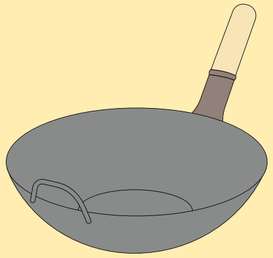


TEAM WOK N' ROLL

STOCK MARKET PREDICTION THROUGH TECHNICAL ANALYSIS USING MACHINE LEARNING TECHNIQUES

Domain : Finance

INTRODUCTION

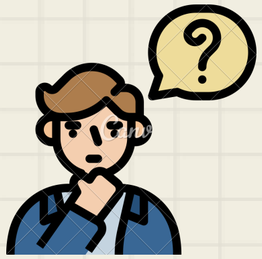
STOCK PRICE

Reflection of extremely complex market interactions, and policies, random and non-linear making it difficult to predict.

Ever-changing data

high accuracy hard to achieve

PROBLEM DEFINITION



- **Stock prices are dynamic in the market and can be influenced by different factors:**
  - supply and demand
  - trading volumes
  - market momentum
- **Investors often make mistakes in purchasing resulting in losing money**

OBJECTIVE & GOALS

We have used a dataset describing 22 selected stocks along with other features (open price, high price, low price, close price, and volume) to :

1

Predict the daily closing prices of 22 selected stocks for 90 days starting 1 Jan 2022

2

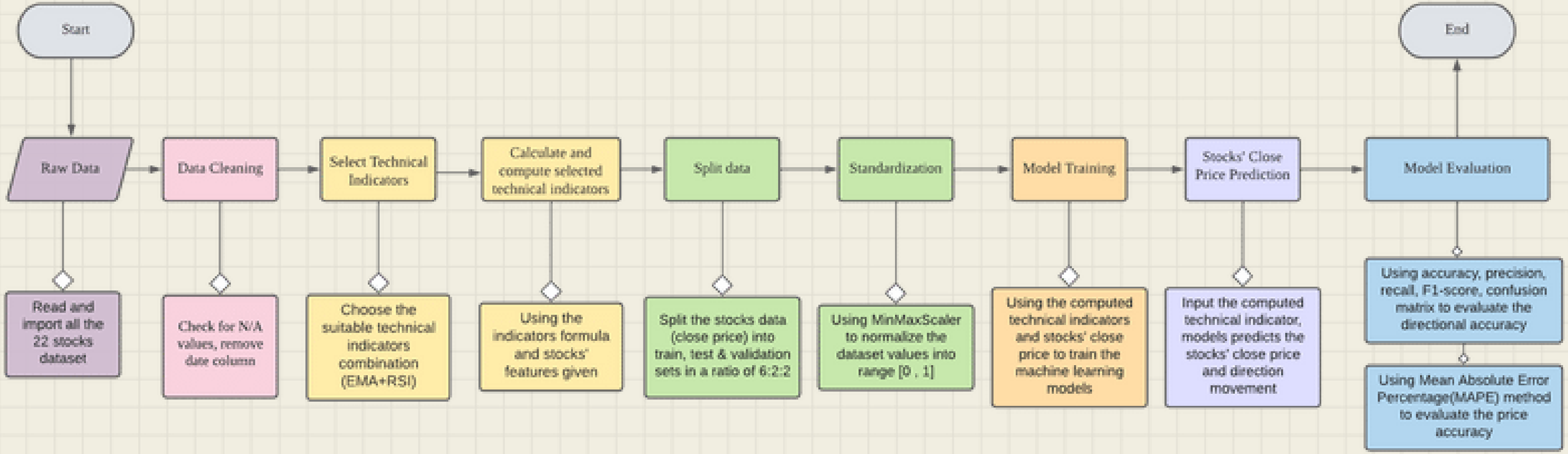
Train machine learning models with selected technical indicators combination

3

Achieve high directional accuracy and high price accuracy

PROPOSED SOLUTIONS

Build machine learning model



Dataset timeframe

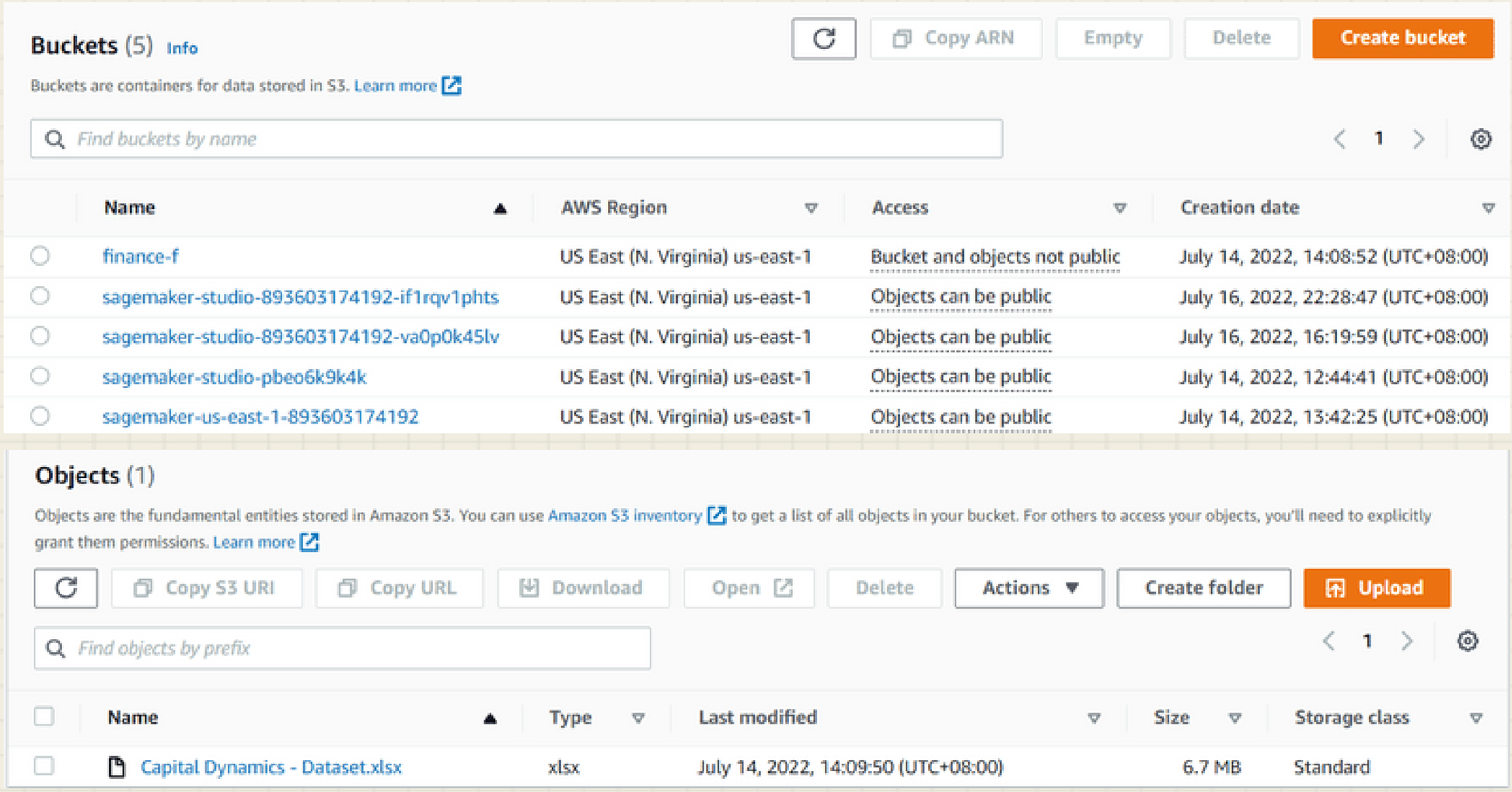
Technical indicator calculation: 22 stocks data since inception to 31 May 2022

