Fil Rouge Project

JPX Tokyo Stock Exchange Prediction

Explore the Tokyo market with your data science skills

Weizhe **XIE**, Tom **OLEJNICZAK**, Dian **CHEN**, Andres **POSADA SANCHEZ COBIZA**

Summary of our analysis:

- Introduction: **Importing** and **exploring** the data
- Step 1: Data **cleaning** and **manipulation**
- Step 2: Building our **Machine Learning Models**
- **Conclusions** of our analysis

Importing and exploring the data

- Importing relevant libraries
- Importing CSV file, including the training data stock prices files
- Exploring the Data

```python
### importing relevant libraries ###
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing all relevant libraries,
Including important ones for
financial forecasting

Importing our train file & our
supplementary file, which will
be used in our financial
forecasting model

```python
#### Importing the stock_list file (train+supplemental)####
df=pd.read_csv("/Users/theresa/Desktop/课件/Fil_Rouge/train_files/train_stock_prices.csv")
supplemental=pd.read_csv('/Users/theresa/Desktop/课件/Fil_Rouge/train_files/supplemental_stock_prices.csv')
df=df.append(supplemental)
df.head()
```

Output:

	RowId	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
0	20170104_1301	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	NaN	False	0.000730
1	20170104_1332	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	NaN	False	0.012324
2	20170104_1333	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	NaN	False	0.006154
3	20170104_1376	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	NaN	False	0.011053
4	20170104_1377	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	NaN	False	0.003026

Content from our table:

1.Exploring the data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2602412 entries, 0 to 269880
Data columns (total 12 columns):
 #   Column            Dtype
---  ------            -----
 0   RowId             object
 1   Date              datetime64[ns]
 2   SecuritiesCode    int64
 3   Open              float64
 4   High              float64
 5   Low               float64
 6   Close             float64
 7   Volume            int64
 8   AdjustmentFactor  float64
 9   ExpectedDividend  float64
 10  SupervisionFlag   bool
 11  Target            float64
dtypes: bool(1), datetime64[ns](1), float64(7), int64(2), object(1)
memory usage: 240.7+ MB
```

SecuritiesCode: Local securities code of the stock

Close: last traded price on a day

Volume: number of traded stocks on a day

Target: Change ratio of adjusted closing price between t+2 and t+1 where t+0 is TradeDate

Information about our columns content:

```
df.shape
```

```
(2602412, 12)
```

```
df.isnull().sum()
```

```
RowId                    0
Date                     0
SecuritiesCode           0
Open                  8426
High                  8426
Low                   8426
Close                 8426
Volume                   0
AdjustmentFactor         0
ExpectedDividend   2581536
SupervisionFlag          0
Target                 246
dtype: int64
```

The table has **2.602.412 rows**, & **12 columns**.

Columns subject to cleaning:

- Open
- High
- Low
- Close
- Target

Columns subject to cleaning

```
df.isnull().sum()
```

```
RowId                     0
Date                      0
SecuritiesCode            0
Open                   8426
High                   8426
Low                    8426
Close                  8426
Volume                    0
AdjustmentFactor          0
ExpectedDividend    2581536
SupervisionFlag           0
Target                  246
dtype: int64
```

Columns subject to cleaning:

- Open
- High
- Low
- Close
- Target

Cleaning our Data Set

Note: Many null values are linked to a **system error, on the 2020-10-01**.

```
df = pd.DataFrame(df)

df['Date'] = pd.to_datetime(df['Date'])

df[df['Close'].isnull()].groupby('Date').size().sort_values(ascending=False)
```

```
Date
2020-10-01    1988
2022-04-15      16
2017-03-16      15
2019-04-04      14
2019-10-09      14
              ...
2018-12-27       1
2017-11-06       1
2017-11-02       1
2021-02-09       1
2021-05-10       1
Length: 1310, dtype: int64
```

```
df[(df["Date"] == "2020-10-01") & df.isnull().any(axis=1)]
```

	RowId	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
1755040	20201001_1301	2020-10-01	1301	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.029208
1755041	20201001_1332	2020-10-01	1332	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.027211
1755042	20201001_1333	2020-10-01	1333	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.027695
1755043	20201001_1375	2020-10-01	1375	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.023833
1755044	20201001_1376	2020-10-01	1376	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.022152
...	...	...	...	...	...	...	...	...	...	...	...	...
1757023	20201001_9990	2020-10-01	9990	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.023297
1757024	20201001_9991	2020-10-01	9991	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.041621
1757025	20201001_9993	2020-10-01	9993	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.034006
1757026	20201001_9994	2020-10-01	9994	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.025047
1757027	20201001_9997	2020-10-01	9997	NaN	NaN	NaN	NaN	0	1.0	NaN	False	0.004301

1988 rows × 12 columns

thejapantimes

Tokyo trading halted due to hardware failure

A man takes a photo of a blank electronic stock board at a securities firm in Tokyo on Thursday. Trading on the Tokyo Stock Exchange was suspended because of a problem with the system for relaying market information. | AP

BY OSAMU TSUKIMORI
STAFF WRITER

SHARE Oct 1, 2020

The Tokyo Stock Exchange halted trading in all listed stocks for a full day Thursday due to the biggest technical glitch it has experienced since it introduced a computer system in 1999.

There are 1337 different dates per stock, with the number of total different values variating between stocks.

```
df["Date"].nunique()
```

```
1337
```

```
df.groupby('SecuritiesCode').size().sort_values(ascending=True).head(140)
```

```
SecuritiesCode
4169     367
7342     368
4168     368
7358     369
4167     370
        ...
9519    1302
4699    1316
2729    1320
6470    1337
6471    1337
Length: 140, dtype: int64
```

There are around 140 stocks doesn't have full 1337 days data.

```
df.head()
```

	RowId	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
0	20170104_1301	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	NaN	False	0.000730
1	20170104_1332	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	NaN	False	0.012324
2	20170104_1333	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	NaN	False	0.006154
3	20170104_1376	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	NaN	False	0.011053
4	20170104_1377	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	NaN	False	0.003026

Step 1: Data cleaning and manipulation

- **Data Cleaning**
- Choosing **one random stock**
- **Data visualisation,** using that stock

Prepare our Data Set

For Close: **Using the “Adjustment Factor”, to adjust the “Close” column**

Why?

