Stock Price Prediction

CIS 4130

Jake Li

LiJake2001@gmail.com

Project Proposal

I found a dataset on Kaggle called Stock Market Data - Nifty 50 Stocks (1 min) data, and it has 33.37 GB of data in the form of comma-separated values (CSV) files. The CSV files contain stock prices per minute from January 2015 to February 2022. There are 52 CSV files which include the Nifty 50 index, the Nifty Bank index, and 49/50 (one missing) individual stocks from the Nifty 50 index. The Nifty 50 index is a benchmark Indian stock market index that represents the weighted average of 50 of the largest Indian companies listed on the National Stock Exchange. Each CSV file has 59 attributes which are date, open price, high price, low price, close price, volume, and 55 attributes from technical indicators used by market experts. The original dataset without the technical indicators can be found here.

For this project, I am going to use the 49 individual stocks from the Nifty 50 index which means excluding the Nifty 50 index and Nifty Bank index. I want to create a machine learning model that predicts the stock closing price. I also want to discover what technical indicator(s) best predicts the stock price.

Notes: The author of the Kaggle dataset updated its version and changed the name to Stock Market Data - Nifty 100 Stocks (1 min) data. For this project, I am using the dataset from version 2.

Data Acquisition

The following steps are for downloading Kaggle dataset directly to Amazon S3.

1) Set up Amazon EC2 (t3.xlarge instance) and configure AWS CLI as the Administrator user.

2) Use Amazon CLI to create a bucket:

```
$ aws s3api create-bucket --bucket cis4130-project-jakeli --region us-east-2
--create-bucket-configuration LocationConstraint=us-east-2
```

3) Install Python libraries (Do this as the regular user, not root):

```
pip3 install boto3 pandas fsspec s3fs
```

4) Follow the instructions from here to edit some Python source code in the Kaggle API to allow the API to download the Kaggle dataset directly to Amazon S3.

5) Download the Kaggle file to Amazon S3:

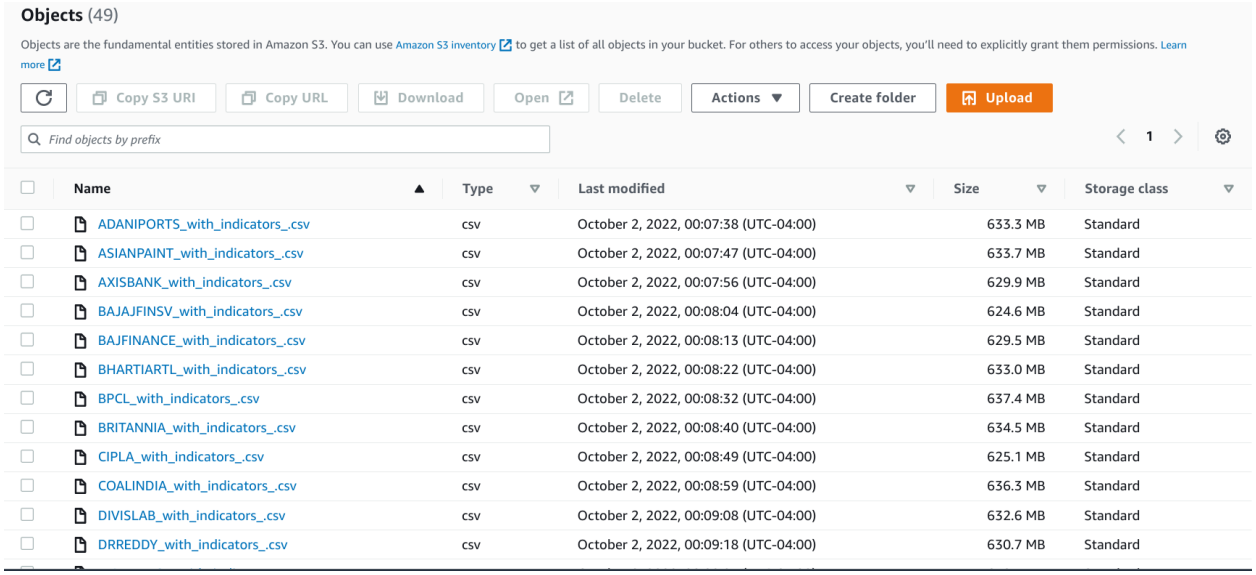
```
$ kaggle datasets download --quiet -d
debashis74017/stock-market-data-nifty-50-stocks-1-min-data -p - | aws s3 cp -
s3://cis4130-project-jakeli/stock_data.zip
```

6) Check the S3 bucket and see if the file was downloaded:

```
$ aws s3 ls s3://cis4130-project-jakeli/
```

7) Go back to the instructions here and follow the code to unzip the zip file from S3 using EC2.
- If the unzipping fails, try adding a 4GB swap file by creating a separate EBS storage and attach it to your EC2 instance. Follow the instructions here. You can also upgrade your instance type.
See appendix A-1 for the source code.

8) Check the S3 bucket to see if the zip file is unzipped and delete the unnecessary files (data description, Nifty 50 index, and Nifty Bank index). I also renamed "M_M_with_indicators_.csv" to "MM_with_indicators_.csv" to have consistent file names.



Adapted from https://github.com/Kaggle/kaggle-api/issues/315

Descriptive Statistics

Note: I changed the EC2 instance type to z1d.2xlarge and created a 32GB swap file using the instance store provided. Follow the instructions here to create a 32GB partition and then make it your swap file.

