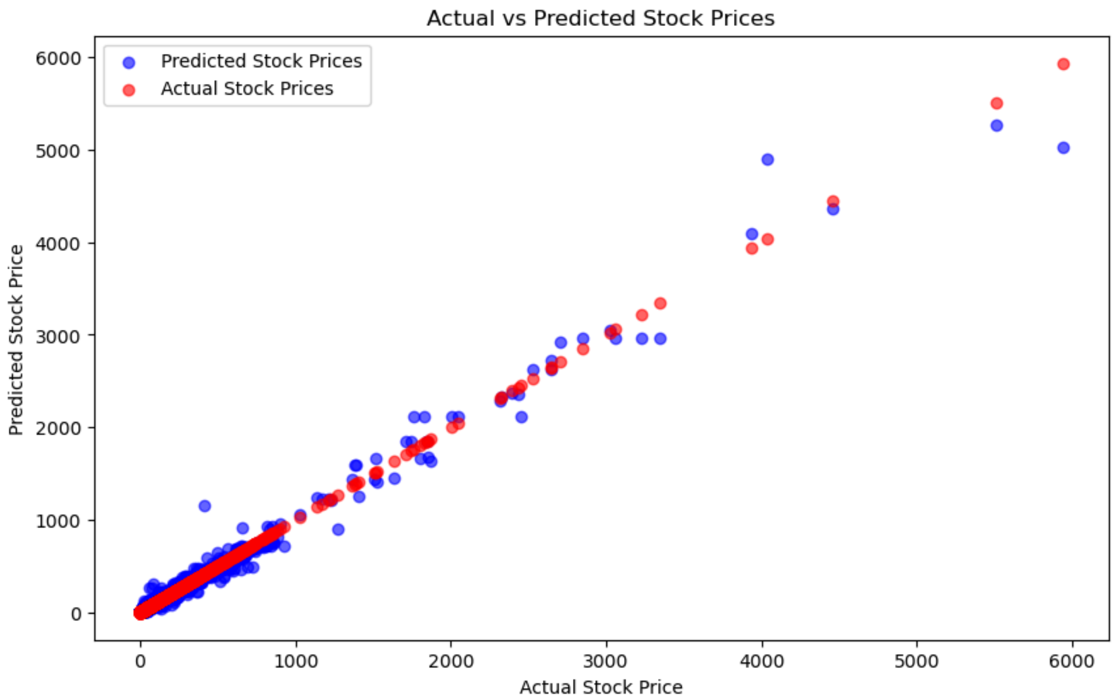
Summary: The last step in the project was to complete the visualizations for the data. To complete this, I created several different charts that would allow me to visualize several important factors, such as sorting the feature importances, the actual vs predicted closing price, and the residual plots.