ML models training process : 22 stocks data since inception to 31 Dec 2021

Stocks close price prediction : 90 days start from 1 Jan 2022

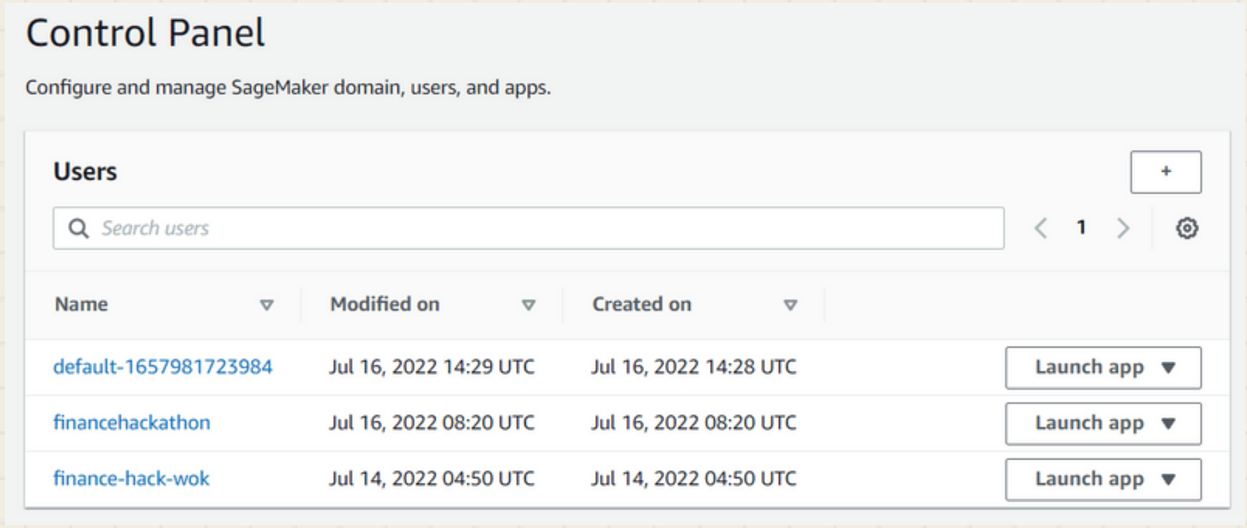
PROPOSED SOLUTIONS

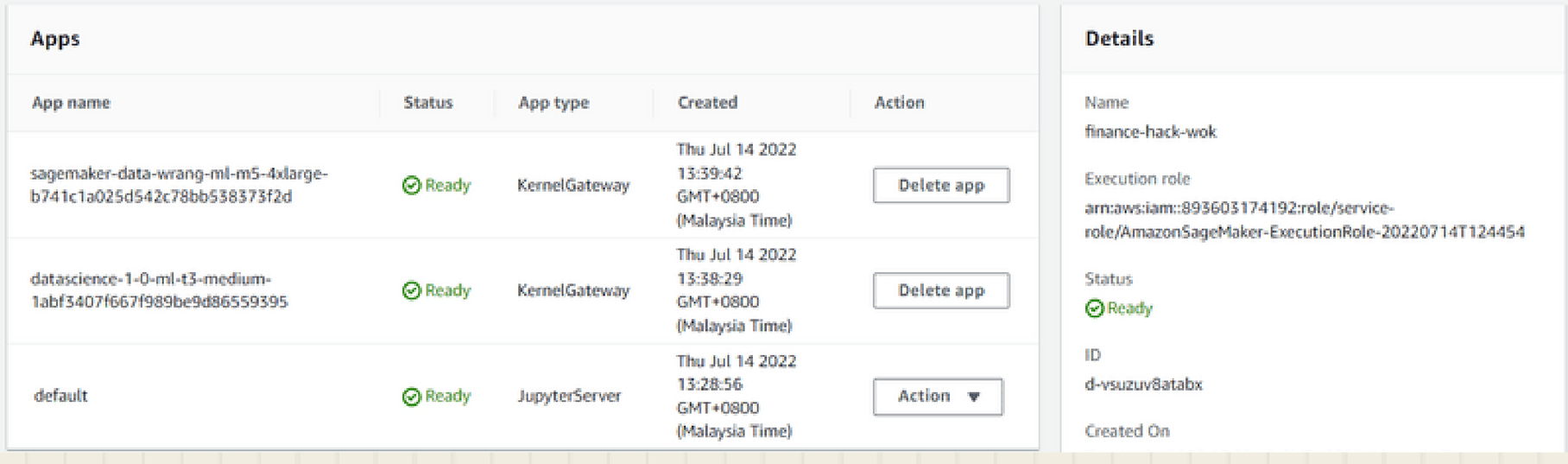
AWS S3 to store dataset



PROPOSED SOLUTIONS

AWS SageMaker Studio





PROPOSED SOLUTIONS

STEP 1: IMPORT ALL THE RAW DATA.