Modify the historical stock prices for events such as **stock splits**, **dividends, and rights offerings**.
These events can significantly alter a stock's price, making historical comparisons misleading

```
## use the "AdjustmentFactor" to adjust the "Close"
df["Close"]=df["Close"]*df["AdjustmentFactor"]
df=df.drop(columns=['RowId','ExpectedDividend','SupervisionFlag',"AdjustmentFactor"],axis=1).reset_index(drop=True)
```

```
## Sorting our data set by Security code, then by date
df.sort_values(by=['SecuritiesCode', 'Date'], inplace=True)
df.reset_index(drop=True, inplace=True)
df
```

Prepare our Data Set

	Date	SecuritiesCode	Open	High	Low	Close	Volume	Target
0	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	0.000730
1	2017-01-05	1301	2743.0	2747.0	2735.0	2738.0	17900	0.002920
2	2017-01-06	1301	2734.0	2744.0	2720.0	2740.0	19900	-0.001092
3	2017-01-10	1301	2745.0	2754.0	2735.0	2748.0	24200	-0.005100
4	2017-01-11	1301	2748.0	2752.0	2737.0	2745.0	9300	-0.003295
...	...	...	...	...	...	...	...	...
2602407	2022-06-20	9997	693.0	697.0	683.0	687.0	122600	0.001416
2602408	2022-06-21	9997	692.0	709.0	692.0	706.0	204800	0.000000
2602409	2022-06-22	9997	706.0	716.0	703.0	707.0	150200	0.016973
2602410	2022-06-23	9997	704.0	713.0	704.0	707.0	114700	0.013908
2602411	2022-06-24	9997	710.0	725.0	710.0	719.0	139600	0.015089

Sorting our data set by **Security code**, then by **Date.**

```
df.sort_values(by=['SecuritiesCode', 'Date'], inplace=True)

df.reset_index(drop=True, inplace=True)
```

Much more structure!!!

Cleaning our Data Set

For Open, High, Low, closed:

Replace NaN values with the average of the previous day and next day non-NaN values

-> Using then mean of ffill()+bfill() method.

To make sure we cleaning by each stock,{For loop}

```
## fill the null value of each 'SecuritiesCode'
securaty_code=df["SecuritiesCode"].unique().tolist()
processed_dfs = []
for s in securaty_code:
  sdf=df[df['SecuritiesCode']==s]
  sdf['Open'].fillna((sdf['Open'].ffill() + sdf['Open'].bfill()) / 2, inplace=True)
  sdf['High'].fillna((sdf['High'].ffill() + sdf['High'].bfill()) / 2, inplace=True)
  sdf['Low'].fillna((sdf['Low'].ffill() + sdf['Low'].bfill()) / 2, inplace=True)
  sdf['Close'].fillna((sdf['Close'].ffill() + sdf['Close'].bfill()) / 2, inplace=True)
  sdf["future2"]=sdf["Close"].shift(-2)
  sdf["future1"]=sdf["Close"].shift(-1)
  sdf["Target"].fillna((sdf["future2"]-sdf["future1"])/sdf["future1"],inplace=True)
  sdf.fillna(0, inplace=True)
  processed_dfs.append(sdf)
df = pd.concat(processed_dfs, ignore_index=True)
```

Generating a random stock for our analysis

```
### generate random stock for the analysis
import random
random.seed(69)
random_security_code = random.sample(securaty_code,1)
random_security_code
```

```
[7550]
```

In [22]:

```
## drop the column of "SecuritiesCode"
df_rd=df_rd.drop(["SecuritiesCode"],axis=1)
df_rd
```

Out[22]:

	Date	Open	High	Low	Close	Volume	Target	future2	future1
1809894	2017-01-04	1943.0	1946.0	1932.0	1938.0	383000	0.005685	1946.0	1935.0
1809895	2017-01-05	1935.0	1949.0	1924.0	1935.0	378300	-0.007194	1932.0	1946.0
1809896	2017-01-06	1930.0	1948.0	1928.0	1946.0	363500	0.016046	1963.0	1932.0
1809897	2017-01-10	1932.0	1944.0	1924.0	1932.0	427700	-0.012226	1939.0	1963.0
1809898	2017-01-11	1942.0	1971.0	1940.0	1963.0	473400	-0.002579	1934.0	1939.0
...	...	...	...	...	...	...	...	...	...
1811226	2022-06-20	3175.0	3175.0	3120.0	3140.0	212400	0.012500	3240.0	3200.0
1811227	2022-06-21	3150.0	3215.0	3150.0	3200.0	303300	-0.001543	3235.0	3240.0
1811228	2022-06-22	3235.0	3290.0	3225.0	3240.0	440100	0.004637	3250.0	3235.0
1811229	2022-06-23	3225.0	3270.0	3210.0	3235.0	282100	-0.001538	0.0	3250.0
1811230	2022-06-24	3250.0	3270.0	3235.0	3250.0	250500	0.004622	0.0	0.0

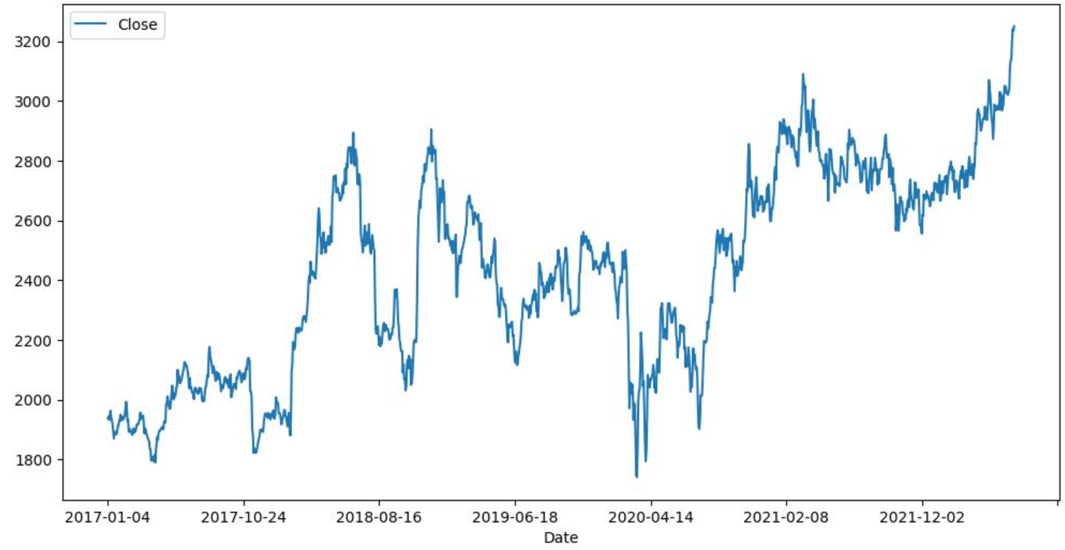
1337 rows × 9 columns

Analysing the trend of our random stock

Displaying the "Close" value of the random stock over time

```
## Displaying the Close value of the random_security_code stock over time
import matplotlib.pyplot as plt
fig,ax=plt.subplots(figsize=(12,6))
df_rd.plot("Date","Close",ax=ax)
```

```
<Axes: xlabel='Date'>
```