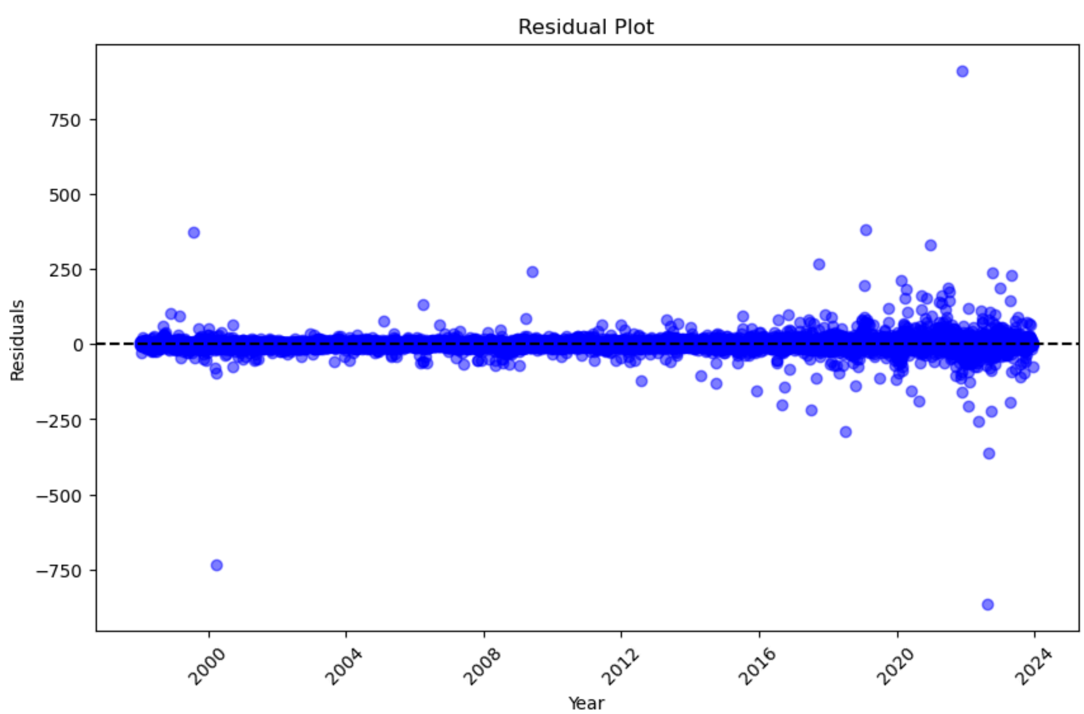
This visualization shows based on the feature importances, ranked by the most important at the top that would be a very big factor in predicting the following day's closing price, to the lowest factor in predicting the price. Based on this chart, the previous day high was a large contributor, of almost 35% in the prediction. The momentum was apparently the lowest factor, contributing to less than 1% of the data.



This bar chart shows the closing price figure, with the blue line being the actual price and the red line being the predicted following day closing price by the model. From a first glance, many of the predictions actually fell short with the higher closing prices, though much of the data was centered in the 0-1000 closing data. This is not the best visualization of the difference, as much of the data is overlapping.



This graph shows the actual vs predicted stock prices by using a scatter plot. The red dots show the actual stock price, versus the predicted stock price in blue. It is an important visualization because it does show us how far off the model is in another format. The Y axis shows us the predicted stock price, versus the X axis that shows the actual stock price.



Lastly, the residual plot indicates how far the predicted value was along the different years, and whether the predicted value was above or below the actual value. Based on this graph, there are many more outliers in the later years closer to 2024, in which the difference between the actual and predicted was very large.

Project Milestone #6

Summary: This project was completed through downloading and importing a large dataset from the internet, analyzing, cleaning, and creating models that would help predict the following days' closing stock prices. Through feature engineering, cross validating, and creating hyperparameters for the data, we were able to create a relatively reliable model that could aid us in the prediction. The conclusion I was able to make is that it is very difficult to predict future results from the stock market, from past results. Much of the modeling we do, even with thorough testing, would not be verifiably able to predict with a considerable amount of accuracy.

Appendix A

Setting Up VM:

1. mkdir .kaggle
2. mv kaggle.json .kaggle/
3. chmod 600 .kaggle/kaggle.json
4. ls -la .kaggle
5. sudo apt -y install zip
6. sudo apt -y install python3-pip python3.11-venv
7. python3 -m venv pythondev
8. cd pythondev
9. source bin/activate
10. pip3 install kaggle
11. kaggle datasets lis

Kaggle API + Unzip:

1. kaggle datasets download -d extra-us-stocks-market-data
2. unzip -l extra-us-stocks-market-data

Creating Bucket + Storing Files:

1. gcloud storage buckets create gs://my-big-data-as --project=cis-4130-project-435001 --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access *(gives me an error)*
2. gcloud auth login
3. gcloud storage buckets create gs://my-big-data-as --project=cis-4130-project-435001 --default-storage-class=STANDARD --location=us-central1
4. gsutil cp -r * gs://my-big-data-as/landing/

Appendix B

#Import the storage module
from google.cloud import storage
from io import StringIO
import pandas as pd

#Source for the files
source_bucket_name = "my-big-data-as"

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

#Get a list of the files in the bucket using blobs
blobs = storage_client.list_blobs(source_bucket_name, prefix="landing/D1")

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

#Define the EDA function
def perform_eda(df):