First, I merged all the CSV files together (adapted from here) and this process will take about 1 to 2 hours. See appendix B-1 for the source code.

After the merging process was completed, I performed descriptive statistics and the following are my results. There are 31,439,316 rows and 60 columns. There are no null values or duplicates. The dates range from 02/02/2015 to 02/18/022. The data types are mostly floats except for stock (object), date (object), and volume (integer). See appendix B-2 for the source code.

Statistics summary (open, high, low, close, volume):

```
>>> round(df[["open", "high", "low", "close", "volume"]].describe())
              open         high         low        close       volume
count  31439316.0  31439316.0  31439316.0  31439316.0   31439316.0
mean       1818.0      1819.0      1817.0      1818.0      16340.0
std        3394.0      3396.0      3393.0      3394.0      70556.0
min           0.0         0.0         0.0         0.0          0.0
25%         333.0       334.0       333.0       333.0       1149.0
50%         687.0       687.0       686.0       687.0       4348.0
75%        1718.0      1719.0      1717.0      1718.0      13719.0
max       32000.0     32027.0     31901.0     32018.0  138638983.0
```

It seems that there are outliers based on the maximum value from the statistics summary. However, I will be using random forest regressor which handles outliers very well.

Coding and Modeling

Note: I created a cluster in AWS EMR with the following configurations:
Release: emr-6.8.0
Applications: Spark: Spark 3.3.0 on Hadoop 3.2.1 YARN with and Zeppelin 0.10.1
Instance type: m5.xlarge
Number of instances: 5 (1 master and 4 core nodes)

I am going to predict the stock closing price per minute using all the features with random forest regressor. I am using random forest regressor because it's a good model to use for predicting non-linear continuous values. It also handles outliers very well.

First, I imported the libraries and set the Spark logging level to only show errors. Second, I defined a schema for the spark dataframe and imported the "all_stock.csv" in my S3 bucket, and changed column names to lowercase. Then I created a selected_features list to include every feature except stock, date, and close. Next, I used VectorAssembler to create a column that holds all the features in the vector form and split the data by date into 70% training and 30% testing. Finally, I created the random forest regressor model, fitted the training data, made predictions using testing data, and evaluated the model using R squared (R^2), root mean square deviation (RMSE), and mean absolute error (MAE). The results are:

```
>>> print("The R^2 for using all features is:", round(r2, 2))
The R^2 for using all features is: 0.93
>>> print("The RMSE for using all features is:", round(rmse, 2))
The RMSE for using all features is: 1183.16
>>> print("The MAE for using all features is:", round(mae, 2))
The MAE for using all features is: 340.13
```

I exported a dataframe that contains the actual closing price and the prediction of the closing price to my S3 bucket.
I found the top 10 features by extracting feature importance and ran the model again with only those 10 features. I got very similar results by just using 10 features.

```
>>> print("The R^2 for using top 10 features only is:", round(r2, 2))
The R^2 for using top 10 features only is: 0.93
>>> print("The RMSE for using top 10 features only is:", round(rmse, 2))
The RMSE for using top 10 features only is: 1171.65
>>> print("The MAE for using top 10 features only is:", round(mae, 2))
The MAE for using top 10 features only is: 341.21
```

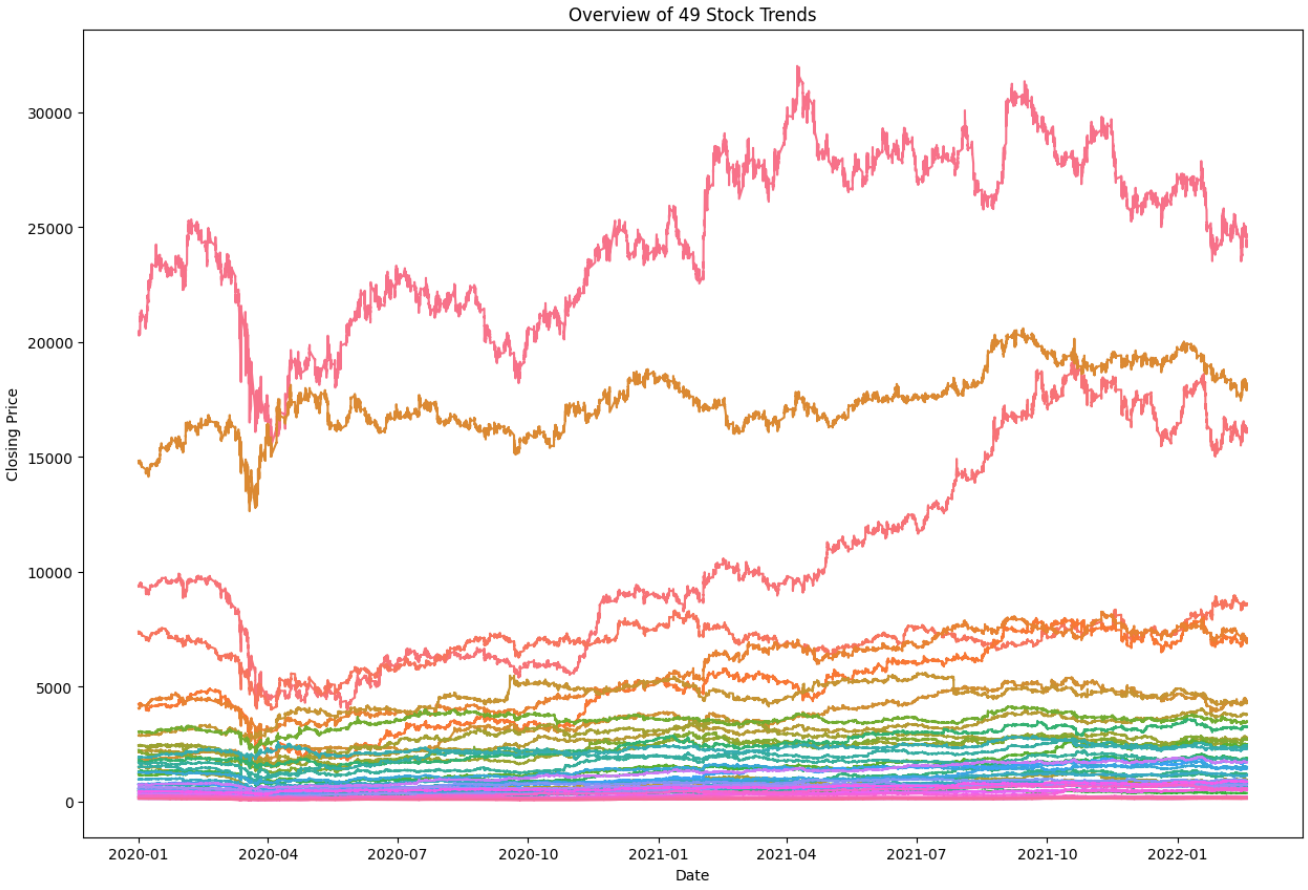
See appendix C-1 for the source code.