Overall Trend:

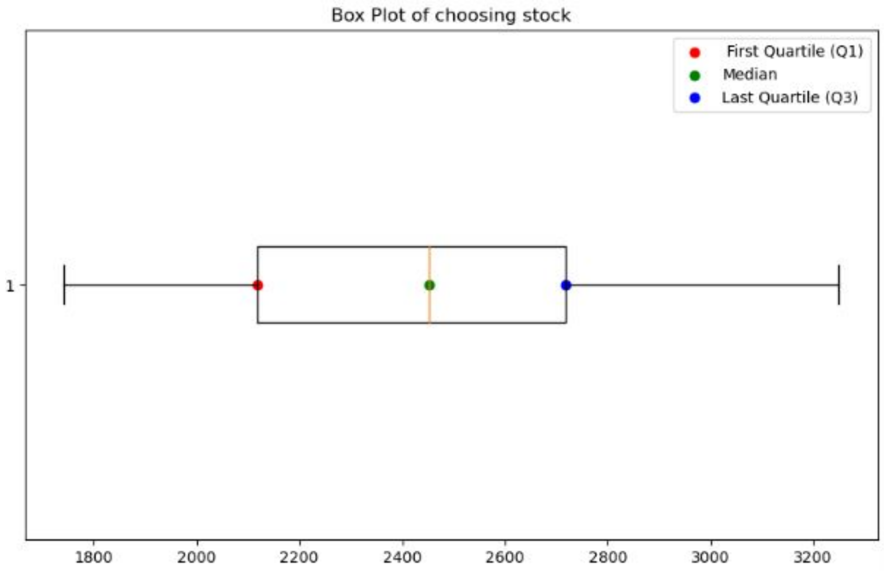
General upward trend

Suggests a positive performance over time

Using Box Plot for “closed”

```python
Q1 = df_rd['Close'].quantile(0.25)
median = df_rd['Close'].quantile(0.5)
Q3 = df_rd['Close'].quantile(0.75)

plt. figure(figsize=(10, 6))
plt. boxplot(df_rd['Close'], vert=False)

plt. scatter (Q1, 1,color='red',label=' First Quartile (Q1)')
plt. scatter (median, 1,color='green', label='Median')
plt.scatter(Q3,1, color='blue', label='Last Quartile (Q3)')

plt.title('Box Plot of choosing stock')
plt. xlabel ( 'Closed Price')
plt. legend()
plt. show()
```

Output:

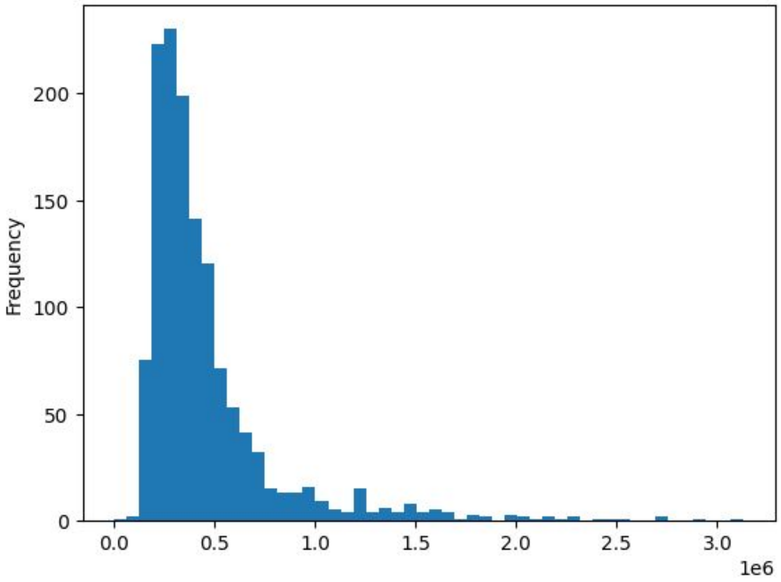


Analysing the trend of our random stock

Analysing the volume over time

```
[ ] ## Displaying the Volume of the random_security_code stock over time
    df_rd["Volume"].plot.hist(bins=50)
```

```
<Axes: ylabel='Frequency'>
```



right-skewed distribution:

low trading volumes are more common than high trading volumes.

Outliers or Extreme Values:

The long tail to the right suggests that while most of the trading volumes are relatively low, there are a few days with exceptionally high trading volumes.

Analysing the trend of our random stock

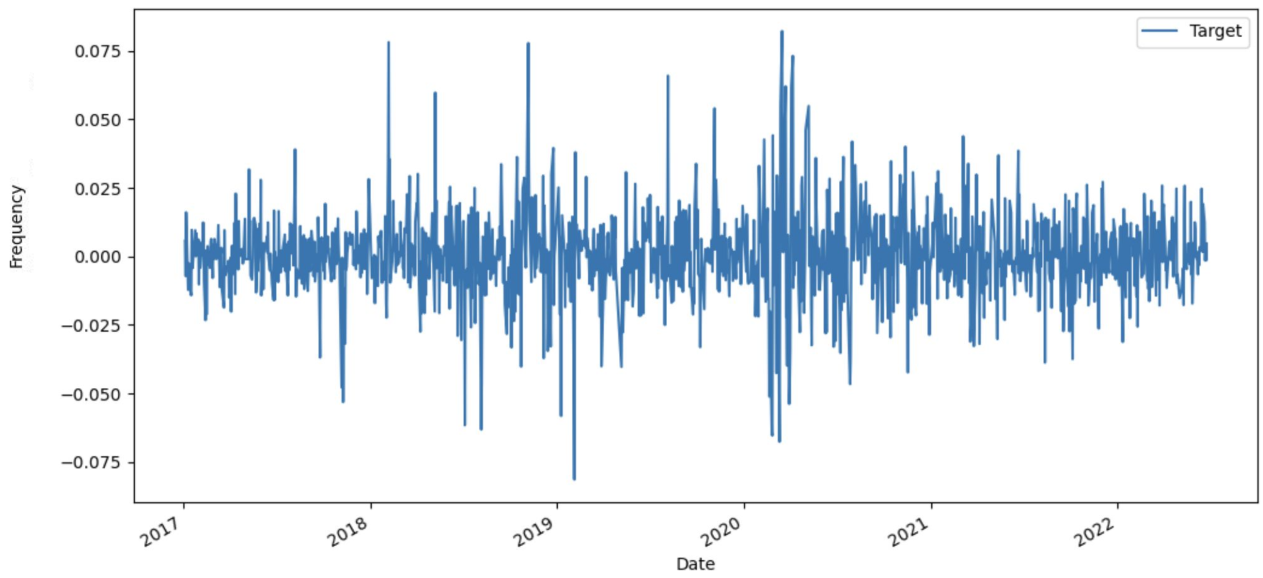
Analysing the target over time

- Short-term volatility
- daily market performance

does not show a long-term trend but rather **fluctuates around the zero line**
No persistent upwards or downward trend

```
fig,ax=plt.subplots(figsize=(12,6))
df_rd.plot("Date","Target",ax=ax)
```

