COMPARISON K-NN REGRESSION AND SVR MSE IN 5 MOST ACTIVE STOCK BASED ON HISTORICAL DATA

[1]Michael The, [2]Rosita Herawati
[1,2]Program Studi Teknik Informatika Fakultas Ilmu Komputer,
Universitas Katolik Soegijapranata
[1]21k10014@student.unika.ac.id, [2]rosita@unika.ac.id

ABSTRACT

This project aims to address the challenge of accurate stock price prediction by comparing the error prediction of two popular machine learning algorithms, K-Nearest Neighbors Regressor (KNN-R) and Support Vector Regressor (SVR) based on historical stock data. Based on previous research about stock prediction, the study evaluates the error in predicting future stock prices based on features such as historical price and volume. The SVR is evaluated for its ability to fit a regression model that minimizes prediction errors. In terms of accuracy, KNN Regression is hypothesized to outperform SVR. The models are trained and rested across multiple stock datasets, and results indicate that SVR consistently achieves superior predictive performance compared to KNN Regression. These findings highlight the importance of selecting the right algorithm for stock price prediction.

Keywords: stock prediction, K-Nearest Neighbors Regression, Support Vector Regression, accuracy, historical data

INTRODUCTION

Stocks are a branch of economics that is very mainstream nowadays. Many people invest in certain companies in the form of shares. Shares are very similar to daily economic activities where we buy at low prices and sell when prices reach their peak. A common problem that occurs from stocks is when will be the best time to buy or sell the stocks. Stock price prediction creates challenges due to the complex and dynamic financial markets. According to Madhumita and Gor [5] a small amount of increasing accuracy in predicting stock can be very impactful and make a huge difference in profit for the investors.

In this paper the researcher is going to make a prediction to determine when the lowest price of a particular stock in this case Amazon, Apple, Microsoft, Google, and Nvidia stocks will be and determine when the highest price will be. The researcher is using the K-NN R (K-Nearest Neighbors) Regressor and SVR (Support Vector Regressor) algorithms based on the past data. Then the researcher is going to compare the lowest error from both algorithm predictions.

The proposed solution is to utilize the K-NN Regressor and SVR algorithms to predict the lowest error of prediction of each stock based on historical data. These two algorithms are trained by using previous stock price patterns to make predictions about stock closing prices. Evaluation is carried out by calculating the error that will be returned from the training process and comparing the prediction value within the actual value based on the dataset.

LITERARURE STUDY

The complicated relation between stocks and market activity becomes the most crucial thing for financial analysis. Nayaka, et al [1] state that stocks are very well connected with the market movements, especially in the India stock market, as was pointed out by the authors from BSE SENSEX. Their study uses the algorithm of SVM (Support Vector Machine) and K-NN (K-Nearest Neighbors) for the purpose of predicting the stock market trends giving the ground rules to make the most accepted prediction for the trading decisions. However their methods can make a loss to the people, because their data have a limitation for only the India stock market and can't be implemented outside the dataset from BSE SENSEX.

On the other hand, Sarala, et al [2] state that there are two different types of approaches for analyzing financial markets, the first one is making an evaluation about the price through the qualitative and quantitative. The second one is making a prediction about the stock price based on the historical data. From the current explanation it's clear that the authors are aiming to make a prediction of the stock price based on the historical data. The limitation is the authors do not explain the parameters for the K-NN works, the authors just make a result based on the python code, but overall the experiments resulted very successfully because it can generate a high percentage of accuracy to predict when to buy or sell the stock.

This work extends the concept of predictive modeling by putting forth a way to predict stock prices in the future by looking from the historical price trends that are comparable to the current market. These methods are being studied by Sarala et al. [2] However in this paper, Nakagawa et al. [3] claims that making a prediction using k*-Nearest Neighbor algorithm and an approach called Indexing Dynamic Time Warping returned a very good prediction. The authors want to assist investors in making better investment decisions by using historical data to predict stock price changes more accurately than traditional methods.

Adding to the conversation, Madeeh and Abdullah [4] investigate the development of an efficient stock market prediction model by cutting-edge advanced machine learning methods in their research. The core of the research involves using the K-NN, which operates based on the proximity of a query instance to labeled instances in the feature space and the majority label among the K-nearest neighbors. From the experimental results, it revealed how to use the precision, recall, and F-measures scores and from the experiment it resulted very good for the precision is 93.23%, the recall is 93.12%, and the F-measures is 93.17%.

Madhumita and Gor [5] highlight the increasing use of machine learning techniques in stock market prediction due to the unpredictability of stock prices. The study aims to enhance predictive models by comparing two machine learning algorithms which are Support Vector Regression (SVR) and K-Nearest Neighbors (KNN) in their ability to predict stock prices. The focus is on using these algorithms to forecast future prices based on historical data from Google's stock, spanning from 2016 to 2019. While the study applies both SVR and KNN, it primarily evaluates their performances through metrics such as Root Mean Square Error (RMSE) and accuracy. The research underscores the importance of selecting appropriate models for stock prediction but does not explore more advanced optimization techniques or ensemble methods that could further enhance prediction accuracy. For my paper, I can consider expanding on this by exploring advanced optimization techniques or applying ensemble methods to further improve stock market prediction accuracy, potentially using a broader dataset or focusing on a different market.

Rehman, et.al [6] state that there are many factors that can influence the value of stocks. The study aims to explore and analyze the effectiveness of various supervised machine learning models to create a stock prediction. The algorithms that will be used are SVR, Random Forest(RF) , K-NN Regression and LSTM. The result from this experiment was that KNN Regression has the best performance compared to the other algorithm. For my paper, I can consider conducting a comparative analysis of KNN Regression and SVR by looking at how to preprocess the data.

Y. Huang. [7] discusses the challenges of predicting stock prices and aims to evaluate the effectiveness of three machine learning models which are Support Vector Regression (SVR), Random Forest (RF), and K-Nearest Neighbor (KNN) for predicting Google stock prices. The study's findings reveal that Random Forest outperforms the other models, achieving the lowest Mean Square Error (MSE) of 13.4, making it the most reliable for predictions. For my paper, I can consider exploring the comparative performance of SVR and KNN particularly in the context of different data preprocessing techniques and their impact on prediction accuracy.

Sikder et al. [8] highlight that predicting stock prices is a complex challenge influenced by various factors such as market sentiment, economic conditions, and investor expectations. The study focuses on comparing the effectiveness of two machine learning algorithms: Support Vector Regression (SVR) and K-Nearest Neighbor (KNN) Regression in predicting the daily closing prices of selected stocks on the Dhaka Stock Exchange. The findings indicate that linear SVR outperforms KNN Regression, achieving a lower root mean squared error (RMSE) and a higher R-squared value. For my research, I plan to conduct a comparative analysis of SVR and KNN Regression, emphasizing the importance of hyper-parameter tuning.