Visualizing Results

Note: Install s3fs, pandas, matplotlib, and seaborn before opening PySpark.

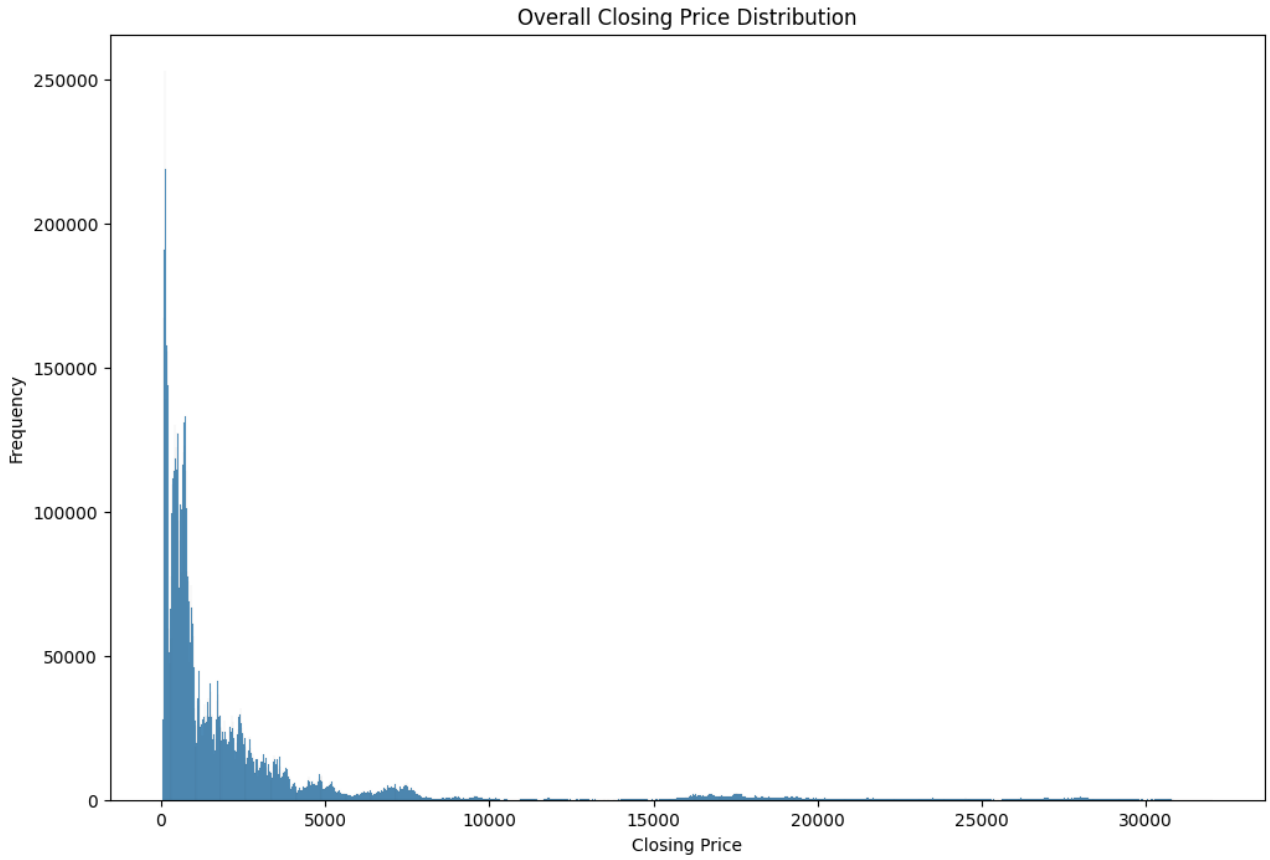
1)

sns.lineplot(x="date", y="close", hue="stock", data=df, legend=False).set(title='Overview of 49 Stock Trends')