<Axes: xlabel='Date'>



Step 2: Building our Machine Learning Model

- 1. Simple Exponential Smoothing
- 2. ARIMA
- 3. Random Forest Regressor
- 4. LGBMRegressor

Step 2.1: Simple Exponential Smoothing

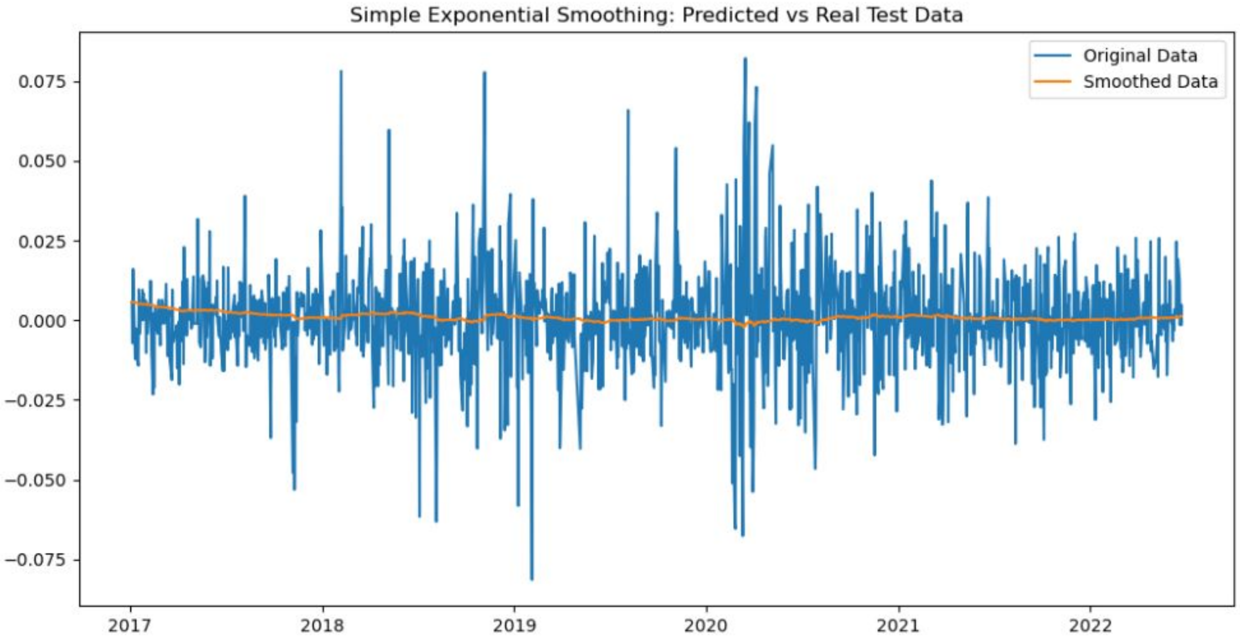
```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
```

```
SES = df_rd["Target"]
model = SimpleExpSmoothing(SES)
fit_model = model.fit()

pred_SES = fit_model.fittedvalues
```

```
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(SES.index, SES, label='Original Data')
ax.plot(pred_SES.index, pred_SES, label='Smoothed Data', linestyle='solid')
ax.set_title(f'Simple Exponential Smoothing: Predicted vs Real Test Data')
ax.legend()
plt.show()
```

Step 2.1: Simple Exponential Smoothing

Output:



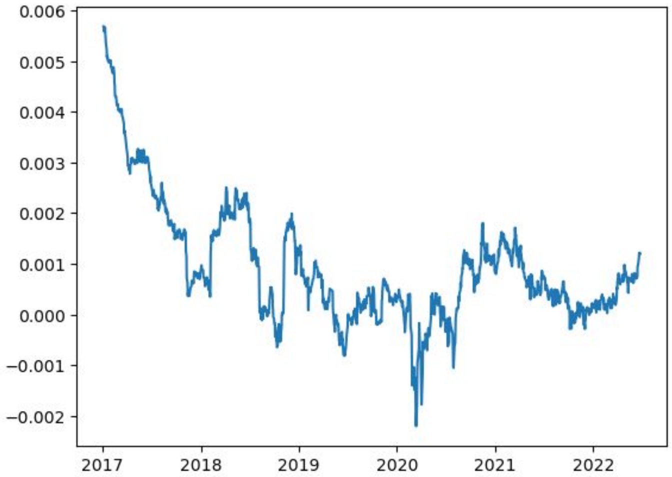
Step 2.1: Simple Exponential Smoothing

```
print("r2_score:",r2_score(SES, pred_SES))
print("Root Mean Squared Error (MSE): ",np.sqrt(mean_squared_error(SES, pred_SES)))
print("Mean Absolute Error (MAE):",np.mean(np.abs(SES - pred_SES)))
```

```
r2_score: -0.011424188849479933
Root Mean Squared Error (MSE):  0.015935269620631663
Mean Absolute Error (MAE): 0.011143197744633486
```

```
plt.plot(pred_SES)
```

```
[<matplotlib.lines.Line2D at 0x2cc83ced0>]
```