Kaliappan et al. [9] discuss several non-linear regression techniques that can predict the COVID-19 reproduction rate, aiming to evaluate their performance and analyze the impact of feature selection and hyperparameter tuning. The algorithms selected for this study include support-vector regression (SVR), k-nearest neighbor (KNN), Random Forest, Gradient Boosting, and XGBOOST. The findings suggest that KNN achieved the best performance, while Random Forest performed optimally when combined with feature selection and hyperparameter tuning. For my paper, I could consider conducting a comparative analysis of KNN and Random Forest, particularly focusing on the influence of various features to achieve a lower error in predicting.

H. Zhao and Y. Chen. [10] emphasize the growing significance of machine learning models in temperature prediction. The study aims to improve forecasting accuracy by comparing three machine learning algorithms: Linear Regression, SVR, CART, KNN and LSTM in short-term temperature prediction data.The research evaluates the performance of these models using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The results show that traditional models can sometimes outperform deep learning models in short-term predictions, challenging the assumption that deep learning is always the best choice. For my paper, I can use the result from this paper to get the MSE for the purpose of the boundary from my experiment result.

The main difference between the previous implementation and this research is that KNN and SVM are usually being used for a classification purpose, while for now the researcher will focus on using both of this algorithm for a regression analysis. Some of the journal articles even declared that KNN can even be almost perfect when predicting the stock market movement. In this research, the researcher will try to create a different style of prediction by taking the minimum error from the experiment.

RESEARCH METHODOLOGY

Research Methodology

K-NN Regression and SVR are both algorithms that can create a good result for making a prediction. K-NN Regression creates a prediction by searching for the best value from the nearest neighbor, by setting the K value and becoming the centroid of the regression. The calculation for searching the best average target values between every other point can be done by using the Euclidean distance, Manhattan distance, or Cosine similarity. The best calculation method that can be used is different for every problem and with the best calculation can minimize the error from the prediction value.

SVR or Support Vector Regression is also good for making a best fit line or a curve between the data. SVR works by creating a line called hyperplane. After creating a line, instead of splitting the data into two classes between upper the line and lower the line. SVR focused on fitting the data within a specific error margin by changing the epsilon parameter from the model.

Table 1. K-NN Regression Distance Equation

K-NN Regression Distance Equation	
Euclidean Distance	$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan distance	$\sum_{i=1}^{k} \left\| x_i - y_i \right\|$
Cosine Similarity	$\left(\sum_{i=1}^{k} \left(\left\| x_i - y_i \right\| \right)^q \right)^{1/q}$

This equation is taken from the website1. This equation will be used for the calculation distance for KNN Regression algorithm, with the x as the first distance y as the second distance. This is the rough picture of how the KNN Regression will calculate the distance between each neighbor from the dataset.

Dataset Collection

In this paper, the researcher will use the K-NN Regression and SVR algorithm in python to make a comparison prediction based on historical data. I am using 5 different dataset that I got from Kaggle.com. There are Amazon, Apple, Microsoft, and Google stock prices. Both of those algorithms will be used to get the lowest MSE and the prediction values from the models. Then, comparing the prediction value with the actual value to get the best algorithm to create stock price prediction based on the historical data. The total data from each dataset is 5,839 from Amazon, 5,839 data from Apple, 5,839 data from Microsoft, and 4,677 data from Google. The datasets can be downloaded via this link https://www.kaggle.com/datasets/soumendraprasad/stock. From this link, there are multiple files that contain Amazon, Apple, Microsoft and Google stocks dataset. Adding another source of dataset from kaggle also the researchers find Nvidia data that contains the same column with the total data of 6,393 data that can be downloaded from this link https://www.kaggle.com/datasets/programmerrdai/nvidia-stock-historical-data. All of the dataset will be imported to Google Drive, so it can be easier for the researchers to get the data without uploading it every time if the researchers are going to test the models. This method is the most suitable way to solve data loss problems.

Dataset Visualization

After collecting the data, the researcher will visualize the data using the library from matplotlib. The researcher will put the date as x label and the close price as y label. This process will return a chart of the dataset stock closing price from year to year. Then, The researcher is going to check if the data contains some anomaly dataset from the chart. This will prevent the existence of outlier data.

Data Preprocessing

This process will help the researcher to prepare the raw dataset into the processed data that will be trained into each of the selected models. This process also will help to create a better prediction model. There are several processes which are handling outlier, data sampling, balancing data, and cleaning the dataset. The researcher will handle the outlier using the interquartile method and then store it for the next preprocessing step. The second step after handling outliers is to take samples from the dataset to be trained in the model. Some models can’t handle a larger dataset. This sampling technique will help the model to prevent unused data from being trained. Next, the researcher will do the balancing data, after handling outlier and sampling, the dataset will become imbalanced. The imbalance dataset can create a major error that can cause an underfitting or overfitting prediction in the training process. The last step to do is to clean the dataset, this method will prevent the process of training the unrelated column for the model to train. All of this process will become the important thing to do before fitting the models.

Handling Outlier

After viewing the data from visualization, begin the process of handling the outlier of the data using the interquartile method. Interquartile is one of the solutions to handle some outlier data, interquartile works by taking the difference from 75% of the data from the bottom and taking the 25% of the top value. The researcher will filter all of the data values based on the interquartile range.

Data Sampling

Before using the data to fit the model, the second step after handling the outlier is doing the sampling. In this step, there are many ways to take the sampling methods, but the researcher is going to use the stratified sampling method, which is a technique to ensure that subgroups within a population are proportionally represented in the sample. This step is very critical because it can cause imbalances and potential biases in the model training process.

Balancing Data

After having the filtered data from sampling the data, the length of data will be reduced and create an imbalance length for each of the data. The researcher begins the process of balancing the data. The researcher is going to take the smallest length of the data and then equalize the length of all the dataset based on the smallest length.

Cleaning Unused Data

After all of the steps from handling the outlier, balancing the data, begin the process of cleaning data. The researcher is going to clean the column from the data that will not be used for the training process to prevent the model from training the unnecessary data. This will prevent the model from over-training the unused column or target.