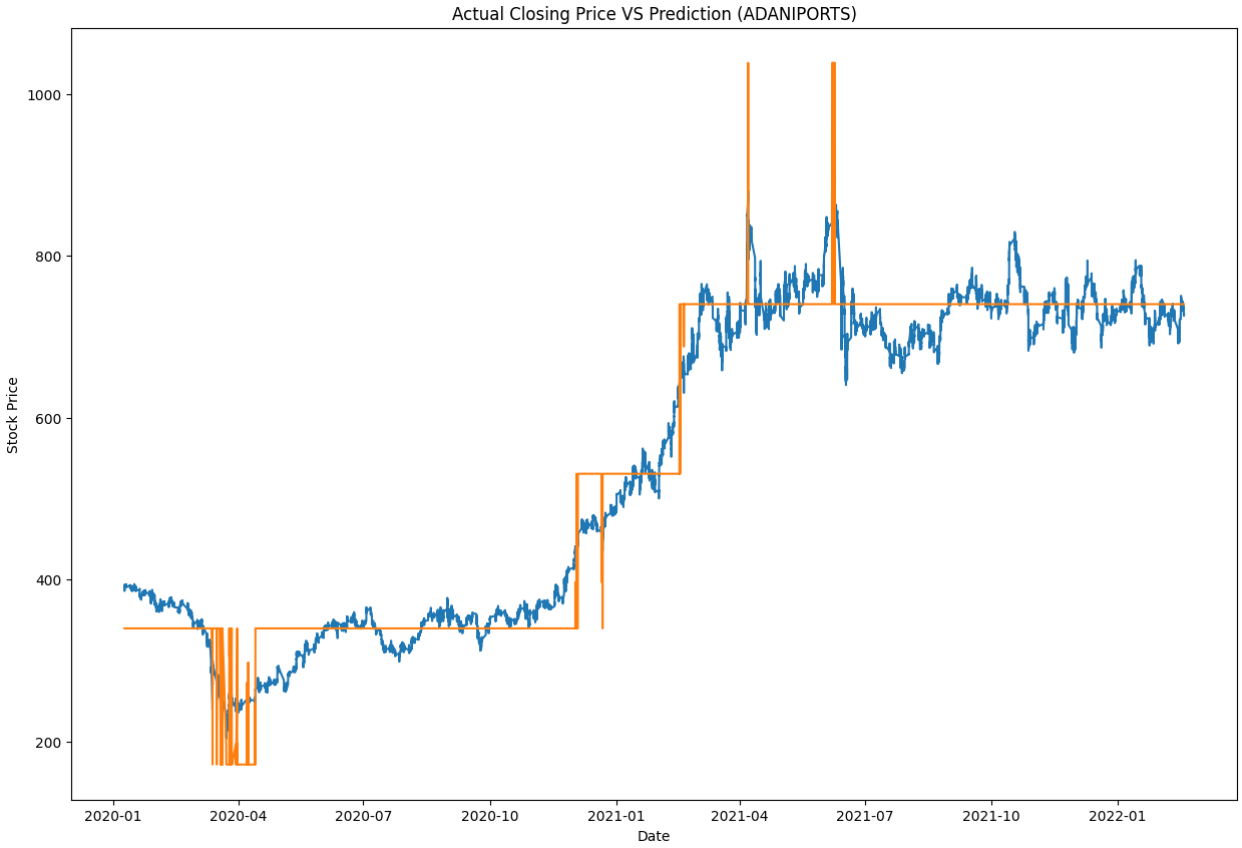
This time series plot shows the overall trends of each 49 individual stocks from January 2020 to February 2022. It's not the prettiest plot but the point is to show general trends of the closing price for each stock.

2)
sns.histplot(x="close", data=df).set(title="Overall Closing Price Distribution")



This distribution shows the closing price is heavily right-skewed. This makes sense because we can see the time series plot above and confirms that some stocks went above $15,000. These might be considered as outliers but I am not going to remove them because random forest regressor are not sensitive to outliers.