Step 2.1: Simple Exponential Smoothing

Conclusion

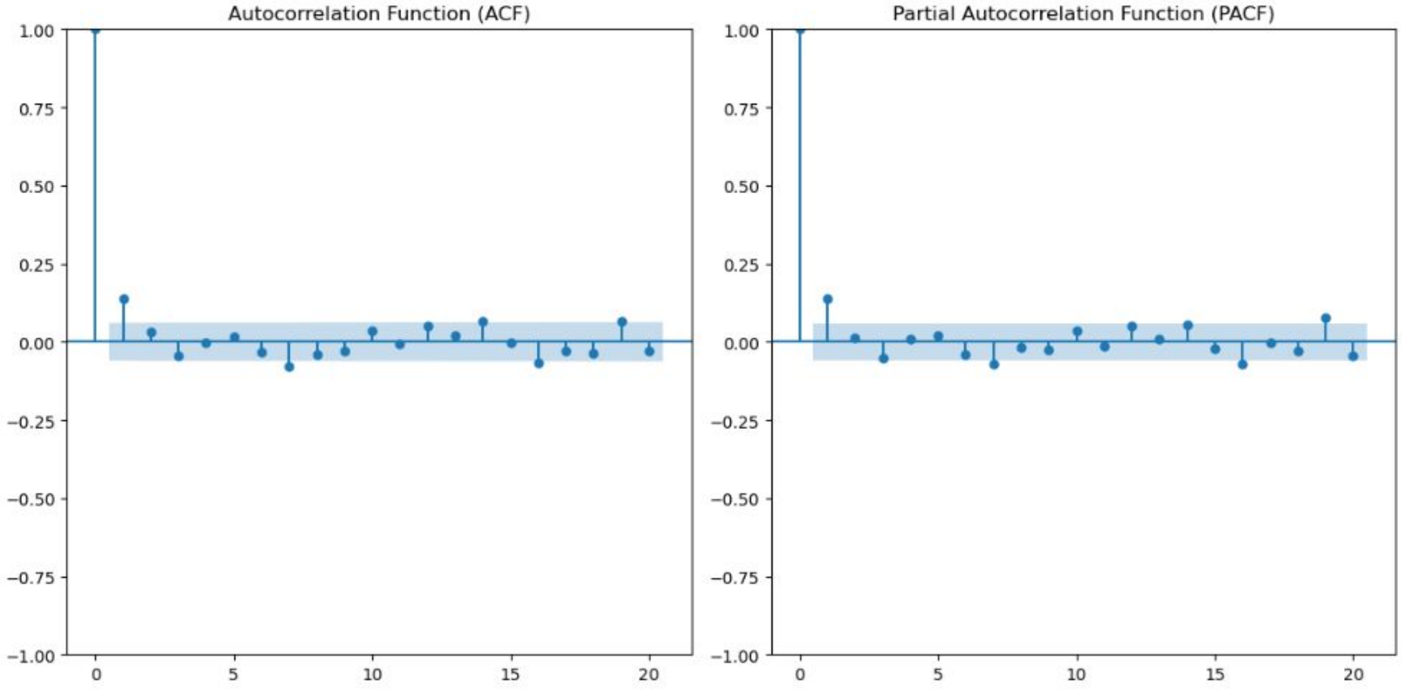
- Model not accurate enough
- Model has not enough information regarding other parameters that might influence the prediction
- A further model is needed to make predictions

Step 2.2: ARIMA

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, pacf

time_series=df_rd["Target"]

# Determine the split point
split_ratio = 0.8  # 80-20 split
split_index = int(len(time_series) * split_ratio)

# Split the data into training and testing sets
train_data = time_series[:split_index]
test_data = time_series[split_index:]

lags = 20
acf_values = acf(train_data, nlags=lags)
pacf_values = pacf(train_data, nlags=lags)

# plot ACF and PACF
plt.figure(figsize=(12, 6))

plt.subplot(121)
plot_acf(train_data, lags=lags, ax=plt.gca())
plt.title('Autocorrelation Function (ACF)')

plt.subplot(122)
plot_pacf(train_data, lags=lags, ax=plt.gca())
plt.title('Partial Autocorrelation Function (PACF)')

plt.tight_layout()
plt.show()
```

Step 2.2: ARIMA

Output:



Step 2.2: ARIMA

```
from statsmodels.tsa.arima.model import ARIMA
from pmdarima.arima import auto_arima

model = auto_arima(train_data, start_p=0, start_q=0,
                   max_p=1, max_q=1,
                   start_P=0, seasonal=False,
                   d=0, D=0, trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True)
```

Output:

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=-5716.600, Time=0.05 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=-5734.859, Time=0.04 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=-5733.733, Time=0.01 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=-5732.945, Time=0.06 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-5733.503, Time=0.02 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0]
Total fit time: 0.186 seconds
```

Step 2.2: ARIMA

Output:

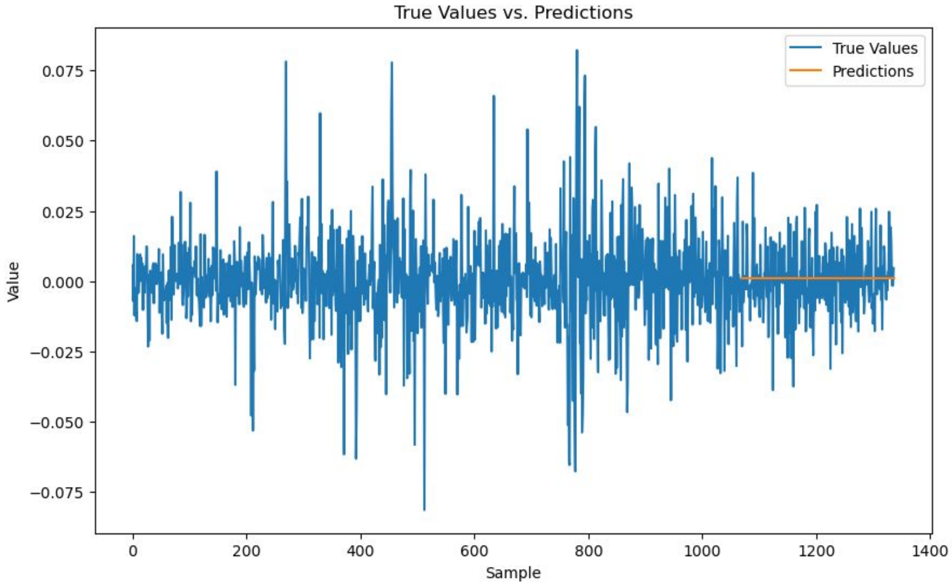
```python
model = ARIMA(train_data, order=(1, 0, 0))
result = model.fit()
print(result.summary())
steps = len(test_data)
forecast = result.predict(start=len(train_data), end=len(train_data) + steps - 1, typ='levels')
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 Target   No. Observations:                 1069
Model:                 ARIMA(1, 0, 0)   Log Likelihood                2869.751
Date:                Mon, 04 Dec 2023   AIC                          -5733.503
Time:                        22:18:18   BIC                          -5718.579
Sample:                             0   HQIC                         -5727.849
                               - 1069
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0005      0.001      0.797      0.426      -0.001       0.002
ar.L1          0.1364      0.022      6.249      0.000       0.094       0.179
sigma2         0.0003   7.74e-06     35.216      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               499.27
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               2.54   Skew:                             0.11
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.34
===================================================================================
```