22 stocks datasets are imported with the training set, testing set and validation set distribution.

```
xls = pd.ExcelFile(r"C:\Users\Tiow Kit Keong\Downloads\Capital Dynamics - Dataset.xlsx")

AHEALTH = pd.read_excel(xls, 'AHEALTH')
APM = pd.read_excel(xls, 'APM')
BIOHLDG = pd.read_excel(xls, 'BIOHLDG')
BSTEAD = pd.read_excel(xls, 'BSTEAD')
CAPITALA = pd.read_excel(xls, 'CAPITALA')
EUPE = pd.read_excel(xls, 'EUPE')
HPMT = pd.read_excel(xls, 'HPMT')
ICAP = pd.read_excel(xls, 'ICAP')
KGB = pd.read_excel(xls, 'KGB')
KRONO = pd.read_excel(xls, 'KRONO')
LUXCHEM = pd.read_excel(xls, 'LUXCHEM')
MKH = pd.read_excel(xls, 'MKH')
OCK = pd.read_excel(xls, 'OCK')
OCNCASH = pd.read_excel(xls, 'OCNCASH')
PADINI = pd.read_excel(xls, 'PADINI')
PARKSON = pd.read_excel(xls, 'PARKSON')
SALUTE = pd.read_excel(xls, 'SALUTE')
SAM = pd.read_excel(xls, 'SAM')
SURIA = pd.read_excel(xls, 'SURIA')
TONGHER = pd.read_excel(xls, 'TONGHER')
UTDPLT = pd.read_excel(xls, 'UTDPLT')
WELLCAL = pd.read_excel(xls, 'WELLCAL')
```

	Date	Open	High	Low	Close	Volume
0	2000-06-26	0.3449	0.3658	0.2927	0.2968	23925000
1	2000-06-27	0.2979	0.3909	0.2979	0.3575	42328100
2	2000-06-28	0.3616	0.4285	0.3491	0.4223	33131200
3	2000-06-29	0.4243	0.4724	0.4118	0.4118	31350000
4	2000-06-30	0.3993	0.4369	0.3909	0.3930	10734400
...	...	...	...	...	...	...
4470	2022-05-25	2.8122	2.8612	2.8122	2.8318	69100
4471	2022-05-26	2.8416	2.8416	2.8318	2.8416	7100
4472	2022-05-27	2.8416	2.9004	2.8416	2.9004	67000
4473	2022-05-30	2.9004	2.9396	2.9004	2.9298	60000
4474	2022-05-31	2.9396	2.9396	2.9200	2.9200	46000

```
dataPreprocessing(AHEALTH)
```

	Open	High	Low	Volume
0	0.043724	0.045101	0.042551	0.565225
1	0.031804	0.050958	0.044010	1.000000
2	0.047960	0.059731	0.058381	0.782723
3	0.063862	0.069974	0.075980	0.740642
4	0.057521	0.061691	0.070113	0.253596
...	...	...	...	...
4470	0.669482	0.627336	0.749719	0.001628
4471	0.676938	0.622763	0.755221	0.000163
4472	0.676938	0.636482	0.757971	0.001578
4473	0.691851	0.645629	0.774475	0.001413
4474	0.701793	0.645629	0.779976	0.001082

STEP 2: DATA CLEANING AND REMOVE IRRELEVANT DATA.

Check for N/A and duplicated values and remove the date column.

```
def checkMissingValues(df):
    return df.isnull().sum()
```

```
checkMissingValues(AHEALTH)
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

```
def dataPreprocessing(df):
    df = df.drop(['Date'], axis=1)
    X_data = df.drop(['Close'],axis=1)
```

```
def checkDuplicatedData(df):
    return df.duplicated().sum()
```

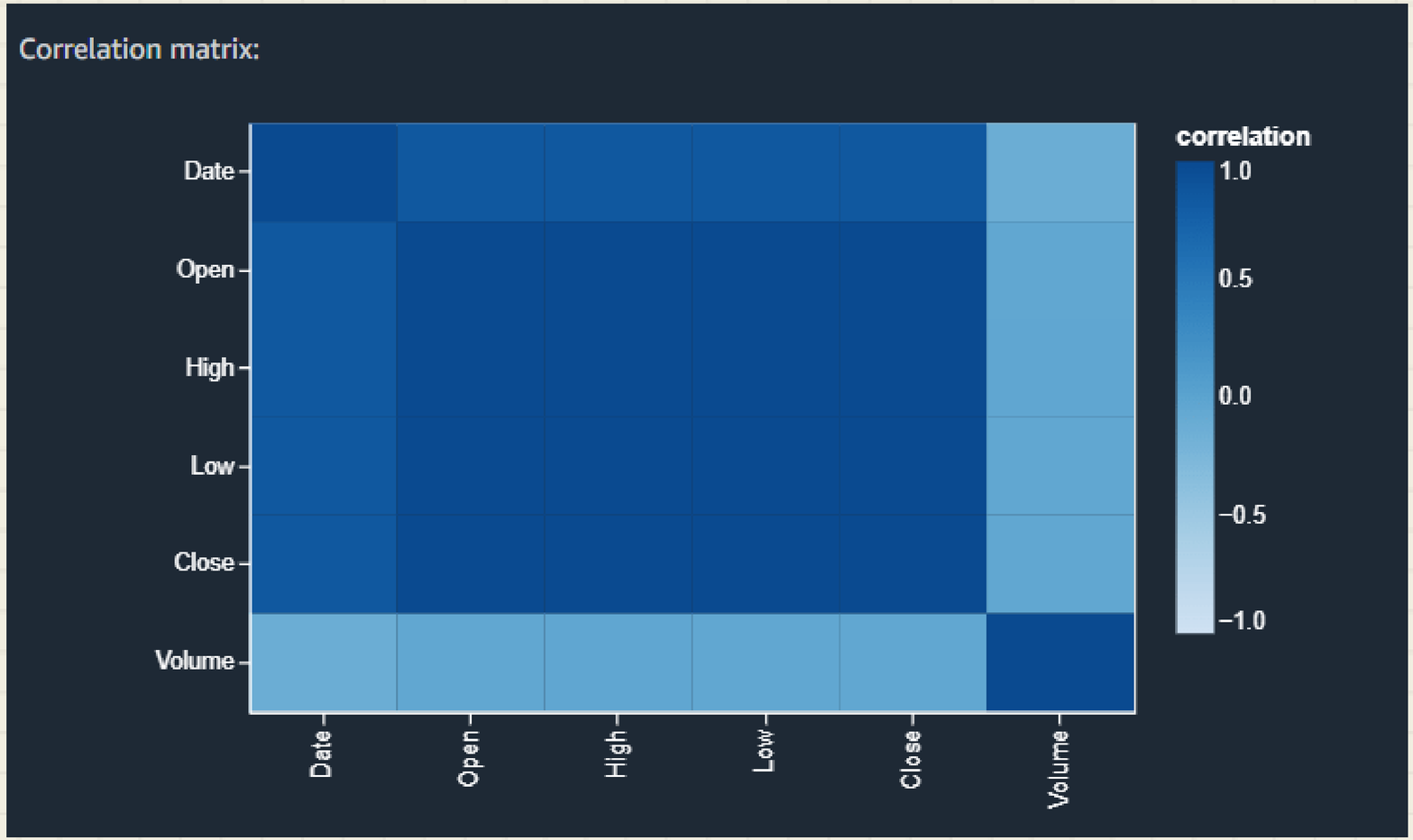
```
checkDuplicatedData(PARKSON)
```