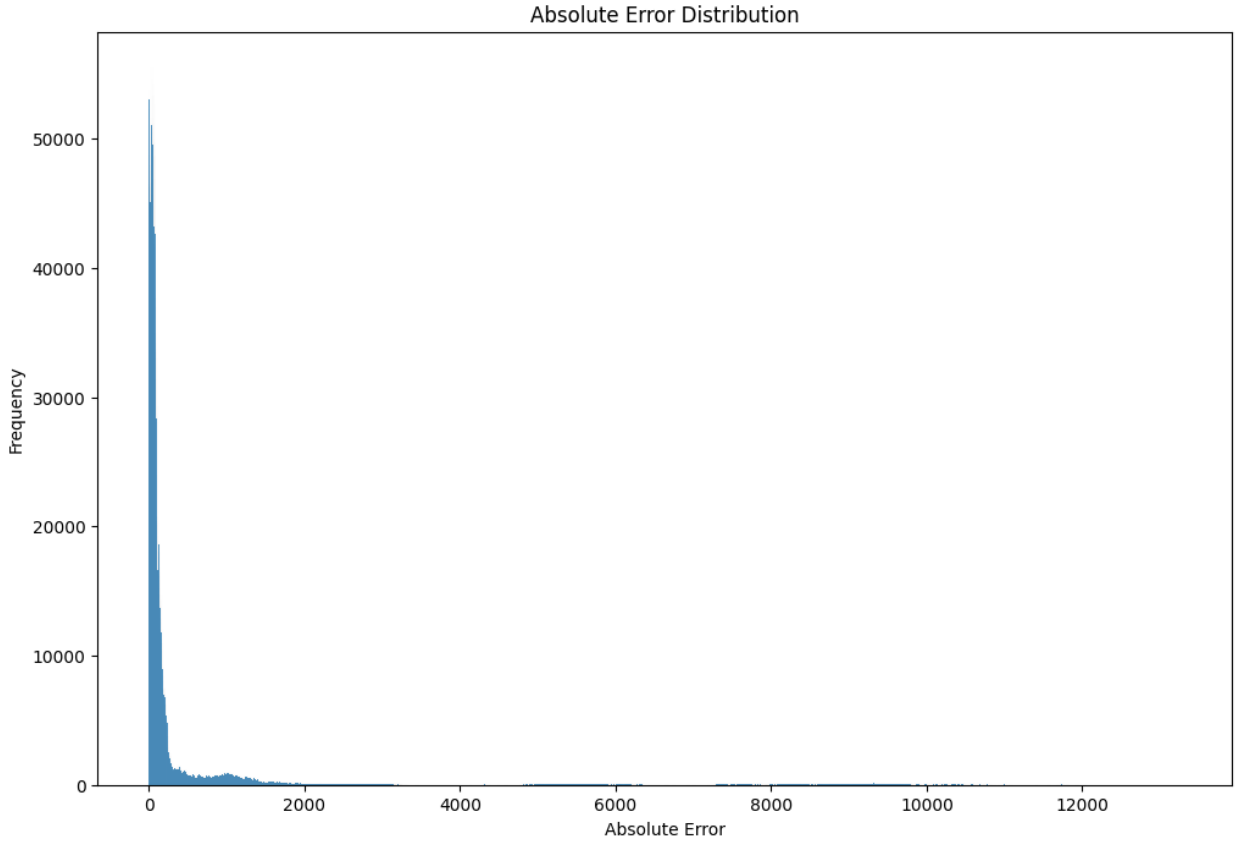
3) Here I created a for loop to produce a line plot for each stock. See here for source code.



This line plot shows the comparison between the actual closing price and the predicted closing price. The blue line is the actual closing price and the orange is the predicted closing price.

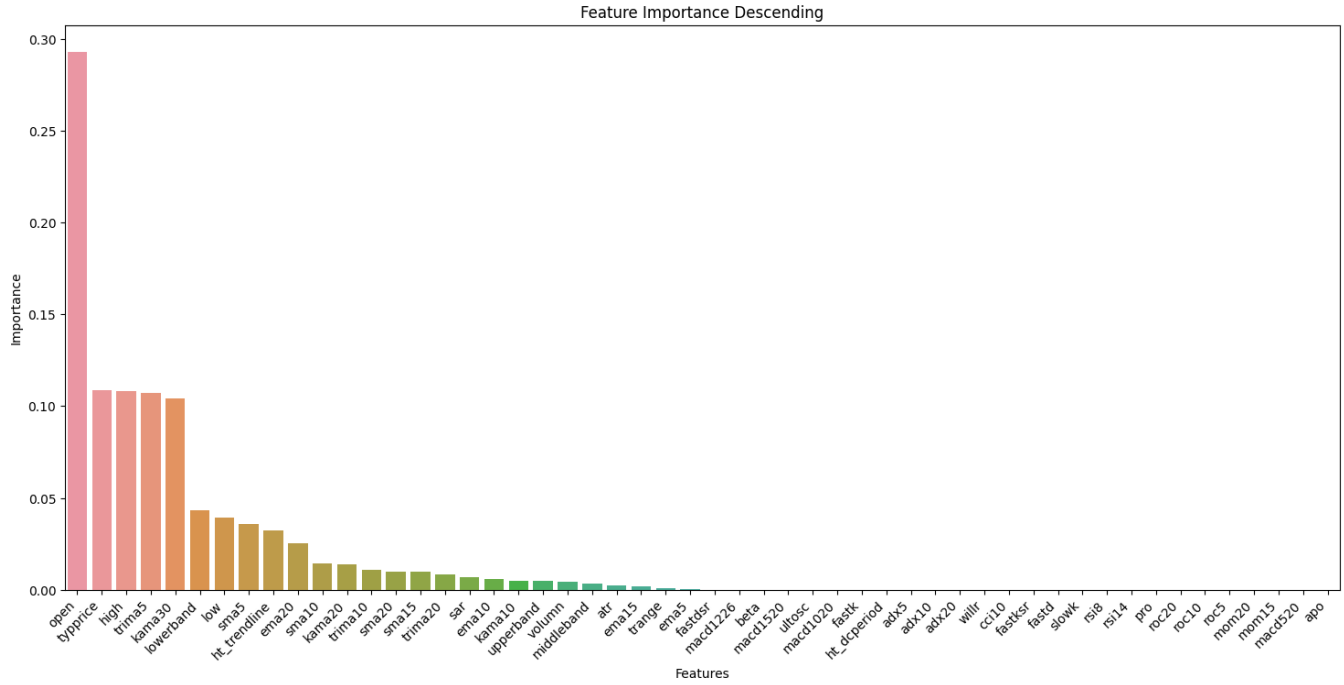
4)

sns.histplot(x="abs_error", data=results_df).set(title="Absolute Error Distribution")



This graph shows the absolute error distribution. The error is the difference between actual and prediction. Most of the errors are within 0 to 500. The graph stretched out because of the outliers.