Step 2.2: ARIMA

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(df_rd['Target'].values, label='True Values')
plt.plot(forecast, label='Predictions')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.legend()
plt.title('True Values vs. Predictions')
plt.show()
```

Output:



Step 2.2: ARIMA

Conclusions:

- **Even when using different values for p, d, and q,** the model still lacks accuracy to be useful
- ARIMA with manual values and AUTOARIMA have no significant difference
- A further method for predicting the stock price is needed

Step 2.3: Random Forest

- **Machine Learning algorithm** that **combines the output of several decision trees** to produce a single result.
- An ensemble method: it **combines multiple results** to get a final one.
- Popular for its:
  - Precision
  - Simplicity
  - Flexibility

Step 2.3: Random Forest

Features engineering

```
## Add some important features
import talib
feature_names = ['Open', 'High', 'Low', 'Close', 'Volume']
for n in [20,50,100]:

    # Create the moving average indicator and divide by Adj_Close
    df_rd['MA' + str(n)] = df_rd['Close'].rolling(window=n).mean()
    # Create the RSI indicator
    df_rd['RSI' + str(n)] = talib.RSI(df_rd['Close'].values,
                                timeperiod=n)
    df_rd['pct_change'+str(n)]=df_rd['Close'].pct_change(n)

    df_rd['volatility'+str(n)]=np.log(df_rd['Close']).diff().rolling(n).std()

    df_rd['EMA'+str(n)] = df_rd['Close'].ewm(span=n, adjust=False).mean()
    # Add rsi and moving average to the feature name list
    feature_names = feature_names  + ['MA' + str(n)]+['pct_change'+str(n)]+['volatility'+str(n)]+['EMA'+str(n)]+['RS
```

Step 2.3: Random Forest

Create important features

```
df_rd["Signal"] = np.where(df_rd["MA20"] > df_rd["MA50"],1,0)
df_rd["Position"] = df_rd["Signal"].diff()
df_rd['Upper_BB'] = df_rd['MA50'] + (df_rd['Close'].rolling(window=50).std() * 2)
df_rd['Lower_BB'] = df_rd['MA50'] - (df_rd['Close'].rolling(window=50).std() * 2)


feature_names=feature_names+["Signal"]+["Position"]+['Upper_BB']+['Lower_BB']
print(feature_names)
```

```
['Open', 'High', 'Low', 'Close', 'Volume', 'MA20', 'pct_change20', 'volatility20', 'EMA20', 'RSI20', 'MA50', 'pct_c
hange50', 'volatility50', 'EMA50', 'RSI50', 'MA100', 'pct_change100', 'volatility100', 'EMA100', 'RSI100', 'Signa
l', 'Position', 'Upper_BB', 'Lower_BB']
```

Step 2.3: Random Forest

import library

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
import numpy as np
```

Create X, Y and split the data

```python
X = df_rd[feature_names]
y = df_rd["Target"]
X.fillna(0,inplace=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 2.3: Random Forest

Use Grid Search to find the best hyperparameters for our model:

```
## use grid search to find our the best hyperparameters
random_grid = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [100, 200, 400, 600, 800, 1000]}


# Initialize the Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Create GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=random_grid, cv=3, scoring='neg_mean_squared_error', verbose=2, n_jobs=-1)

# Perform Grid Search to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_
```

Step 2.3: Random Forest

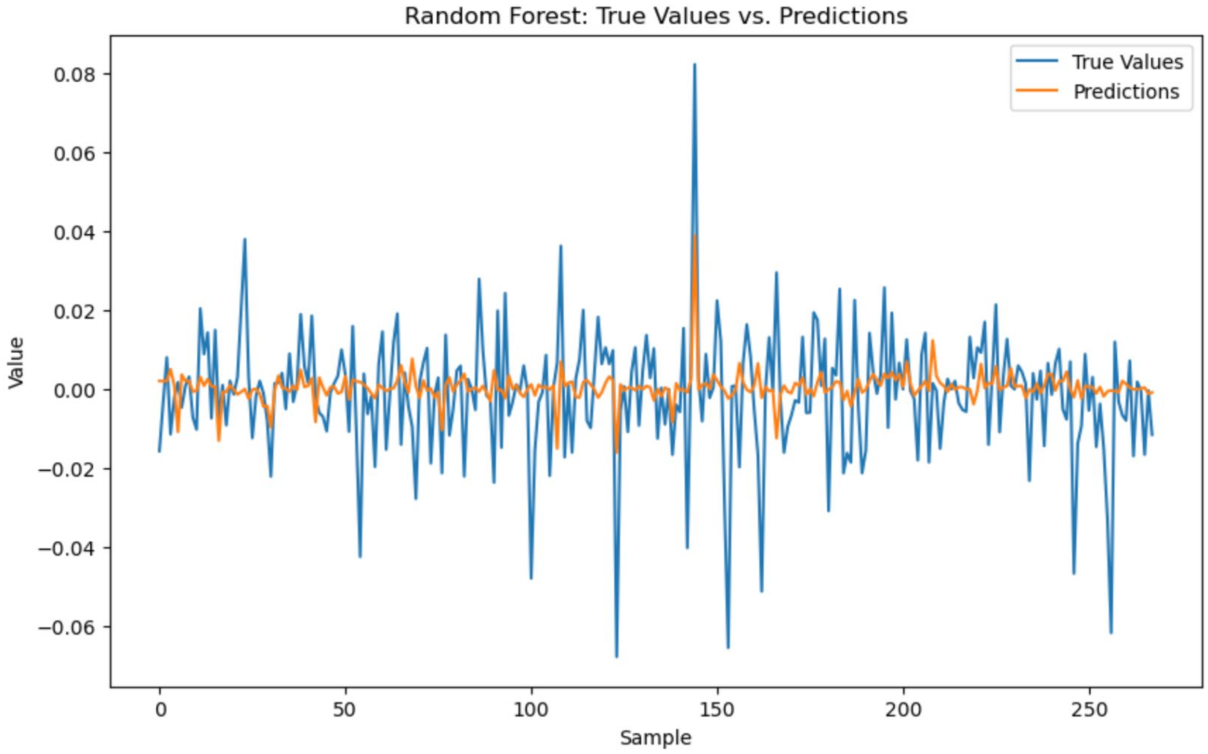
Output (sample):

```
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
1, min_samples_split=5, n_estimators=600; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
1, min_samples_split=10, n_estimators=600; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
1, min_samples_split=10, n_estimators=1000; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
1, min_samples_split=10, n_estimators=1000; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
2, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=
```

```
▼ RandomForestRegressor
RandomForestRegressor(max_depth=10, max_features='sqrt', n_estimators=1000,
                      random_state=42)
```

Step 2.3: Random Forest

```
pred_rf = best_estimator.predict(X_test)
```



Step 2.3: Random Forest

Evaluate the model

```
# Evaluate the model on the test set
print("r2_score:",r2_score(y_test, pred_rf))
print("Root Mean Squared Error (RMSE): ",np.sqrt(mean_squared_error(y_test, pred_rf)))
print("Mean Absolute Error (MAE):",np.mean(np.abs(y_test - pred_rf)))
```