```
0
```

PROPOSED SOLUTIONS

Step 3: Data Visualization & Exploration

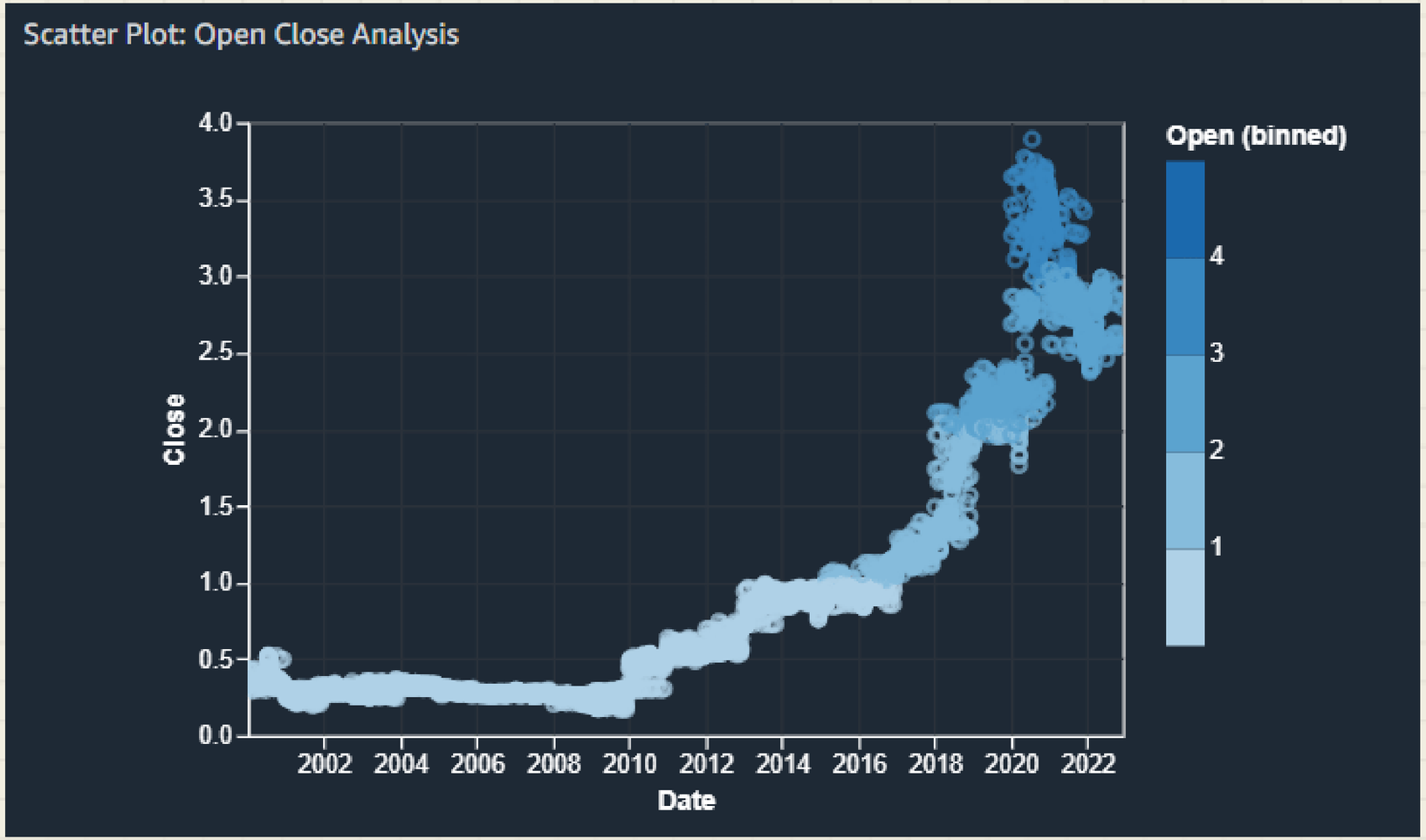
Using AWS SageMaker's Data Wrangler to make visualization.



PROPOSED SOLUTIONS

Step 3: Data Visualization & Exploration

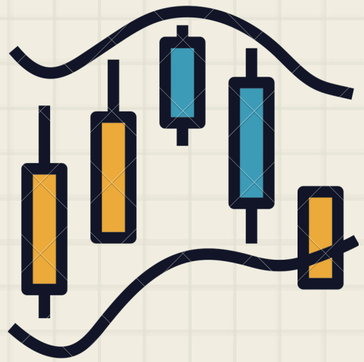
Using AWS SageMaker's Data Wrangler to make visualizations.



PROPOSED SOLUTIONS

STEP 4: SELECT TECHNICAL INDICATORS

TECHNICAL INDICATORS SELECTED:



Relative Strength Index (RSI)

+

Exponential Moving Average (EMA)

+

Percentage Volume Oscillator (PVO)

PROPOSED SOLUTIONS

WHY ?

1. **Compared to the SMA, the EMA weighs recent price changes more heavily than later changes in price. This means that the EMA is more responsive than the SMA to current price fluctuations.**

2. **RSI is often used to obtain an early sign of possible trend changes. Therefore, adding exponential moving averages (EMAs) that respond more quickly to recent price changes can help.**

3. **The volume oscillator consists of two moving averages of volume – one fast and one slow. It is used as a confirmation indicator.**

PROPOSED SOLUTIONS

STEP 5: COMPUTE THE SELECTED TECHNICAL INDICATORS

We compute the selected technical indicators which are RSI and EMA using the given formula.

$$\mathbf{RSI} = 100 - \frac{100}{1+\mathbf{RS}}$$

$$\mathbf{where\ RS} = \frac{\mathbf{Average\ Gain}}{\mathbf{Average\ Loss}}$$

To Compute RS

- First Average Gain = $\frac{Sum\ of\ Gains\ over\ the\ past\ 14\ periods}{14}$
- First Average Loss = $\frac{Sum\ of\ Losses\ over\ the\ past\ 14\ periods}{14}$

For the second, and subsequent RS,
- Average Gain

= [(Previous Average Gain) x 13 + Current Gain] / 14.

- Average Loss

= [(Previous Average Loss) x 13 + Current Loss] / 14.

EMA = Price (t) x k + EMA (y) x (1-k)

Where:

t = today

y = yesterday

N = number of days in EMA

k = 2 ÷ (N+1)

PROPOSED SOLUTIONS

STEP 5: COMPUTE THE SELECTED TECHNICAL INDICATORS

We compute the selected technical indicators which are RSI and EMA using the given formula.

VO in points

Shorter EMA - Longer EMA

VO in percentage

$$\left(\frac{Shorter\ EMA\ -\ Longer\ EMA}{Longer\ EMA}\right) \times 100$$

PROPOSED SOLUTIONS

STEP 6: COMPUTE THE SELECTED TECHNICAL INDICATORS