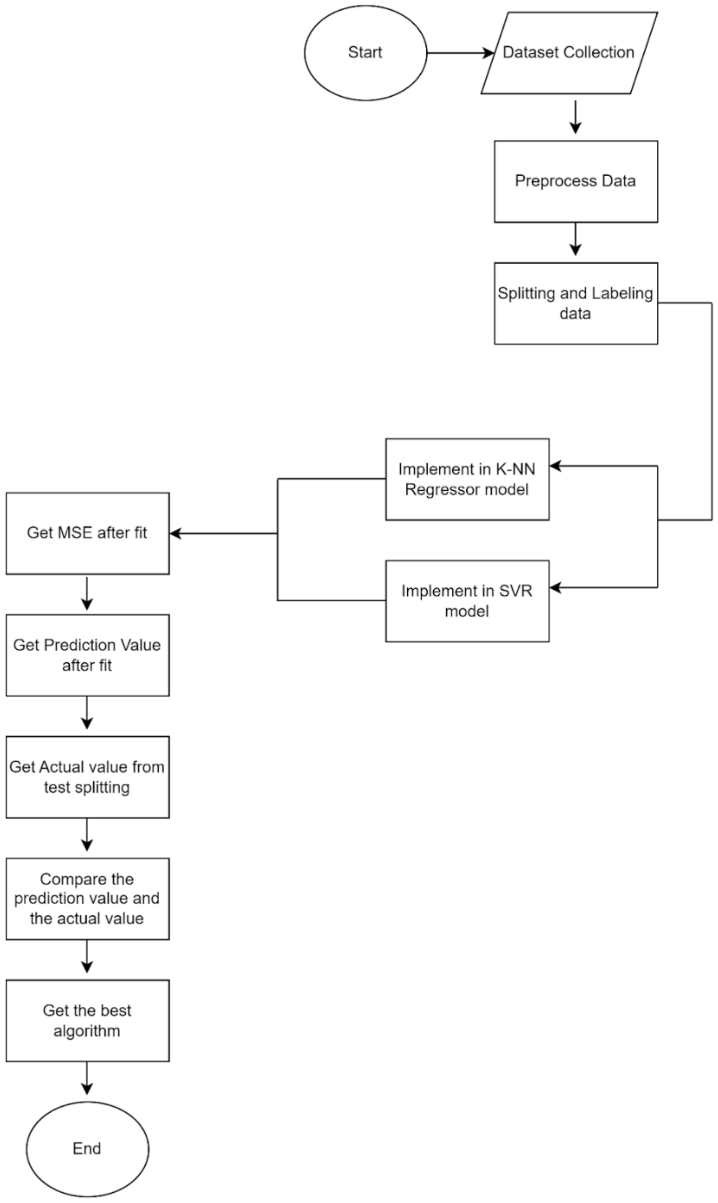
Splitting Data

After having the preprocessing step, the researcher is also going to split the data into training and testing. With the help from python library sklearn.model_selection import train_test_split. The researcher is going to try the ratio of train and test with 90:10, 80:20, and 70:30. This ratio will divide the amount of training and testing data. The best splitting that returns a lower MSE will be used to determine which splitting is the best for the model training process. This process can prevent the model from creating a larger bias or variance.

Feature Importance

This process will return which of the following features will become more impactful from the others. With the help from python library sklearn.inspection import permutation_importance. This library will give a score for each feature that affected the process of model training. The researcher will be using it to return the feature column and the average value of the feature impact. The purpose of this is to know which of the features that will cause a better performance for the model to predict stock closing price.

Comparing Algorithms


Start
Dataset Collection
Preprocess Data
Splitting and Labeling data
Implement in K-NN Regressor model
Implement in SVR model
Get MSE after fit
Get Prediction Value after fit
Get Actual value from test splitting
Compare the prediction value and the actual value
Get the best algorithm
End

Gambar 1. Flowchart Comparing Algorithms

The last step is to compare the best algorithm between KNN Regression and SVR to predict the stock market close price based on the historical data. After getting the clean dataset or the processed data, and splitting the data, it's time for the researcher to try to train the models and get the MSE from the fitting process. The researcher will use the scaling method, which standardizes the close price to have a boundary range number.

IMPLEMENTATION AND RESULTS

Experiment Setup

This experiment uses a regular Google Colab with a capacity of 12.7GB RAM, 15.0 GB RAM GPU, and 36.5 GB Disk. While using the experiment there are several libraries that are used, such as pandas, random, sklearn, and matplotlib. Pandas was used for the process of reading the dataset and appending the dataset into one 'DataFrame' variable. Random was used for random numbers at the process of sampling and balancing. Sklearn to handle all of the preprocessing and the metrics from the training and building the models. Matplotlib to plot the dataset into a chart.

Implementation

This section describes the functionality for each line of code that the researcher creates to achieve the better algorithm to predict the 5 most active stocks. This section also gives the reader some insight into how to process until training the model. First step, the researcher is going to take the uploaded dataset from google drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

Data Collection

This line of code is going to ask for permission to access google drive. After getting the permission the researcher now can possibly take the dataset without concerning the missing dataset. This process will help to set the default data from the downloaded dataset.

```
appledf = pd.read_csv("/content/drive/MyDrive/dataset-skripsi/Apple.csv", delimiter=",")
amazondf = pd.read_csv("/content/drive/MyDrive/dataset-skripsi/Amazon.csv", delimiter=",")
googledf = pd.read_csv("/content/drive/MyDrive/dataset-skripsi/Google.csv", delimiter=",")
microsoftdf = pd.read_csv("/content/drive/MyDrive/dataset-skripsi/Microsoft.csv", delimiter=",")
nvidiadf = pd.read_csv("/content/drive/MyDrive/dataset-skripsi/NVDA.csv", delimiter=",")
```