```
r2_score: 0.09770102894576749
Root Mean Squared Error (RMSE):  0.014910576328452609
Mean Absolute Error (MAE): 0.010549270204817005
```

Step 2.3: Random Forest

Another Perspective to Evaluate Prediction:

- Predicting the direction of stock movements rather than precise values
- Investor Decision-making

Model Evaluation Focus : accuracy in predicting positive or negative target value

Transformation: Both the "Target" column and model predictions were transformed into binary format.

- **1**: Denotes positive targets, indicating an increase in stock value.
- **0**: Represents negative targets, signifying either no gain or a decrease in stock value.

Step 2.3: Random Forest

Transform into binary values

```
def to_binary(x):
    return np.where(x >= 0, 1, 0)

y_test_binary = to_binary(y_test)
y_pred_binary = to_binary(y_pred)
```

Accuracy:

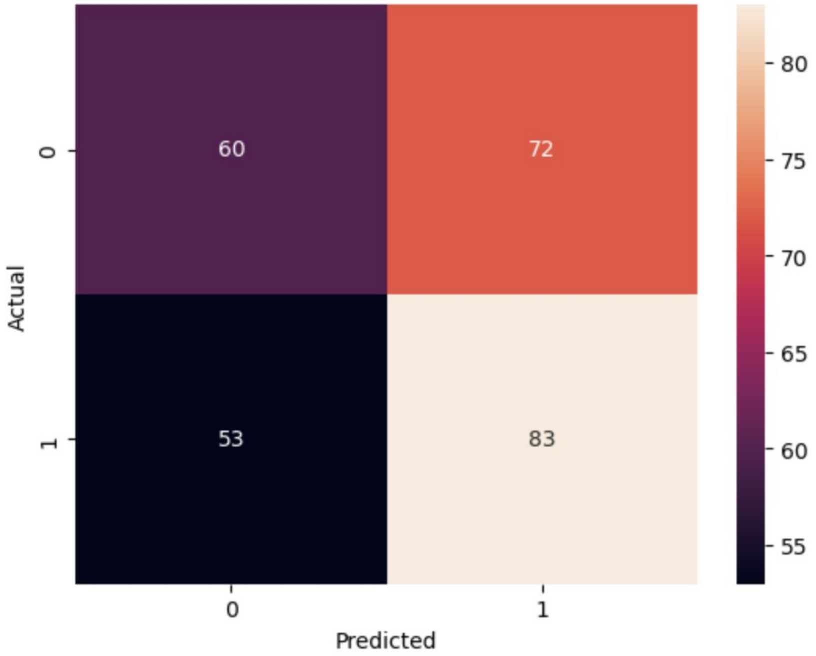
```
from sklearn import metrics
print("Accuracy=",metrics.accuracy_score(y_test_binary,y_pred_binary))
```

```
Accuracy= 0.5335820895522388
```

Step 2.3: Random Forest

```
import seaborn as sn
confusion_matrix=pd.crosstab(y_test_binary,y_pred_binary,rownames=['Actual'],colnames=['Predicted'])
sn.heatmap(confusion_matrix,annot=True)
```

```
<Axes: xlabel='Predicted', ylabel='Actual'>
```



Confusion Matrix

Step 2.4: LightGBM Regressor

- Light Gradient Boosted Machine
- A stochastic gradient boosting ensemble algorithm

```python
from scipy.stats import pearsonr

def feval_pearsonr(preds, train_data):
    labels = train_data.get_label()
    return 'pearsonr', pearsonr(labels, preds)[0], True
```

```python
import lightgbm as lgb
import pandas as pd
from sklearn.model_selection import train_test_split

train_data = lgb.Dataset(X_train, label=y_train)
test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

params_lgb = {
    'learning_rate': 0.005,
    'metric': 'None',
    'objective': 'regression',
    'boosting': 'gbdt',
    'verbosity': 0,
    'n_jobs': -1,
    'force_col_wise': True
}

model = lgb.train(
    params=params_lgb,
    train_set=train_data,
    valid_sets=[train_data, test_data],
    num_boost_round=3000,
    feval=feval_pearsonr,
    callbacks=[
        lgb.early_stopping(stopping_rounds=300, verbose=True),
        lgb.log_evaluation(period=100)
    ]
```

Using **Pearson correlation coefficient** between predictions and actual value to train a LightGBM model

Step 2.4: LGBMRegressor

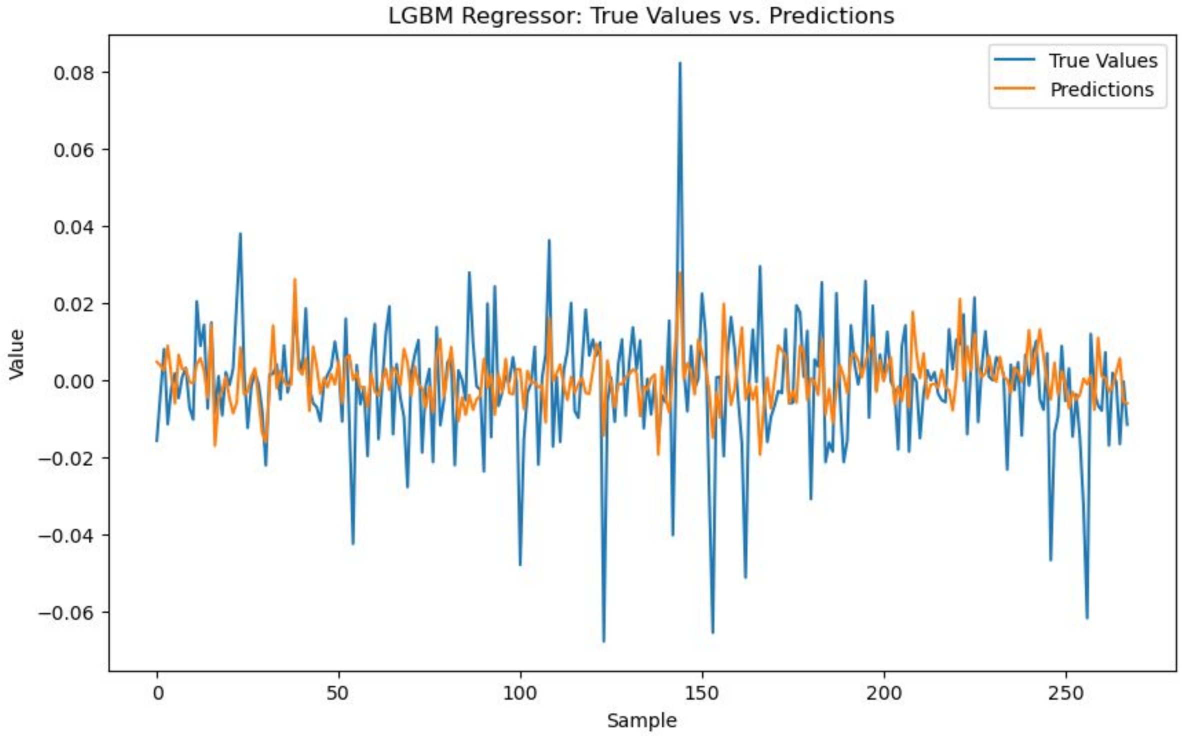
```python
import lightgbm as lgb
import pandas as pd
from sklearn.model_selection import train_test_split