```
def calculateEMA(df,stock_name):

    ema = df['Close'].ewm(com=13, adjust=False).mean()
    df['EMA'] = ema

    # Comparison plot b/w stock values & EMA
    plt.figure(figsize=(15,15))
    title = "Close Price & EMA Values for "
    title = title + stock_name
    plt.title(title)
    plt.plot(df['Close'], label="Close Price", color="blue")
    plt.plot(df['EMA'], label="EMA Values", color="#ff0000")
    plt.xlabel("Days")
    plt.ylabel("Price")
    plt.legend()
    plt.show()
    return df

def plotRSI(df,stock_name):
    EMA_values = df['EMA']
    delta = df['Close'].diff()
    up = delta.clip(lower=0)
    down = -1*delta.clip(upper=0)
    ema_up = up.ewm(com=13, adjust=False).mean()
    ema_down = down.ewm(com=13, adjust=False).mean()
    rs = ema_up/ema_down
    df['RSI'] = 100 - (100/(1 + rs))
    RSI_values = df['RSI']

    # Skip first 14 days to have real values
    ticker = df.iloc[14:]
    RSI_data = ticker['RSI']

    #print(ticker)
    plt.figure(figsize=(15,15))
    title = "RSI Values for "
    title = title + stock_name
    plt.title(title)
    plt.plot(RSI_data, label="RSA Values", color="#0aa6ff")
    plt.axhline(30, color='r', linestyle='--')
    plt.axhline(70, color='r', linestyle='--')
    plt.xlabel("Days")
    plt.ylabel("RSI Value")
    plt.show()
    return df
```

```
import pandas_ta as ta

def calculateVO(df,stock_name):
    #Volume Oscillator
    volume = df["Volume"]
    pvo = df.ta.pvo(volume=volume, fast=14, slow=28, signal=9, scalar=100, offset=0)
    df["PVO"] = pvo["PVO_14_28_9"]
    print(df["PVO"])
    ticker = df.iloc[14:]
    plt.figure(figsize=(15,15))
    title = "PVO Values for "
    title = title + stock_name
    plt.title(title)
    plt.plot(ticker["PVO"], label="Percentage Volume Oscillator", color="#E9184C")
    plt.xlabel("Days")
    plt.ylabel("Values")
    plt.axhline(0, color='b', linestyle='--')
    plt.legend()
    plt.show()
    return df
```

PROPOSED SOLUTIONS

STEP 6: COMPUTE THE SELECTED TECHNICAL INDICATORS

We compute the selected technical indicators which are RSI and EMA using the given formula.

$$\mathbf{RSI} = 100 - \frac{100}{1+\mathbf{RS}}$$

$$\mathbf{where\ RS} = \frac{\mathbf{Average\ Gain}}{\mathbf{Average\ Loss}}$$

To Compute RS

- First Average Gain = $\frac{Sum\ of\ Gains\ over\ the\ past\ 14\ periods}{14}$
- First Average Loss = $\frac{Sum\ of\ Losses\ over\ the\ past\ 14\ periods}{14}$

For the second, and subsequent RS,
- Average Gain

= [(Previous Average Gain) x 13 + Current Gain] / 14.

- Average Loss

= [(Previous Average Loss) x 13 + Current Loss] / 14.

EMA = Price (t) x k + EMA (y) x (1-k)

Where:

t = today

y = yesterday

N = number of days in EMA

k = 2 ÷ (N+1)

PROPOSED SOLUTIONS

STEP 6: COMPUTE THE SELECTED TECHNICAL INDICATORS

We compute the selected technical indicators which are RSI and EMA using the given formula.

VO in points

Shorter EMA - Longer EMA

VO in percentage

$$\left(\frac{Shorter\ EMA\ -\ Longer\ EMA}{Longer\ EMA}\right) \times 100$$

PROPOSED SOLUTIONS

STEP 7: SELECT DATA WITHIN A TIME FRAME FOR TRAINING AND EVALUATING

```
def a1_timeframe(df):
    df['Date'] = pd.to_datetime(df['Date'])
    end_date = '2021-12-31'
    mask = df['Date'] <= end_date
    a1 = df.loc[mask]
    rf = targetcolumn(a1)
    return rf
```

```
def a2_timeframe(df, rf):
    a2 = df
    a2['Date'] = pd.to_datetime(a2['Date'])
    start_date1 = '2021-12-31'
    reversemask = a2['Date'] > start_date1
    a2 = a2.loc[reversemask]
    a2 = a2.iloc[0:90]
    targetcolumn2(a2, rf)
```

STEP 8: SPLIT DATA

Split the stocks data (close price) into train, test sets in a ratio of 8:2

```
def RF(X_data, y):