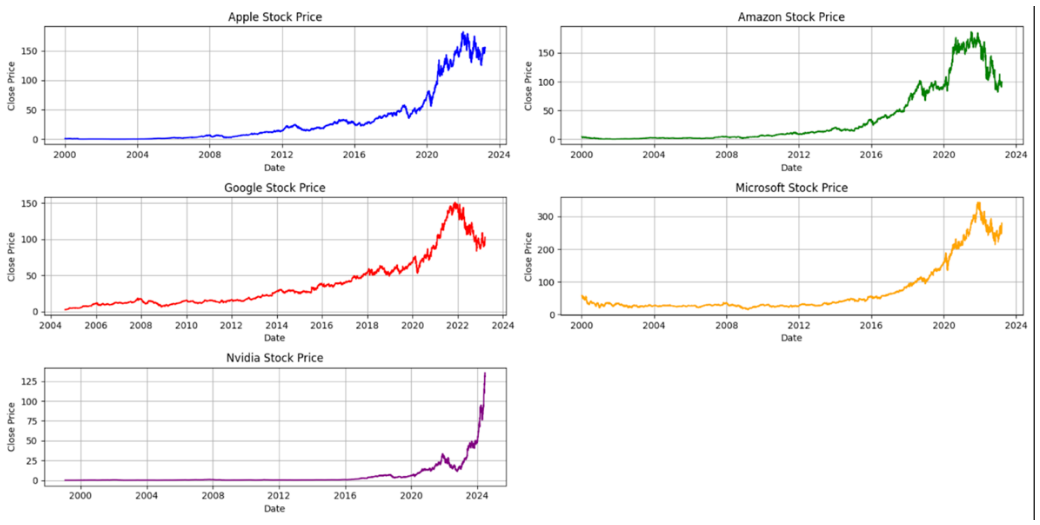
After getting the dataset, the researcher will read the content of the dataset using the pandas library to read the csv file. The researcher will set each dataset into the variable of the stocks name plus "df". Delimiter works for separating each column from the csv file into a separate column.

Data Visualization

Data visualization is conducted to observe the movement of the dataset. This process is going to return into a chart incase of some anomaly or unused data that occured in the dataset. This process also give more insight on handling the data.

```
appledf['Date'] = pd.to_datetime(appledf['Date'])
amazondf['Date'] = pd.to_datetime(amazondf['Date'])
googledf['Date'] = pd.to_datetime(googledf['Date'])
microsoftdf['Date'] = pd.to_datetime(microsoftdf['Date'])
nvidiadf['Date'] = pd.to_datetime(nvidiadf['Date'])

plt.figure(figsize=(16, 8))

plt.subplot(3, 2, 1)
plt.plot(appledf['Date'], appledf['Close'], color='blue')
plt.title('Apple Stock Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)

plt.subplot(3, 2, 2)
plt.plot(amazondf['Date'], amazondf['Close'], color='green')
plt.title('Amazon Stock Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)

plt.subplot(3, 2, 3)
plt.plot(googledf['Date'], googledf['Close'], color='red')
plt.title('Google Stock Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)

plt.subplot(3, 2, 4)
plt.plot(microsoftdf['Date'], microsoftdf['Close'], color='orange')
plt.title('Microsoft Stock Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)

plt.subplot(3, 2, 5)
plt.plot(nvidiadf['Date'], nvidiadf['Close'], color='purple')
plt.title('Nvidia Stock Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)

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

The visualization data will use the library from matplotlib and can be seen from code above in line 10 - 52. This code will help to return the year of date for each stock as an x-axis (x) and the closing prices as a y-axis in the chart. This chart will be divided into a 16 x 8 inch frame. The researcher will use automatically calculating the optimal spacing between subplots and adjusting the margin in line 51.



Gambar 2. Data Visualization

Handle Outliers

In this process, the researcher will take the data into the process of handling the outlier using the interquartile method. This method will return a boundary to cap the dataset value. This process will prevent the model from handling a larger spreading data. This process will be one of the main reasons the model has a better prediction value.

```
53.  def handle_outliers_iqr(df, column_name):
54.      Q1 = df[column_name].quantile(0.25)
55.      Q3 = df[column_name].quantile(0.75)
56.      IQR = Q3 - Q1
57.
58.      lower_bound = Q1 - 1.5 * IQR
59.      upper_bound = Q3 + 1.5 * IQR
60.
61.      df_filtered = df[(df[column_name] >= lower_bound) & (df[column_name]
   <= upper_bound)]
62.      return df_filtered
63.
64.  microsoftdf = handle_outliers_iqr(microsoftdf, 'Close')
65.  amazondf = handle_outliers_iqr(amazondf, 'Close')
```

After having the visualization data and rechecking briefly, the dataset contains some of the unused data range. The unused data appears because of inflation in some years. This inflation triggers a rise in stock prices. That's why in some years anomaly data appears. This anomaly data must be removed from the training process, so the training will have a better result. The method that the researcher will use is called the interquartile method. This process will work by using the difference between the 25% top data and the 75% bottom data. The difference will be used as a boundary to cap the closing price, this means the close price that has value outside the boundary will be trimmed.

Sampling Data

The researcher have done some research in multiple ways of sampling the data. The sampling is quite important to limit the process of model training. While there are many ways of taking samples from the data, the researcher tried to use stratified sampling to take the sample of stock data. This sampling method can cause a better result when taking a sample.

```
66.  def stratified_sampling(df, total_samples):
67.      df['Year'] = pd.to_datetime(df['Date']).dt.year
68.      strata = df['Year'].unique()
69.      stratified_sample = pd.DataFrame()
70.      total_records = len(df)
71.
72.      for stratum in strata:
73.          stratum_data = df[df['Year'] == stratum]
74.          proportion = len(stratum_data) / total_records
75.          n_samples = int(total_samples * proportion)
76.          random_state = random.randint(0, 100)
77.          stratum_sample        =        stratum_data.sample(n=n_samples,
    random_state=random_state)
78.          stratified_sample        =        pd.concat([stratified_sample,
    stratum_sample])
79.
80.      return stratified_sample
81.
82.  appledf_sampled = stratified_sampling(appledf_balanced, 4000)
83.  amazondf_sampled = stratified_sampling(amazondf_balanced, 4000)
84.  googledf_sampled = stratified_sampling(googledf_balanced, 4000)
85.  microsoftdf_sampled = stratified_sampling(microsoftdf_balanced, 4000)
86.  nvidiadf_sampled = stratified_sampling(nvidiadf_balanced, 4000)
```

Stratified sampling works by creating multiple subpopulations with the different characteristics of data for each subpopulation. The researcher will find the year that is unique and set the unique year into each of the subpopulations. From all subpopulations, the researcher will distribute the data into each of the subpopulations. The process also creates balanced data for each of the subpopulations.

This process of stratified sampling can be seen in a function called stratified _sampling in line 66 - 80, by using dataframe and total_samples for the parameter. This action will create the subpopulation based on the unique year. Then using the for loop to build the subpopulation with the proportion of dividing the length from dataframe['Year'] with the length from the parameter data frame. The researcher will create a sample by multiplying the total_samples parameter with the proportion and distribute it inside the subpopulation.

Balancing Data

After having the outlier of data handled, the researcher is going to balance the data. Balancing data is also one of the most important things to do. This process will help the model to train a balanced dataset and not an imbalance dataset. An imbalance dataset can cause more bias in the training process.