5)
sns.barplot(x="Feature", y="Importance", data=features_df).set(title="Feature Importance Descending")



This bar plot shows the feature importance value for each feature in descending order. Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node.
The top 10 features respectively are:
open: Open price of the index on a particular day
typprice: Typical price
high: High price of the index on a particular day
trima5: Triangular Moving Average of 5 close price
kama30: Kaufman Adaptive Moving Average of 30 close price
lowerband: Lowerband of Bollinger band
low: Low price of the index on a particular day
sma5: Simple moving average for 5 close price
ht_trendline: Hilbert Transform - Instantaneous Trendline
ema20: Exponential Moving Average for 20 close price

See appendix D for the source code for all 4 visuals.

Summary and Conclusions

In conclusion, from this machine learning project, I learned the importance of big data and the fundamentals of AWS. I also learned how to build a complete machine learning pipeline using AWS.

Summary of the pipeline:

1) Used Amazon EC2 and S3 to download, load, and zip CSV files from the 33GB Kaggle dataset.
2) Merged CSV files and then performed descriptive statistics using EC2.
3) Used AWS EMR to load data and perform data restructuring.
4) Performed feature engineering using VectorAssembler and pipeline. Then fit and transform the data using the pipeline.
5) Split the data into 70% training and 30% testing by dates.
6) Fitted and transformed the data by using a random forest regressor and evaluated performance using r-squared, RMSE, and MAE.
7) Run the model again using only the top 10 features by feature importance and evaluate performance.
8) Exported results to S3.
9) Created plots/graphs to visualize data and prediction results.

After completing this project, I found the top 3 technical indicators that were most helpful in predicting the closing price are the Typical Price, Triangular Moving Average of 5 close price, and Kaufman Adaptive Moving Average of 30 close price. The Typical Price indicator provides a simple, single-line plot of the day's average price. The Triangular Moving Average of 5 close prices shows the average price of the stock over 5 data points (it's similar to simple moving average). The Kaufman's Adaptive Moving Average is a moving average designed to account for market noise or volatility. From these 3, we could see that using the closing price average seems to be most effective.

From visualization 3, we can see the model is having a hard time predicting the price even though it can follow the general trend. I knew the stock market is unpredictable and has a lot of noise and this confirms it.

Future improvements:

1) Use model tuning to find the best parameters.
2) Remove outliers. Even though the random forest algorithm is not sensitive to outliers, it will make visualizations hard to interpret because it will be skewed/stretched out.

Appendix: Source Code

This appendix includes all the source code I used for this project.

A) Data Acquisition

A-1. Unzip the zip file in S3.

```
# import libraries
import zipfile
import boto3
from io import BytesIO

bucket="cis4130-project-jakeli"
zipfile_to_unzip="x"
s3_client = boto3.client('s3', use_ssl=False)
s3_resource = boto3.resource('s3')

zip_obj = s3_resource.Object(bucket_name=bucket, key=zipfile_to_unzip)
buffer = BytesIO(zip_obj.get()["Body"].read())
z = zipfile.ZipFile(buffer)
# loop through all of the files contained in the Zip archive
for filename in z.namelist():
    print('Working on ' + filename)
    # unzip the file and write it back to S3 in the same bucket
    s3_resource.meta.client.upload_fileobj(z.open(filename),Bucket=bucket,Key=f'{filename}')
```

B) Descriptive Statistics

B-1. Merge CSV files.

```
# import libraries
import s3fs
import pandas as pd
import boto3

client = boto3.client('s3')
bucket = 'cis4130-project-jakeli'
resp = client.list_objects_v2(Bucket = bucket)

# get stock names from files names
stock_names = []
for i in resp['Contents']:
        if ('.csv' in i['Key']):
                file_name = i['Key']
```

```
            stock_name = file_name.split('_', 1)[0]
            stock_names.append(stock_name)

# merge CSV files
df = pd.DataFrame()
for i in stock_names:
        path = "s3://" + bucket + "/" + i + "_with_indicators_.csv"
        df_temp = pd.read_csv(path)
        df_temp['stock'] = i
        cols = df_temp.columns.tolist()
        cols = cols[-1:] + cols[:-1]
        df_temp = df_temp[cols]
        df = df.append(df_temp)

# export dataframe to S3
df.to_csv('s3://cis4130-project-jakeli/all_stocks.csv', index=False)
```

B-2. Descriptive Statistics on the dataframe

```
# number of rows and columns
df.shape
# column names
df.columns
# sum of null values
df.isnull().sum().sum()
# sum of duplicated rows
df.duplicated().sum().sum()
# earliest and latest dates
min(df['date'])
max(df['date'])
# dataframe information
df.info()
# generate statistics summary for open, high, low, close, and volume column
round(df[["open", "high", "low", "close", "volume"]].describe())
```

C) Coding and Modeling

```
sc.setLogLevel("ERROR")
# import libraries
import pandas as pd
import numpy as np
from pyspark.sql.types import *
```

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler, StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from itertools import chain