    #Number of observations
    num_observations = df.shape[0]
    print(f"Number of observations: {num_observations}")

    #Number of missing fields
    missing_values = df.isnull().sum()
    print("Number of missing values in each field:")
    print(missing_values[missing_values > 0])

    #List of variables

```
    variables = df.columns.tolist()
    print(f"List of variables: {variables}")

    #Summary for date variables
    date_columns = df.select_dtypes(include=['datetime', 'datetime64']).columns
        if date_columns.size > 0:
            for date_col in date_columns:
                min_date = df_cleaned[date_col].min()
                max_date = df_cleaned[date_col].max()
                print(f"Min date for {date_col}: {min_date}")
                print(f"Max date for {date_col}: {max_date}")

#Iterate through the list and print out their names

for blob in filtered_blobs:
    print(f"file {blob.name} with size {blob.size} bytes")
    source_file_path = f"gs://{source_bucket_name}/landing/D1/{blob.name}"
    df = pd.read_csv(StringIO(blob.download_as_text()), header=0, sep=",")
    filename = blob.name.replace('landing/D1/', '')
    filename_parts = filename.split('_')
    ticker_symbol = filename_parts[0]

    #Add a new column using ticker name
    df['ticker_symbol'] = ticker_symbol
    df.info()  # Display DataFrame info

    #Perform EDA
    perform_eda(df)

#Create a chart showing the relationship between 2 variables, open and difference in close price.

#Import libraries

import matplotlib.pyplot as plt
import seaborn as sns

#Source for the files
source_bucket_name = "my-big-data-as"
```

```
#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/D1")

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

#Loop through all CSV files using for loop
for blob in filtered_blobs:
    print(f"File: {blob.name}")

    #Read the CSV content into a DataFrame
    csv_data = blob.download_as_text()
    df = pd.read_csv(StringIO(csv_data))

   #For Open and Close Columns, calculate the difference
    if 'open' in df.columns and 'close' in df.columns:
        #Calculate the difference between open and close prices
        df['pricedifference'] = df['close'] - df['open']

        #Creating the plot
        plt.figure(figsize=(10, 6))
        sns.scatterplot(x='open', y='pricedifference', data=df)
        plt.title(f'Difference between Open and Close Prices for {blob.name}')
        plt.xlabel('Open Price')
        plt.ylabel('Price Difference (Close - Open Price)')
        plt.tight_layout()
        plt.show()
```

Appendix C

```
#Importing libraries
from google.cloud import storage
from io import StringIO
import pandas as pd

#Source for the files
source_bucket_name = "my-big-data-as"
```

```
#Create an object for cloud storage
storage_client = storage.Client()

#List of files within the bucket, under the landing folder
blobs = storage_client.list_blobs(source_bucket_name, prefix="landing")

#Data cleaning function
def clean_data(df):

    #Keep needed columns/attributes only, use copy so it does not give me error
    columns_to_keep = [‘datetime’, ‘ticker_symbol’, 'high', 'close', 'open', 'volume', 'obv_0',
'mom_3', 'ema_3', 'bbands_3_upperband', 'bbands_3_middleband', 'bbands_3_lowerband']
    df = df[columns_to_keep].copy()

    #Add Ticker Symbol Column
    df['ticker_symbol'] = ticker_symbol

    #Ensure datetime format is correct for Pyspark
    df['datetime'] = df['datetime'].astype('datetime64[us]')


    #Remove nulls
    df = df.dropna()

    #Printing head to check DF
    print(df.head())

    #Returning back to DF
    return df

#Loop through all CSV files using for loop
for blob in blobs:
    if blob.name.endswith('.csv'):
        print(f"Processing file: {blob.name}")

        #Read the CSV into DF
        csv_data = blob.download_as_text()
        df = pd.read_csv(StringIO(csv_data))
```

```
    #Extract the ticker symbol
    filename = blob.name.split('/')[-1]
    ticker_symbol = filename.split('_')[0]

    #Cleaning the data by calling function
    df = clean_data(df)

    #Writing the cleaned DF to the cleaned folder as a Parquet file

    cleaned_file_path =
f"gs://{source_bucket_name}/cleaned/{blob.name.split('/')[-1].replace('.csv', '.parquet')}"
    df.to_parquet(cleaned_file_path, index=False)
    print(f"Cleaned files written to: {cleaned_file_path}")
```