```
87.  def balance_datasets(dfs):
88.      min_length = min([len(df) for df in dfs])
89.
90.      balanced_dfs = []
91.      for df in dfs:
92.          random_state = random.randint(0, 100)
93.          df_balanced = df.sample(n=min_length, random_state=random_state)
94.          balanced_dfs.append(df_balanced)
95.
96.      return balanced_dfs
97.
98.  balanced_dfs = balance_datasets([appledf, amazondf, googledf,
   microsoftdf, nvidiadf])
99.  appledf_balanced, amazondf_balanced, googledf_balanced,
   microsoftdf_balanced, nvidiadf_balanced = balanced_dfs
```

The process of handling outliers and stratified sampling will cap the data and cause some of the data to be trimmed. The balanced dataset can create a better prediction for the model. This process is easier than the other preprocessing steps. This process also prevents the model from becoming overfitting or underfitting because of the imbalanced data.

The researcher will create a function to handle the balancing data in line 100 - 109. The important thing of balancing is to get random inside the sampling process. This can create more variability in shuffling the data. SVR and KNN Regression can easily handle a larger spread of data, that's why using the variability can reduce more errors in the prediction process.

Cleaning Data

The last thing to do before training the model is cleaning data. Dropping unused data can be very helpful to prevent the machine from working extra to train the model. This process can help the model to prevent overfitting, training unused data can cause an overfitting result. The most important part of cleaning is to create a better efficiency for the model to train, this is the reason some of the model training can take forever to train.

```
100. def cleaning(df):
101.     df = df.drop(columns=['Date', 'Adj Close'])
102.     nulldata = df.isnull().values.any()
103.     duplicated = df.duplicated().values.any()
104.     if nulldata or duplicated:
        print("Data contains null values or duplicates")
    return df

appledf = cleaning(appledf_sampled)
amazondf = cleaning(amazondf_sampled)
googledf = cleaning(googledf_sampled)
microsoftdf = cleaning(microsoftdf_sampled)
nvidiadf = cleaning(nvidiadf_sampled)
```

This function will help the cleaning process after the all of the preprocessing process. The function begins by deleting the column of ‘Date’ and ‘Adj Close’ from the data frame. Then the function will check if there is missing data or duplicated data, if the data frame contains missing or duplicated data, it will give the researcher a message that contains null or duplicates. Using the same variable name from reading the csv file, will help the researcher for not having a multiple of variables where the variable have the same purpose to store the preprocess data.

Assign Data

After all of the preprocessing data has been completed, it’s time for storing all of the data into one variable. This process will use the pandas library to help create an unified dataset. This can help the researcher to call the dataset without running the data collection every time. This process also becomes important for the efficiency of the training process.

```
sampled_data = pd.concat([
    appledf.assign(DataFrame='appledf'),
    amazondf.assign(DataFrame='amazondf'),
    googledf.assign(DataFrame='googledf'),
    microsoftdf.assign(DataFrame='microsoftdf'),
    nvidiadf.assign(DataFrame='nvidiadf')
])
```

Implement in KNN Regression Model

The researcher is using the library from sklearn to build the model, splitting the dataset, scaling the data and using the evaluation to get the MSE from the training process.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.inspection import permutation_importance

grouped_data = sampled_data.groupby('DataFrame')

n_neighbors_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
  29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51]
weights_list = ['uniform', 'distance']
algorithm_list = ['ball_tree', 'kd_tree', 'brute']
p_list = [1, 2]
```

The researcher will create the model, train the model, and get the MSE value, prediction value, and actual price value. This process is declared in line 138 - 191. The researcher will call the sampled_data that the researcher creates from assigning the preprocessed data and then store it into a variable named grouped_data. This variable will group all of the data into separate names for each dataset. Then, declaring all of the parameters that the KNN Regression has. This will come very handy for the training process. The n_neighbors_list variable will become the n_neighbor parameter. The weights_list variable will become the weights parameter. The algorithm_list variable will become algorithm parameter and the p_list variable will become p.

```
132. X = group[['Open', 'High', 'Low', 'Volume']]
133.     y = group['Close']
134.
135.     X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
   test_size=0.3, random_state=42)
136.
137.     scaler = StandardScaler()
138.     X_train_scaled = scaler.fit_transform(X_train)
139.     X_test_scaled = scaler.transform(X_test)
```

After declaring all of the parameter variables and defining which is the feature and the target. Then begin the process of splitting the dataset. The researcher will split the data into a train and test split with the proportion of 90:10, 80:20, and 70:30 with randoming the splitting process. This could be possible by changing the value from the train_test_split library parameter in test_size, with the test_size of 0.1 means it will split the data into 90% training and 10% for testing, test_size of 0.2 means it will split the data into 80% training and 20% for testing, and test_size of 0.3 means it will split 70% into training and 30% for testing. After having the data splitted, the researcher found that scaling the data is one of the important steps to get a lower MSE for the training process. This standardization is a common practice to normalize input data (X) to ensure the input to have a mean of 0 and a standard deviation of 1. Because of the background process, the machine will consume the data without concerning spreading data, this standardization will create a restriction for the machine to train the data.

```
140.     for n_neighbors in n_neighbors_list:
141.         for weights in weights_list:
142.             for algorithm in algorithm_list:
143.                 for p in p_list:
144.
145.                     knn  =  KNeighborsRegressor(n_neighbors=n_neighbors,
   weights=weights, algorithm=algorithm, p=p)
```

Begin the process for the researcher to loop each of the declared parameter variables. This process will become the model parameters for the fitting process. By using loops, this process will become a one time trained model and not manually train the model for each different combination of parameters. The parameter contains the n_neighbors, weights, algorithm, and p value.

The n_neighbors parameter acts for determining the number of the neighbors. The number of neighbors determines the error from predicting the stock prices. K-NN Regression works by asking the nearest neighbor from the K points. The neighbor point that spreads from the K points means the model will create larger error points compared to the nearest neighbor.

The Weights parameter acts as how it influences the neighbors. In this experiment the researcher will try all of the weights parameters which are ‘uniform’ and ‘distance’. The uniform performs with influence on all of the neighbors' points equally. While the distance performs with taking the closer distance, the neighbor has greater influence.

The Algorithms parameter acts as an option for the K-NN to choose the best algorithm that returns a smaller error. There are several algorithms that will be tested in this experiment which are ‘ball_tree’, ‘kd_tree’, and ‘brute’. The ball_tree algorithm has a similar performance with the kd_tree algorithm, basically both of these algorithms use the tree-based algorithm that handles large datasets in the lower dimension. The main difference is that the kd_tree works a little bit faster than the ball_tree in some specific cases. The brute algorithm works by brute force searching every possible neighbor, this algorithm will cause the model to work more slowly but have a better accuracy for high-dimensional data.