""" Run below once
# define schema
schema = StructType([
        StructField("stock", StringType()),
        StructField("date", TimestampType()),
        StructField("open", DoubleType()),
        StructField("high", DoubleType()),
        StructField("low", DoubleType()),
        StructField("close", DoubleType()),
        StructField("volumn", DoubleType()),
        StructField("sma5", DoubleType()),
        StructField("sma10", DoubleType()),
        StructField("sma15", DoubleType()),
        StructField("sma20", DoubleType()),
        StructField("ema5", DoubleType()),
        StructField("ema10", DoubleType()),
        StructField("ema15", DoubleType()),
        StructField("ema20", DoubleType()),
        StructField("upperband", DoubleType()),
        StructField("middleband", DoubleType()),
        StructField("lowerband", DoubleType()),
        StructField("HT_Trendline", DoubleType()),
        StructField("KAMA10", DoubleType()),
        StructField("KAMA20", DoubleType()),
        StructField("KAMA30", DoubleType()),
        StructField("SAR", DoubleType()),
        StructField("TRIMA5", DoubleType()),
        StructField("TRIMA10", DoubleType()),
        StructField("TRIMA20", DoubleType()),
        StructField("ADX5", DoubleType()),
        StructField("ADX10", DoubleType()),
        StructField("ADX20", DoubleType()),
```

```
        StructField("APO", DoubleType()),
        StructField("CCI5", DoubleType()),
        StructField("CCI10", DoubleType()),
        StructField("CCI15", DoubleType()),
        StructField("macd510", DoubleType()),
        StructField("macd520", DoubleType()),
        StructField("macd1020", DoubleType()),
        StructField("macd1520", DoubleType()),
        StructField("macd1226", DoubleType()),
        StructField("MOM10", DoubleType()),
        StructField("MOM15", DoubleType()),
        StructField("MOM20", DoubleType()),
        StructField("ROC5", DoubleType()),
        StructField("ROC10", DoubleType()),
        StructField("ROC20", DoubleType()),
        StructField("PRO", DoubleType()),
        StructField("RSI14", DoubleType()),
        StructField("RSI8", DoubleType()),
        StructField("slowk", DoubleType()),
        StructField("slowd", DoubleType()),
        StructField("fastk", DoubleType()),
        StructField("fastd", DoubleType()),
        StructField("fastksr", DoubleType()),
        StructField("fastdsr", DoubleType()),
        StructField("ULTOSC", DoubleType()),
        StructField("WILLR", DoubleType()),
        StructField("ATR", DoubleType()),
        StructField("Trange", DoubleType()),
        StructField("TYPPRICE", DoubleType()),
        StructField("HT_DCPERIOD", DoubleType()),
        StructField("BETA", DoubleType())])
# read csv from S3
df = spark.read.csv('s3a://cis4130-project-jakeli/all_stocks.csv', header=True, schema=schema)
df.printSchema()
# change all columns names to lowercase
for col in df.columns:
    df = df.withColumnRenamed(col, col.lower())
# export dataframe as parquet file
df.write.parquet("s3a://cis4130-project-jakeli/all_stocks.parquet")
"""
```

```
### Start from here 1
df = spark.read.parquet('s3a://cis4130-project-jakeli/all_stocks.parquet')

# build pipeline
selected_features = df.columns
del selected_features[0:2]
selected_features.remove("close")
assembler = VectorAssembler(inputCols=selected_features, outputCol="features")
pipeline = Pipeline(stages=[assembler])
transformed_df = pipeline.fit(df).transform(df)

transformed_df = transformed_df["stock", "date", "close", "features"]
#transformed_df.write.mode("overwrite").parquet("s3a://cis4130-project-jakeli/all_stocks_transf
ormed.parquet")

### Start from here 2
#transformed_df =
spark.read.parquet("s3a://cis4130-project-jakeli/all_stocks_transformed.parquet")

# random forest regressor model
max_date = transformed_df.agg({"date": "max"}).collect()[0][0]
min_date = transformed_df.agg({"date": "min"}).collect()[0][0]
seventy_percent_threshold = (max_date - min_date)*0.7
split_date = min_date + seventy_percent_threshold
train_df = transformed_df.where(transformed_df["date"] < split_date)
test_df = transformed_df.where(transformed_df["date"] >= split_date)

rf = RandomForestRegressor(labelCol="close", featuresCol="features", seed=2)
model = rf.fit(train_df)
prediction = model.transform(test_df)

# calculate RMSE, R^2, and MAE score
evaluator = RegressionEvaluator(labelCol="close", predictionCol="prediction")
r2 = evaluator.evaluate(prediction, {evaluator.metricName: "r2"})
rmse = evaluator.evaluate(prediction, {evaluator.metricName: "rmse"})
mae = evaluator.evaluate(prediction, {evaluator.metricName: "mae"})
print("The R^2 for using all features is:", round(r2, 2))
print("The RMSE for using all features is:", round(rmse, 2))
print("The MAE for using all features is:", round(mae, 2))
```

```
rf_results = prediction.select("stock", "date", "close", "prediction")
#rf_results.write.mode("overwrite").csv('s3a://cis4130-project-jakeli/rf_results.csv')