Appendix D

```
from pyspark.ml.feature import StringIndexer, StandardScaler, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import SparkSession
from pyspark.sql.functions import lead
from pyspark.sql.window import Window

#Initialize Spark
spark = SparkSession.builder.appName("StockDataModel").getOrCreate()

#Define the path to the cleaned data in GCS
cleaned_data_path = "gs://my-big-data-as/cleaned/*.parquet"

#Load the cleaned data into a Spark DataFrame
df = spark.read.parquet(cleaned_data_path)

#Using the full sample data

#Show the first few rows
df.show(5)

#Create a Window specification to calculate the next day's closing price
windowSpec = Window.partitionBy("ticker_symbol").orderBy("datetime")
```

#Use the LEAD function to look ahead one day and get closing price
df = df.withColumn("next_day_close", lead("close", 1).over(windowSpec))

#Drop the last row null for each since it will have no value for next_day_close
df = df.dropna(subset=["next_day_close"])

#Ensure columns are in double type for scaling
df = df.withColumn("volume", df.volume.cast('double'))

#List of columns to scale
columns_to_scale = ["high", "open", "volume", "obv_0", "mom_3", "ema_3",
"bbands_3_upperband", "bbands_3_middleband", "bbands_3_lowerband"]

#Assemble columns to scale into one vector in the pipeline
assembler_ = VectorAssembler(inputCols=columns_to_scale,
outputCol="columns_to_scale_vector")

#Scale vector using StandardScaler
scaler = StandardScaler(inputCol="columns_to_scale_vector", outputCol="scaled_vector",
withStd=True, withMean=False)

#StringIndexer for ticker_symbol (categorical feature)
indexer = StringIndexer(inputCol="ticker_symbol", outputCol="ticker_index")

#Assemble the final features into a feature vector
final_assembler = VectorAssembler(
inputCols=["scaled_vector", "ticker_index"], # Includes scaled features and ticker_index
outputCol="features"
)

#Create the pipeline with all the stages
pipeline = Pipeline(stages=[assembler_, scaler, indexer, final_assembler])

#Transform dataframe based on pipeline
df_transformed = pipeline.fit(df).transform(df)

#Save transformed feature vectors to trusted folder before model
df_transformed.write.parquet("gs://my-big-data-as/trusted/transformed_feature_vectors")

#Define the Random Forest model
```
rf = RandomForestRegressor(featuresCol='features', labelCol='next_day_close', maxBins=2048)
```

#Create the pipeline and define the stages along with RF
```
pipeline = Pipeline(stages=[assembler_, scaler, indexer, final_assembler, rf])
```

#Split the data into training and testing sets
```
train_data, test_data = df_transformed.randomSplit([0.8, 0.2], seed=49)
```

#Cache training data to speed up cross-validation
```
train_data.cache()
```

#Set up cross-validation with hyperparameter tuning
```
paramGrid = ParamGridBuilder() \
   .addGrid(rf.numTrees, [10, 20]) \
   .addGrid(rf.maxDepth, [5, 10]) \
   .build()
```

#Regression evaluator for RMSE
```
rmse_evaluator = RegressionEvaluator(labelCol="next_day_close", predictionCol="prediction",
metricName="rmse")
```

#Set up the CrossValidator with RandomForest model, parameter grid, and evaluator
```
cv = CrossValidator(estimator=rf,
                    estimatorParamMaps=paramGrid,
                    evaluator=rmse_evaluator,  # Evaluator for RMSE
                    numFolds=2)
```

#Train the model using cross-validation
```
cvModel = cv.fit(train_data)
```

#Make predictions on the test set
```
predictions = cvModel.transform(test_data)
```

#Evaluate the model
```
rmse = rmse_evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) = {rmse}")
```

#Mean absolute error:

```
mae_evaluator = RegressionEvaluator(labelCol="next_day_close", predictionCol="prediction",
metricName="mae")
mae = mae_evaluator.evaluate(predictions)
print(f"Mean Absolute Error (MAE) = {mae}")
```