    X_train_AHEALTH, X_test_AHEALTH, y_train_AHEALTH, y_test_AHEALTH = train_test_split(X_data, y, test_size=0.2, random_state = 42)
```

PROPOSED SOLUTIONS

STEP 9: STANDARDIZATION

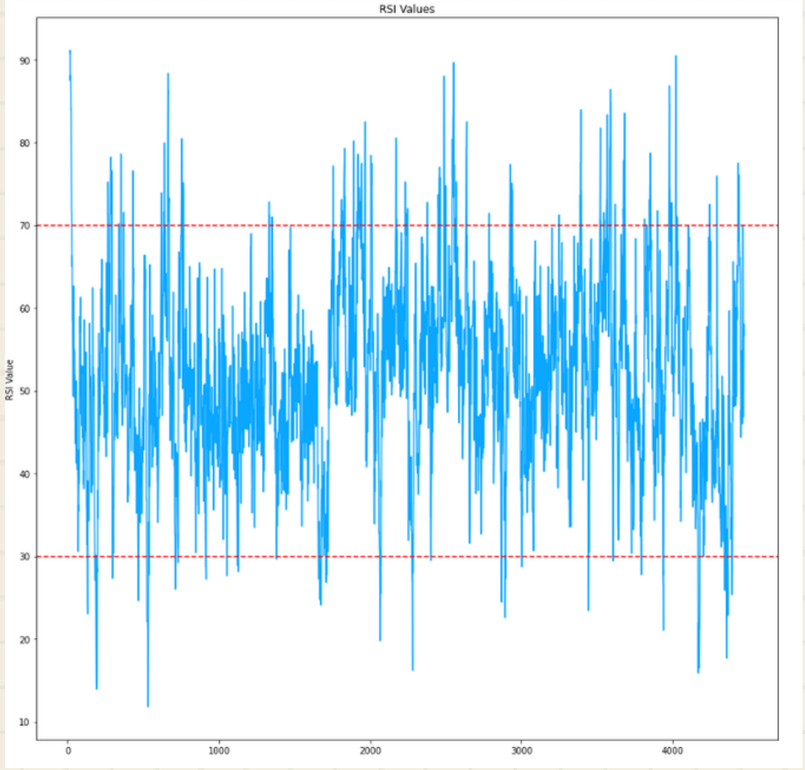
Using MinMaxScaler to normalize the dataset values into range [0 , 1]

```
MinMaxScaler = preprocessing.MinMaxScaler()
X = MinMaxScaler.fit_transform(X_data)
newdata = pd.DataFrame(X ,columns=['Open', 'High', 'Low', 'Volume'])
return newdata
```

PROPOSED SOLUTIONS

STEP 10: MODEL TRAINING

The computed technical indicators and stocks' close prices are used to train the machine learning models.



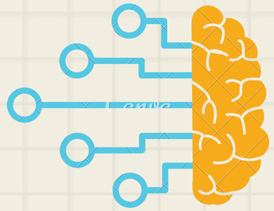
PROPOSED SOLUTIONS

STEP 11: STOCKS' CLOSE PRICE PREDICTION

Pass the computed technical indicators to the models to predict the stocks' close price and direction movement

Random Forest with Grid Search Techniques

```
### Grid Search to tune hyperparameter for random forest
param_grid = {
    'n_estimators' : [50,60,65,70,75,80,85,90,100],
    'max_depth' : [5,6,7,8,9,10,11,12]
}

### Grid Search
RF=GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=2,n_jobs=3)
RF.fit(X_train_AHEALTH, y_train_AHEALTH)

best_params=RF.best_params_

max_depth=best_params["max_depth"]
n_estimators=best_params["n_estimators"]

rf=RandomForestRegressor(random_state=42,n_estimators=n_estimators,max_depth=max_depth)
rf.fit(X_train_AHEALTH, y_train_AHEALTH)

# Use the forest's predict method on the test data
predictions = rf.predict(X_test_AHEALTH)
```



PROPOSED SOLUTIONS

STEP 12: MODEL EVALUATION

- **Accuracy, precision, recall, F1-score, and confusion matrix are used to evaluate the directional accuracy.**
- **The Mean Absolute Percentage Error (MAPE) method is employed to evaluate the price accuracy.**

Random Forest with Grid Search Techniques

```
# Calculate the absolute errors
errors = abs(predictions - y_test_AHEALTH)

# Print out the mean absolute percentage error (mape)
print(f'Mean Absolute Percentage Error: {round(np.mean(errors)*100, 6)}%')
print("")

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test_AHEALTH)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy test:', round(accuracy, 6), '%.')
print("")

r_sq = r2_score(y_test_AHEALTH, predictions)
print('coefficient of determination:', np.round(r_sq,6))
```

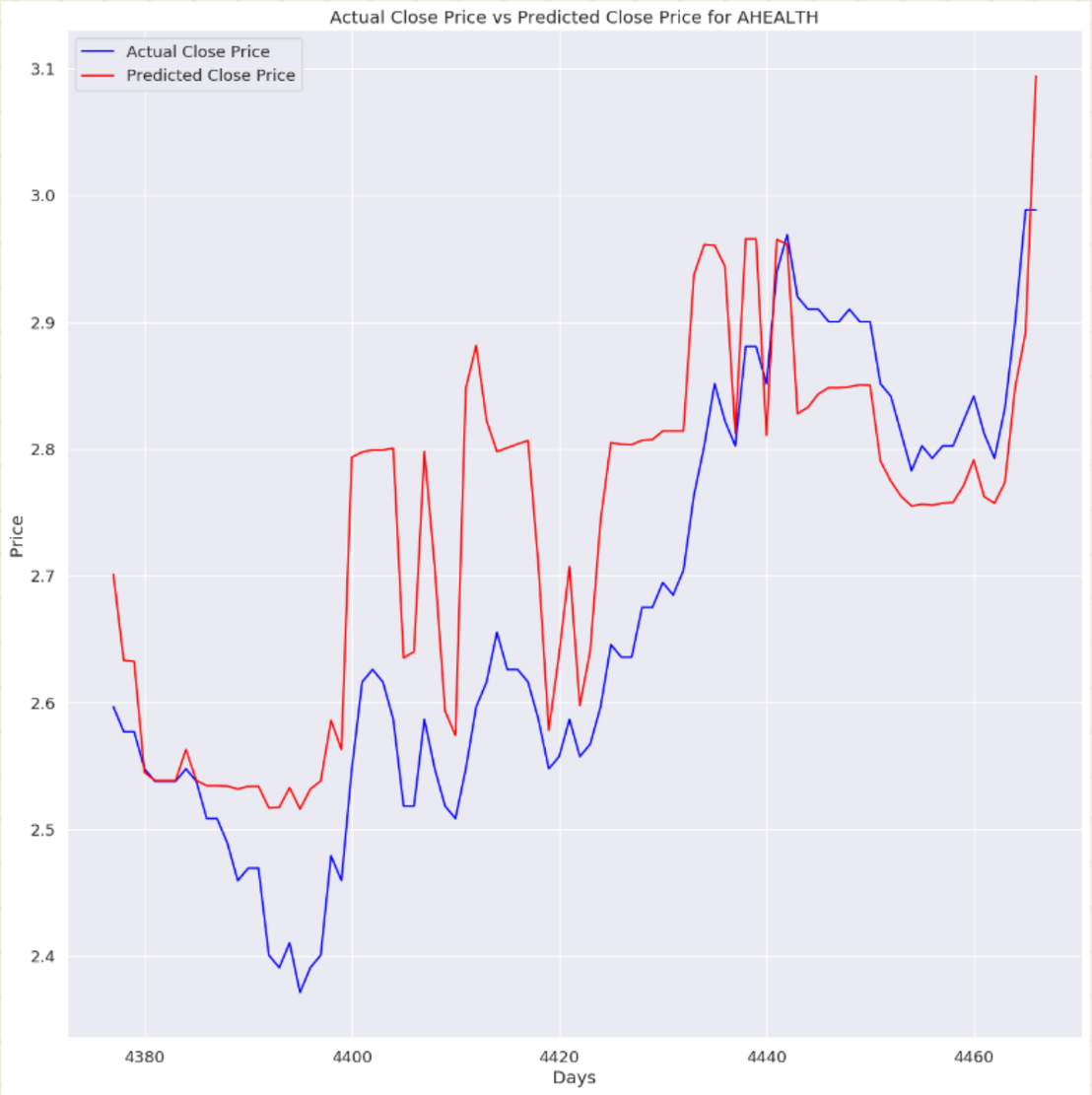
```
Mean Absolute Percentage Error: 0.653691%