# run model again using only the top 10 features
attrs = sorted((attr["idx"], attr["name"]) for attr in
(chain(*transformed_df.schema["features"].metadata["ml_attr"]["attrs"].values())))
feature_list = [(name, round(model.featureImportances[idx],5)) for idx, name in attrs if
model.featureImportances[idx]]
features_df = pd.DataFrame(feature_list, columns=["feature",
"importance"]).sort_values(by="importance", ascending=False)

top_10_features = features_df["feature"][0:10].tolist()
df = spark.read.parquet('s3a://cis4130-project-jakeli/all_stocks.parquet')
assembler = VectorAssembler(inputCols=top_10_features, outputCol="features")
pipeline = Pipeline(stages=[assembler])
transformed_df2 = pipeline.fit(df).transform(df)

transformed_df2 = transformed_df["stock", "date", "close", "features"]
#transformed_df2.write.mode("overwrite").parquet("s3a://cis4130-project-jakeli/all_stocks_trans
formed_2.parquet")

max_date = transformed_df2.agg({"date": "max"}).collect()[0][0]
min_date = transformed_df2.agg({"date": "min"}).collect()[0][0]
seventy_percent_threshold = (max_date - min_date)*0.7
split_date = min_date + seventy_percent_threshold
print(split_date)
train_df2 = transformed_df2.where(transformed_df2["date"] < split_date)
test_df2 = transformed_df2.where(transformed_df2["date"] >= split_date)

rf = RandomForestRegressor(labelCol="close", featuresCol="features", seed=2)
model = rf.fit(train_df2)
prediction2 = model.transform(test_df2)

# calculate RMSE, R^2, and MAE score
evaluator = RegressionEvaluator(labelCol="close", predictionCol="prediction")
r2 = evaluator.evaluate(prediction, {evaluator.metricName: "r2"})
rmse = evaluator.evaluate(prediction, {evaluator.metricName: "rmse"})
mae = evaluator.evaluate(prediction, {evaluator.metricName: "mae"})
print("The R^2 for using top 10 features only is:", round(r2, 2))
```

```
print("The RMSE for using top 10 features only is:", round(rmse, 2))
print("The MAE for using top 10 features only is:", round(mae, 2))

rf_results2 = prediction2.select("stock", "date", "close", "prediction")
#rf_results2.write.mode("overwrite").csv('s3a://cis4130-project-jakeli/rf_results_2.csv')

D) Visualizing Results
# import libraries
import matplotlib.pyplot as plt
import seaborn as sns
import io
import s3fs

# visual 1: overall trends time series
df = spark.read.parquet('s3a://cis4130-project-jakeli/all_stocks.parquet')
df = df.filter(df.date >= "2020-01-01 00:00:00")
df = df.select("stock", "date", "close").toPandas()

plt.figure(figsize=(15, 10))
sns.lineplot(x="date", y="close", hue="stock", data=df, legend=False).set(title='Overview of 49
Stock Trends')
plt.xlabel("Date")
plt.ylabel("Closing Price")

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
s3 = s3fs.S3FileSystem(anon=False)
with s3.open("s3a://cis4130-project-jakeli/visuals/stocks_time_series.png", 'wb') as f:
        f.write(img_data.getbuffer())

# Visual 2: Closing Price Distribution
plt.figure(figsize=(12, 8))
sns.histplot(x="close", data=df).set(title="Overall Closing Price Distribution")
plt.xlabel("Closing Price")
plt.ylabel("Frequency")

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
```

```
s3 = s3fs.S3FileSystem(anon=False)
with s3.open("s3a://cis4130-project-jakeli/visuals/stocks_price_distribution.png", 'wb') as f:
        f.write(img_data.getbuffer())

# Visual 3: Actual VS Prediction (49 stocks)
results_df = spark.read.parquet("s3a://cis4130-project-jakeli/rf_results.parquet").toPandas()
stocks = results_df["stock"].unique()

for x in stocks:
        stock_df = results_df[results_df["stock"]==x]
        plot_title = "Actual Closing Price VS Prediction" + " (" + x + ")"
        plt.figure(figsize=(15, 10))
        sns.lineplot(x="date", y="close", data=stock_df)
        sns.lineplot(x="date", y="prediction", data=stock_df).set(title=plot_title)
        plt.xlabel("Date")
        plt.ylabel("Stock Price")
        path_name = "s3a://cis4130-project-jakeli/visuals/actual_vs_prediction" + "_" + x +
".png"
        img_data = io.BytesIO()
        plt.savefig(img_data, format='png', bbox_inches='tight')
        img_data.seek(0)
        s3 = s3fs.S3FileSystem(anon=False)
        with s3.open(path_name, 'wb') as f:
                f.write(img_data.getbuffer())

# Visual 4: Absolute Error Distribution
results_df = spark.read.parquet("s3a://cis4130-project-jakeli/rf_results.parquet").toPandas()
results_df["abs_error"] = abs(results_df["close"]-results_df["prediction"])

plt.figure(figsize=(12, 8))
sns.histplot(x="abs_error", data=results_df).set(title="Absolute Error Distribution")
plt.xlabel("Absolute Error")
plt.ylabel("Frequency")

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
s3 = s3fs.S3FileSystem(anon=False)
with s3.open("s3a://cis4130-project-jakeli/visuals/abs_error_distribution.png", 'wb') as f:
        f.write(img_data.getbuffer())
```

```
# Visual 5: Feature importance (Fit and transform using random forest regressor first then run the follow)

plt.figure(figsize=(18, 8))
sns.barplot(x="Feature", y="Importance", data=features_df).set(title="Feature Importance Descending")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.xticks(rotation=45, horizontalalignment='right')

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
s3 = s3fs.S3FileSystem(anon=False)
with s3.open("s3a://cis4130-project-jakeli/visuals/feature_importance.png", 'wb') as f:
        f.write(img_data.getbuffer())
```