train_data = lgb.Dataset(X_train, label=y_train)
test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

params_lgb = {
    'learning_rate': 0.005,
    'metric': 'None',
    'objective': 'regression',
    'boosting': 'gbdt',
    'verbosity': 0,
    'n_jobs': -1,
    'force_col_wise': True
}

model = lgb.train(
    params=params_lgb,
    train_set=train_data,
    valid_sets=[train_data, test_data],
    num_boost_round=3000,
    feval=feval_pearsonr,
    callbacks=[
        lgb.early_stopping(stopping_rounds=300, verbose=True),
        lgb.log_evaluation(period=100)
    ]
)
```

Output:

```
Training until validation scores don't improve for 300 rounds
[100]	training's pearsonr: 0.605395	valid_1's pearsonr: 0.184312
[200]	training's pearsonr: 0.686504	valid_1's pearsonr: 0.216185
[300]	training's pearsonr: 0.732395	valid_1's pearsonr: 0.229372
[400]	training's pearsonr: 0.772121	valid_1's pearsonr: 0.246295
[500]	training's pearsonr: 0.802949	valid_1's pearsonr: 0.251504
[600]	training's pearsonr: 0.82679	valid_1's pearsonr: 0.260276
[700]	training's pearsonr: 0.844032	valid_1's pearsonr: 0.26799
[800]	training's pearsonr: 0.859387	valid_1's pearsonr: 0.272212
[900]	training's pearsonr: 0.872519	valid_1's pearsonr: 0.275818
[1000]	training's pearsonr: 0.883563	valid_1's pearsonr: 0.274286
[1100]	training's pearsonr: 0.893035	valid_1's pearsonr: 0.27346
[1200]	training's pearsonr: 0.901796	valid_1's pearsonr: 0.274575
Early stopping, best iteration is:
[924]	training's pearsonr: 0.875492	valid_1's pearsonr: 0.2764
```

Step 2.4: LGBMRegressor

```
pred_lgb=model.predict(X_test)
```



Step 2.4: LGBMRegressor

Evaluate the model

```
# Evaluate the model on the test set
print("r2_score:",r2_score(y_test, pred_lgb))
print("Root Mean Squared Error (RMSE): ",np.sqrt(mean_squared_error(y_test, pred_lgb)))
print("Mean Absolute Error (MAE):",np.mean(np.abs(y_test - pred_lgb)))
```

```
r2_score: 0.04550210023491208
Root Mean Squared Error (RMSE):  0.015335808886900032
Mean Absolute Error (MAE): 0.010968749687230744
```

Step 2.4: LGBMRegressor

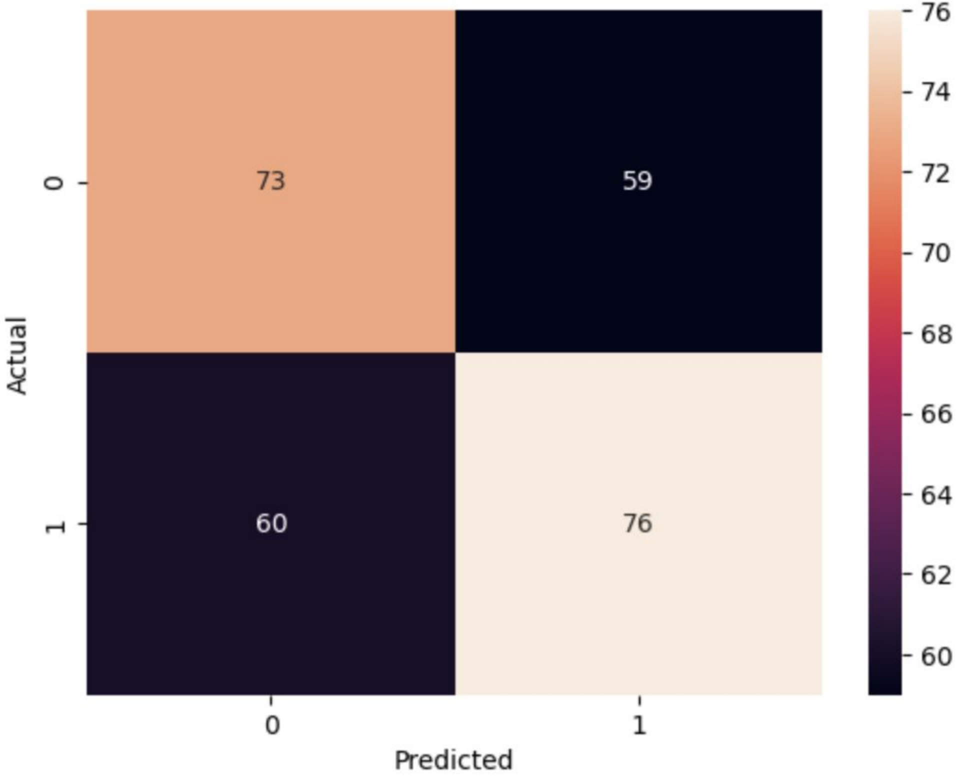
Transform into binary format & Accuracy

```
def to_binary(x):
    return np.where(x >= 0, 1, 0)

y_test_binary = to_binary(y_test)
pred_lgb_binary = to_binary(pred_lgb)

from sklearn import metrics
print("Accuracy=",metrics.accuracy_score(y_test_binary,pred_lgb_binary))
```

```
Accuracy= 0.5559701492537313
```

Step 2.4: LGBMRegressor

Confusion Matrix



Conclusion of our analysis

- Explore and clean the data
- 4 Machine Learning models: **Simple Exponential Smoothing**, **ARIMA**, **Random Forest Regressor**, and **LGBM Regressor**;
- Our goal: **predict** the "**Target**" column
  - **Random Forest Regressor** yielded the **most promising results;**
- Accuracy rate of predicting stock movement **over 50%.**

Thank you !

Weizhe **XIE**, Tom **OLEJNICZAK**, Dian **CHEN**, Andres **POSADA SANCHEZ COBIZA**