Accuracy test: 99.301326 %.

coefficient of determination: 0.999698
```

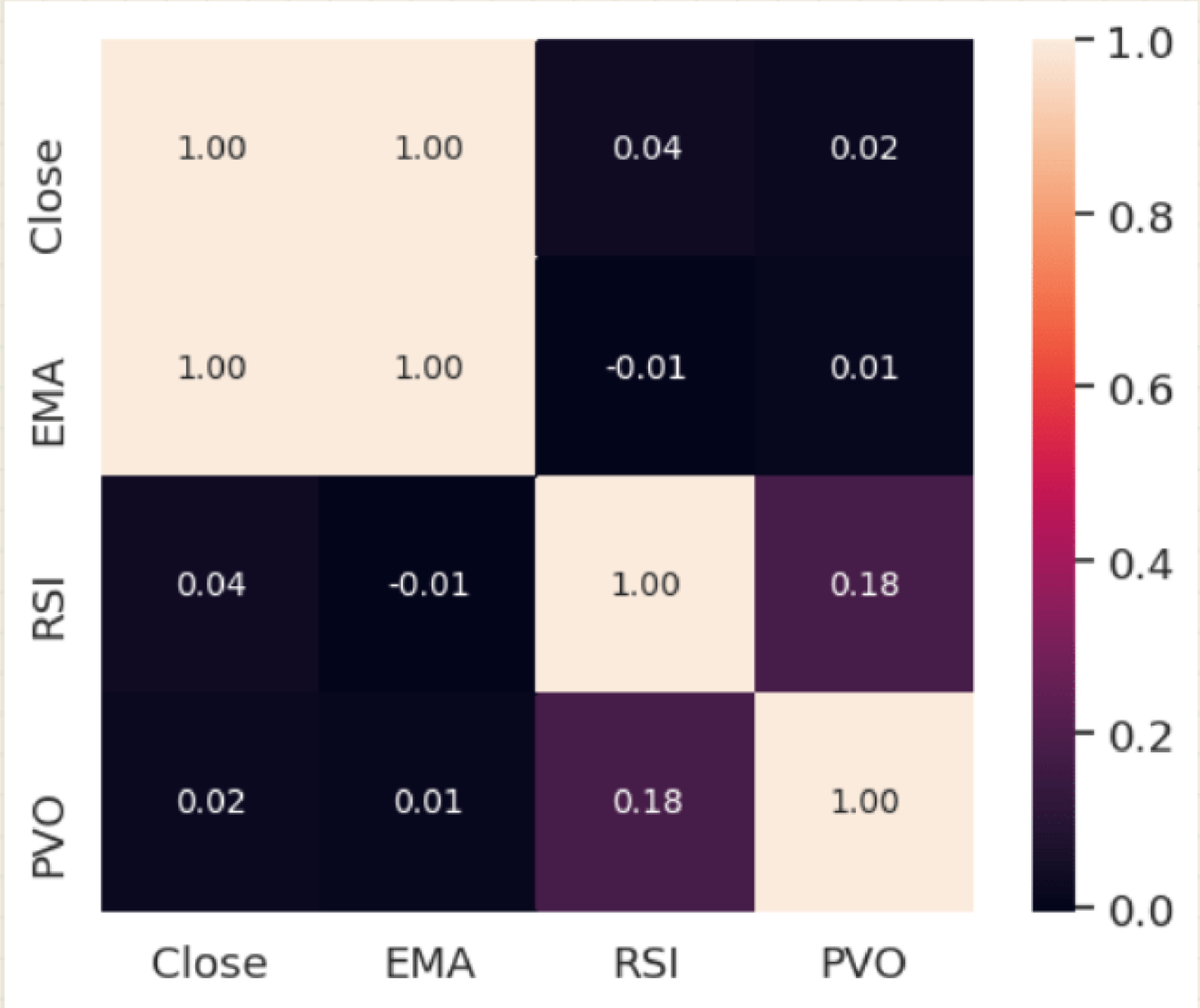
Linear Regression Model

```
# Calculate the absolute errors
errors = abs(predictions - y_test_AHEALTH)

# Print out the mean absolute percentage error (mape)
print(f'Mean Absolute Percentage Error: {round(np.mean(errors), 6)}%')
print("")

# Calculate mean absolute percentagy_test_AHEALTHe error (MAPE)
mape = 100 * (errors / y_test_AHEALTH)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 6), '%.')
print("")

r_sq = r2_score(y_test_AHEALTH, predictions)
print('coefficient of determination:', np.round(r_sq,6))
```

```
Mean Absolute Error: 0.022553%

Accuracy: 96.040289 %.

coefficient of determination: 0.998042
```

IMPLEMENTATION





IMPLEMENTATION

Classification (Directional prediction)

- **Accuracy, precision, recall, F1-score, and confusion matrix are used to evaluate the directional accuracy.**
- **E.g. Random forest classifier, Logistic regression, and so on.**