The p parameter acts as an option for the machine to learn using the manhattan distance(p=1) or euclidean distance(p=2). This p parameter calculation can be seen at chapter 3. This experiment will just include the manhattan and euclidean distance. This choice was chosen because it is more mainstream for people to know and use the calculation distance. The manual calculation and the model calculation were actually similar.

```
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
```

Then, begin the fitting process. Fitting processes are to train the dataset with the input that has been standardized to achieve the target. The researcher will store the prediction by using one of the model features which is knn.predict(). With knn is the KNearestNeighborsRegressor with the declared parameter. This process will be inside of the parameter loop, because everytime the parameter changes, the predicted value will also change. This model will be evaluated by the MSE that will be achieved by using the library from sklearn.metrics import mean_squared_error. The MSE value will also be replaced until it achieved the lowest MSE for each of the parameters trained.

```
best_predictions = y_pred
actual_values = y_test

mean_best_predictions = best_predictions.mean()
mean_actual_values = actual_values.mean()
```

After getting the best_prediction, the researcher will take the average from all of the best_prediction values by using the mean() function from the python and storing it into the mean_best_prediction. The same procedure will be used again for the purpose of getting the average of the actual_value from all of the training process. This process will return the average for the best prediction and the actual value from the trained model. The value of this will be used as a comparison if the model is good enough to predict the actual values.

```
best_knn = KNeighborsRegressor(**best_params)
    best_knn.fit(X_train_scaled, y_train)
    result = permutation_importance(best_knn, X_test_scaled, y_test,
n_repeats=10, random_state=42)

    feature_importance = pd.DataFrame({
        'feature': X.columns,
        'importance': result.importances_mean
}).sort_values(by='importance', ascending=False)
```

The last thing to do is to get the feature importance for each of the dataset, by using the help from the library from sklearn. The researcher will fit the process one more time with the best_params that have been stored. This process will return the feature and the importance from the feature. The importance for each feature will be calculated by taking the average from the permutation calculation.

```
results.append({
        'dataset': name,
        'best_params': best_params,
        'best_mse': best_mse,
        'predictions': mean_best_predictions,
        'actual_values': mean_actual_values,
        'feature_importance': feature_importance
})
```

The result of this K-NN Regression training is to return the value of the dataset, best parameter, best MSE, average of prediction value, the average of actual value, and the feature imporance. This process is going to append all of the results into one array. This will help the researcher to import the result into excel. This excel soon will be compiled into a table for the experiment analysis.

This process will help the researcher to just call the variable of sampled_data into the training process. This process is usually called mapping the data. Basically, by creating a key and value, the key is ‘DataFrame’ and the value for each key is the processed dataset. This variable is called sampled_data that will now store the apple, amazon, google, microsoft and nvidia dataset after the cleaning process.

Implement in SVR Model

This implementation of SVR, actually has a similar implementation from the KNN Regression. The main difference is that instead of using the KNN Regression algorithm, the researcher used the SVR model.

```
from sklearn.svm import SVR

grouped_data = sampled_data.groupby('DataFrame')
kernel_list = ['poly', 'rbf', 'sigmoid', 'linear']
gamma_list = ['scale', 'auto', 0.1, 0.5, 0.9]
c_list = [0.1, 0.5, 0.7, 1.0]
epsilon_list = [0.1, 0.2, 0.5, 1.0]
```

After having the data preprocessed, the data is now ready to be implemented for the training purpose. The researcher will call the sampled_data that the researcher creates from assigning the preprocessed data and then store it into a variable named grouped_data. This variable will group all of the data into separate names for each dataset. Next, the researchers are going to declare all of the parameters from the model, for the purpose of one time training but cover all of the parameters that will be trained. The researcher creates multiple variables in an array for handling the parameter for the model, start from the kernel_list that will be used for the kernel parameter, gamma_list for the gamma parameter, c_list for the C parameter, and epsilon_list for the epsilon parameter.

```
results = []

for name, group in grouped_data:
    print(f"\nProcessing {name} dataset...")

    best_predictions = None
    actual_values = None
    best_params = {}
    best_mse = float('inf')
```

After creating all of the variables for the model parameter, the researcher will declare one more variable called results that have an empty array, this variable will be used to store the experiment result. After having the grouped_data variable, the researcher will use the loop to take out the name and the group. The group has the context of the column name for each dataset. The loop also works to always clear the 4 variables above that will be used to store the prediction value, actual value, best parameter, and the lowest MSE value.

```
X = group[['Open', 'High', 'Low', 'Volume']]
    y = group['Close']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

After creating all of the variables, it becomes the process to declare which column will be the input and which column will be the target value. The purpose of this experiment is to predict the value from the ‘Close’ column. The researcher will split the data into a train and test split with the proportion of 90:10, 80:20, and 70:30 with randoming the splitting process. This could be possible by changing the value from the train_test_split library parameter in test_size, with the test_size of 0.1 means it will split the data into 90% training and 10% for testing, test_size of 0.2 means it will split the data into 80% training and 20% for testing, and test_size of 0.3 means it will split 70% into training and 30% for testing. The SVR algorithm needs to have scaling data because this algorithm cannot handle a larger spreading dataset value rather than KNN.

```
for kernel in kernel_list:
        for gamma in gamma_list:
            for c in c_list:
                for epsilon in epsilon_list:

svr = SVR(kernel=kernel, gamma=gamma, C=c, epsilon=epsilon)
```

Then, the researcher will go through a loop for the parameter variables. This is the variable that contains the parameter list for the SVR parameter, starting from the kernel, gamma, C, and epsilon parameters. All of these parameters will affect the SVR performance to predict the stock close price. Some of these parameters, if not being handled properly can create a lot of time wasted on the training process.

Starting from the kernel parameter, the kernel works to define what hyperplane will be used to separate the data in feature space. This kernel parameter can be very fatal if the preprocessing of the data is not right and will cause the model to train inside the loop and not return the values from the training process. There are several kernels that will be tried in this experiment, such as the polynomial kernel, radial basis function (rbf) kernel, sigmoid kernel, and linear kernel. The polynomial kernel works by changing the structure of the hyperplane into a curve or loop. This kernel can be very useful if the data has gone too far from the hyperplane, this kernel can cover the lost data. Next, the radial basis function kernel or more likely to be said as rbf. This kernel works similarly to the KNN algorithm, by looking the range from one point data into the other point data and then grouped them based on the similarity. This kernel also can handle the data that are not in a straight line or maybe curved. Next is the sigmoid kernel, this kernel works like the human brain. This kernel can detect if the data have some "activated" data or are very likely to move a lot. The last but not least is the linear kernel, this kernel helps the model to create a single line to separate or predict the data. This linear kernel is reflected in the regression model.

Gamma parameter works like a spotlight, when gamma value is low the light will go spread and not point so if the gamma value is high the light will go focus on one point. This has the meaning that when the value of gamma is low, the model can reach almost all of the data and can read a spreading data while if the gamma value is high, then it will focus on the data that is being pointed and ignore those that have a further away data. Gamma can become a contender in training SVR because if the gamma value is too low then can create the model underfitting and if set too high can create the model overfitting because of too much pointing. The method to fix it can be done with the scale gamma, this can automatically calculate the best gamma for the model training purpose, the mathematical calculation is by dividing one with the number of features (X) times the variance of data. There's also auto gamma, this has a similarity to scale gamma but only by dividing one with the number of features (X), this means if there are a lot of features the gamma value will become smaller and if only have a few of features the gamma values will become higher.

Then comes the C parameter for the SVR, this parameter can decide how much the model can receive the error from the training process. Lower C value means that the model can receive error as long as it can turn the prediction model smoother and achieve a higher prediction. Higher C value means that the model has very strict rules that can't accept any error even if it can cause the model to overfit. This means that a higher C number can create the model to train more slowly because of the need for no error.

The last but not least, the epsilon parameter. This parameter creates a margins error where if the prediction is close enough then the result is good enough. Lower epsilon value means that the model will try to get very close into the actual data points. This is more sensitive and aims for smaller errors. This means that a lower epsilon can create a better prediction and will take more time to train the model. Higher epsilon value means that the model won't care about small errors and the model will care more about the long range data more than focusing on one point of data.

```
svr.fit(X_train_scaled, y_train)

y_pred = svr.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
```

Then, begin the fitting process. Fitting processes are to train the dataset with the input that has been standardized to achieve the target. The researcher will store the prediction by using one of the model features which is svr.predict(). With svr is the SVR with the declared parameter. This process will be inside of the parameter loop, because everytime the parameter changes, the predicted value will also change. This model will be evaluated by the MSE that will be achieved by using the library from sklearn.metrics import mean_squared_error. The MSE value will also be replaced until it achieved the lowest MSE for each of the parameters trained.

After getting the best_prediction, the researcher will take the average from all of the best_prediction values by using the mean() function from the python and storing it into the mean_best_prediction. The same procedure will be used again for the purpose of getting the average of the actual_value from all of the training process. This process will return the average for the best prediction and the actual value from the trained model. The value of this will be used as a comparison if the model is good enough to predict the actual values.

```
best_svr = SVR(**best_params)
best_svr.fit(X_train_scaled, y_train)
result = permutation_importance(best_svr, X_test_scaled, y_test, n_repeats=10, random_state=42)

        feature_importance = pd.DataFrame({
            'feature': X.columns,
            'importance': result.importances_mean
        }).sort_values(by='importance', ascending=False)
```

With the same method that was used in the previous implementation, the researcher will implement the search of feature importance for the SVR algorithm. This process will return the feature column and the importance from the column. The calculation will be taken from the library and the researcher will take the average of each calculation. This will be stored inside a data frame so it will map for each dataset which feature affected the most the training process.

```
results.append({
           'dataset': name,
           'best_params': best_params,
           'best_mse': best_mse,
           'predictions': mean_best_predictions,
           'actual_values': mean_actual_values,
           'feature_importance': feature_importance
  })
```

The result of this SVR training is to return the value of the dataset, best parameter, best MSE, average of prediction value, the average of actual value, and the feature importance. This process is going to append all of the results into one array. This will help the researcher to import the result into excel. This excel soon will be compiled into a table for the experiment analysis.

```
!pip install openpyxl==3.1.2
       data = []

       for result in results:
           print(f"\nDataset: {result['dataset']}")
           print(f"Best Parameters: {result['best_params']}")
           print(f"Lowest MSE: {result['best_mse']}")
           print(f"Feature importance : {result['feature_importance']}")
           print(f"First 5 Predictions: {result['predictions']}")
           print(f"First 5 Actual Values: {result['actual_values']}")

           data.append({
               'Dataset': result['dataset'],
               'Best Params': result['best_params'],
               'Lowest MSE': result['best_mse'],
               'Feature importance' : result['feature_importance'],
               'First 5 Predictions': result['predictions'],
               'First 5 Actual Values': result['actual_values'],
           })