#R^2

```
r2_evaluator = RegressionEvaluator(labelCol="next_day_close", predictionCol="prediction",
metricName="r2")
r2 = r2_evaluator.evaluate(predictions)
print(f"R-Squared (R2) = {r2}")
```

#Print the feature importances for best model after CV

```
# Get feature importance from the best model
rf_model = cvModel.bestModel  # Best model after cross-validation
```

#Extract feature importances

```
feature_importances = rf_model.featureImportances
```

#Print the feature importances

```
print("Feature Importances: ")
for feature, importance in zip(df_transformed.columns, feature_importances):
    print(f"{feature}: {importance}")
```

#Save the trained model to a location

```
rf_model.save("gs://my-big-data-as/models/stock_model")
```

#Save the test predictions to GCS

```
predictions.select("ticker_symbol", "datetime", "next_day_close",
"prediction").write.parquet("gs://my-big-data-as/models/test_predictions.parquet")
```

Appendix E

```
from pyspark.ml.regression import RandomForestRegressionModel
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

#Load the model from gcs

```
model_path = "gs://my-big-data-as/models/stock_model"
```

```
rf_model = RandomForestRegressionModel.load(model_path)

#Load test predictions
test_predictions_path = "gs://my-big-data-as/models/test_predictions.parquet"
predictions = spark.read.parquet(test_predictions_path)

#Convert the prediction data to Pandas for visualization
predictions_pd = predictions.select("ticker_symbol", "datetime", "next_day_close",
"prediction").toPandas()

#Check predictions data
print(predictions_pd.head())

#Call feature importance
feature_importances = rf_model.featureImportances

#Convert feature importance to an numpy array
feature_importances_array = feature_importances.toArray()

#Features used in the model
features = ["high", "open", "volume", "obv_0", "mom_3", "ema_3",
            "bbands_3_upperband", "bbands_3_middleband", "bbands_3_lowerband",
"ticker_index"]

#Sort feature importance by most important to least
sorted_idx = np.argsort(feature_importances_array)[::-1]

#Bar graph for feature importances
plt.figure(figsize=(10, 6))
plt.barh(np.array(features)[sorted_idx], feature_importances_array[sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Feature Importances in the Random Forest Model')
plt.gca().invert_yaxis()  # Invert the axis to have the most important feature on top
plt.show()

#Charting the actual vs predicted closing price using bar chart by years
plt.figure(figsize=(10, 6))
plt.plot(predictions_pd['datetime'], predictions_pd['next_day_close'], label='Actual', color='blue')
plt.plot(predictions_pd['datetime'], predictions_pd['prediction'], label='Predicted Price',
color='red', linestyle='--')
plt.xlabel('Datetime')
```

```
plt.ylabel('Closing Price')
plt.title('Actual vs Predicted Closing Price')
plt.legend()
plt.show()
```

#Extract actual and predicted stock prices

```
y_true = predictions.select("next_day_close").rdd.flatMap(lambda x: x).collect()
y_pred = predictions.select("prediction").rdd.flatMap(lambda x: x).collect()
```

#Scatter plot of actual vs predicted stock prices

```
plt.figure(figsize=(10, 6))
plt.scatter(y_true, y_pred, alpha=0.6, color='blue', label="Predicted Stock Prices")
plt.scatter(y_true, y_true, alpha=0.6, color='red', label="Actual Stock Prices")
plt.xlabel('Stock Price')
plt.ylabel('Stock Price')
plt.title('Actual vs Predicted Stock Prices')
plt.legend()
plt.show()
```

#Calculate Residual

```
predictions_pd['residual'] = predictions_pd['next_day_close'] - predictions_pd['prediction']
```

#Plotting residuals

```
plt.figure(figsize=(10, 6))
plt.scatter(predictions_pd['datetime'], predictions_pd['residual'], alpha=0.5, color='blue')
plt.axhline(y=0, color='black', linestyle='--')
plt.xlabel('Year')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.xticks(rotation=45)
plt.show()
```