```
def DirectionalPrediction(df):
    print("Directional Classifier")
    df["Direction"] = "x"
    x = df.Open
    y = df.Close
    z = df.Direction
    start = 27
    for i in range(start,(len(df.Direction)+start)):
        if x[i] == y[i]:
            z[i] = "sideways"
        elif x[i] > y[i]:
            z[i] = "bullish"
        else:
            z[i] = "bearish"

    #print(AHEALTH["Direction"])

    # Encode the three classes: ,"bullish","sideways",bearish" into 0,1,2 respectively.
    df['Direction'] = df['Direction'].map({'bullish' :0, 'sideways' :1, 'bearish' :2}).astype(int)

    #Plot correlation matrix
    k = 4 #number of variables for heatmap
    data = df.drop(['Open','Date','High','Low','Volume'], axis=1)
    corrmat2=data.corr()
    cols = corrmat2.nlargest(k, "Direction")["Direction"].index
    cm = np.corrcoef(data[cols].values.T)
    sns.set(font_scale=1.2)
    hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values,
    plt.show()
```

IMPLEMENTATION

Classification (Directional prediction)

```
#Naive Bayes Model
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
print()
print("Naive Bayes Model")
print()
param_grid_nb = {'var_smoothing':np.logspace(0,-19,num=100)}
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=10, n_jobs=-1)
gnb=nbModel_grid.fit(X_train, y_train)
# training a Naive Bayes classifier
gnb_predictions = gnb.predict(X_test)

# accuracy on X_test
accuracy = gnb.score(X_test, y_test)
```

IMPLEMENTATION

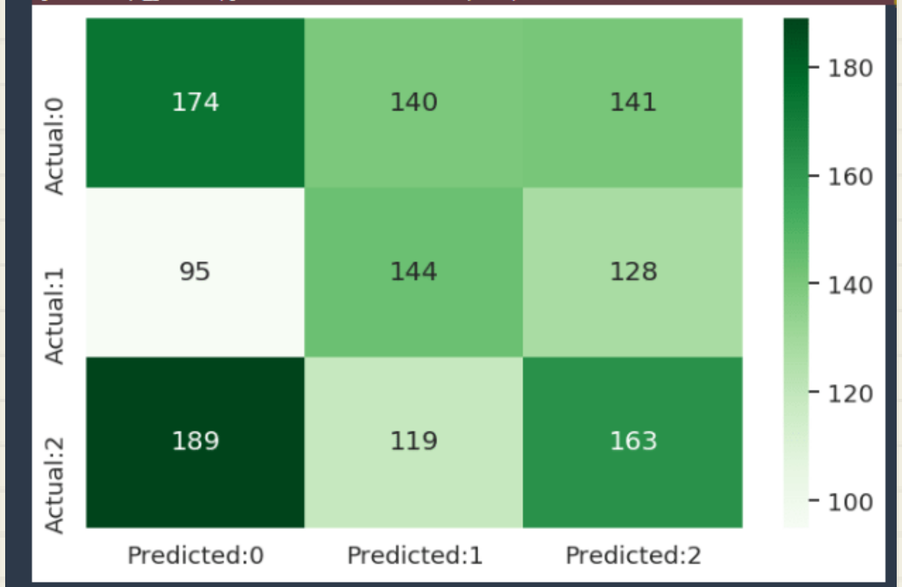
Classification (Directional prediction)

```
#XGBoost model
print()
print("XGBoost Model")
print()
X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 42)
xgb_model = xgb.XGBClassifier(objective="multi:softprob", random_state=42)
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
accuracy = xgb_model.score(X_test,y_test)
cm = confusion_matrix(y_test, y_pred)

conf_matrix = pd.DataFrame(data = cm,
columns = ['Predicted:0', 'Predicted:1','Predicted:2'],
index =['Actual:0', 'Actual:1',"Actual:2"])
plt.figure(figsize = (8, 5))
sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = "Greens")
plt.show()
```

IMPLEMENTATION

Classification (Directional prediction)

```
Naive Bayes Model

Fitting 10 folds for each of 100 candidates, totalling 1000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 926 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    3.7s finished
```




```
The details for confusion matrix is =
              precision    recall  f1-score   support

           0       0.38      0.38      0.38       455
           1       0.36      0.39      0.37       367
           2       0.38      0.35      0.36       471

    accuracy                           0.37      1293
   macro avg       0.37      0.37      0.37      1293
weighted avg       0.37      0.37      0.37      1293


0.3720030935808198
```

IMPLEMENTATION

Classification (Directional prediction)

TARGET BENEFICIARIES/STAKEHOLDERS

Stockbrokers

Challenges before making a buy/sell decision:

- assess a large number of stock information
- constantly monitor the stock price movements
- evaluate the past performance of the stock

Fund managers

Investment advisory tools help them with:

- stock portfolio management
- implementing investment strategies
- investment evaluation, managing trading funds

Local/Foreign investors

Financial/Investment Analysts

Publicly Traded Company

LIMITATIONS OF THE EXISTING SYSTEM

1. Non-Portable
2. Accuracy is relatively low
3. High computational cost
4. Not efficient in handing enormous amount of data

WHY DOES OUR MODEL HAVE HIGHER FEASIBILITY OF SUCCESS AND POTENTIAL FOR FUTURE EXECUTION ?

1. High accuracy
2. Portable
3. Can be trained easily even on systems with relatively low computational power
4. Lower Time Complexity

BUSINESS VALUE

Obtain accurate future stock price based on a large datasets

Accurate mean values with a smaller margin of error

Purchase stocks according to our predictions based on the use of technical indicators

Save time and reduce risks

Visualise the performance of each stock

Stakeholder can see the difference between the actual stock price and predicted stock price

BUSINESS VALUE

High Scalability

Still can works efficiently even when the dataset grows larger

Easy to retrain model

Easy to retrain model frequently with the latest datasets to achieve higher accuracy

FUTURE IMPROVEMENTS/WORKS

1. Add more types of technical indicator, e.g. volatility indicators.

2. Consider another technical indicator for verifying the signal.

3. Changing the definition of oversold and overbought based on the market.

4. Add a true oscillator like the STOCHASTIC indicator to counter ranging market.

CONCLUSION

Our model is able to predict the future stock price accurately with the combination of RSI & EMA technical indicators. Besides, our model also can visualize the stock price movement, which enables the user to analyze the market easily. This can help them to make the most accurate buy/sell decision at the right time.