       results_df = pd.DataFrame(data)
       output_file = 'name file'
       results_df.to_excel(output_file, index=False)
       print(f"\nResults saved to {output_file}")
```

The next process is to change the format from an array into an excel. The researcher will use the openpyxl library to compile the data frame into excel format that can be downloaded from the folder in google colab. The table and structured data will help to get a better analysis from the SVR and K-NN Regression implementation. The result table can be seen in the result section in the next section.

Results

After doing all of the implementation, now the researcher will be able to do the analysis for this experiment by splitting the dataset. This process will determine the best algorithm to minimize the error score for prediction of 5 most active stocks in 2024, which are Amazon, Apple, Google, Microsoft and Nvidia. The experiment is conducted by splitting 3 kinds of numbers of dataset. The first split is used for 90% of dataset to training and 10% of dataset to testing. The others use 80:20 and 70:30. The table belows shows the result of splitting 90:10 of the dataset. The feature importance column will represent the importance of each training column. The lower importance value tells that column is not that important to the training process.

Table 2. KNN Split 90:10

KNN SPLIT 90:10				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.2741502069	1. Low: 0.001576 2. High: 0.001245 3. Open: 0.000494 4. Volume: 0.000324	21.01281725
appledf	{'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.4266102043	1. Volume: 0.003999 2. Low: 0.000553 3. High: 0.000375 4. Open: 0.000128	31.32673755
googledf	{'n_neighbors': 1, 'weights': 'uniform', 'algorithm': 'ball_tree', 'p': 1}	0.6395543141	1. Volume: 0.006672 2. Low: 0.000647 3. High: 0.000438 4. Open: 0.000174	35.5609575
microsoftdf	{'n_neighbors': 15, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.6959328848	1. Volume: 0.001545 2. High: 0.001472 3. Low: 0.001445 4. Open: 0.000790	40.54539607
nvidiadf	{'n_neighbors': 9, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.06410010434	1. Volume: 0.002692 2. High: 0.001115 3. Low: 0.000926 4. Open: 0.000924	5.382165225

The researcher found that using the K-NN Regression with the splitting of 90:10, the nvidia dataset had the lowest MSE with 0.064 with the average prediction of 5.382 and the average actual value of 5.404. The researcher also found that the highest MSE was achieved from the microsoft dataset with the MSE of 0.696 with the average prediction of 40.545 and the average actual value of 40.559. Amazon's dataset has the lowest MSE that of 0.274 with the average value prediction of 21.012 and the average of the actual value is 21.011. Apple’s dataset has the lowest MSE of 0.437 with the average prediction of 31.327 with the average actual value of 31.417. The last is Google’s dataset has the lowest MSE of 0.640 with the average prediction of 35.561 and the average actual value of 35.525.

The table also showed that the volume column most likely affects more than the other column, this affects the nvidia, microsoft, google, and apple dataset. The importance value from each column doesn’t have a larger difference. The order column for each dataset has a similar likelihood that the most important is volume rather than highest price or lowest price and the least important is the open price. The biggest difference is achieved by amazon with the order from lowest price, highest price, open price and the volume.

Table 3. KNN Split 80:20

KNN SPLIT 80:20				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.3140281532	1. Low: 0.001048 2. High: 0.000978 3. Volume: 0.000636 4. Open: 0.000384	21.60574476
appledf	{'n_neighbors': 7, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.36083256	1. Volume: 0.003590 2. Low: 0.000567 3. High:0.000380 4. Open: 0.000144	30.74431545
googledf	{'n_neighbors': 1, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.516016571	1. Volume: 0.005421 2. Low: 0.000706 3. High: 0.000460 4. Open: 0.000155	36.80304426

microsoftdf	{'n_neighbors': 15, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.5293455065	1. High: 0.001729 2. Low: 0.001459 3. Volume: 0.001127 4. Open: 0.000981	39.84280256
nvidiadf	{'n_neighbors': 9, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.142005531	1. High:0.003778 2. Low: 0.003540 3. Open: 0.003368 4. Volume: 0.000880	6.063147949

The researcher also found that for splitting 80:20, the lowest MSE was achieved by the nvidia dataset with the MSE of 0.142 with the average prediction of 6.063 and the average actual value of 6.067. The highest MSE was achieved by the microsoft dataset with the MSE of 0.529 with the average prediction of 39.843 and the average actual value of 39.860. This 80:20 splitting creates the same result from the 90:10 splitting. The difference is for the nvidia dataset, 90:10 splitting achieved a lower MSE with a lower prediction and the actual value. The microsoft dataset created a lower MSE, prediction, and the actual value with the splitting of 80:20. This splitting created a lower MSE except for the amazon and the nvidia dataset. The Amazon dataset achieved the lowest MSE that of 0.314 with the average value prediction of 21.606 and the average of the actual value is 21.632. Apple achieved the lowest MSE of 0.361 with the average prediction of 30.744 with the average actual value of 30.790. Google achieved the lowest MSE of 0.516 with the average prediction of 36.803 and the average actual value of 36.794.

The researcher also found that the importance from each dataset is kindly different with the 90:10 splitting. While for the amazon dataset the order of importance is not changing, but for the other dataset there are some of different orders of importance. Apple dataset and Google dataset now have the same importance order which is the most important column is volume. Microsoft and Nvidia dataset have the order of higher price as the most important column.

Table 4. KNN Split 70:30

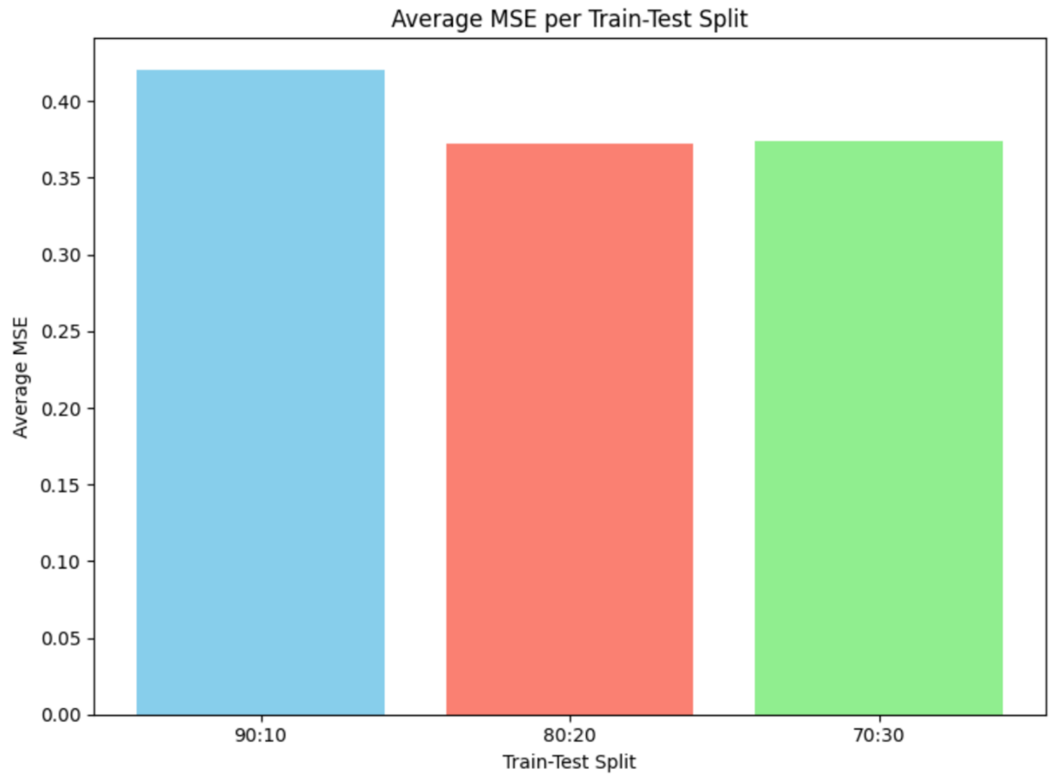
KNN SPLIT 70:30				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'n_neighbors': 7, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.395151449	1. Volume: 0.001477 2. High: 0.001189 3. Low: 0.001079 4. Open: 0.000586	21.7520017

appledf	{'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.3614172026	1. Volume: 0.004197 2. Low: 0.000667 3. High: 0.000528 4. Open: 0.000292	30.40118798
googledf	{'n_neighbors': 3, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.4738783371	1. Volume: 0.008361 2. Low: 0.000761 3. High: 0.000485 4. Open: 0.000201	37.00775162
microsoftdf	{'n_neighbors': 13, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.4968379384	1. High: 0.002195 2. Low: 0.002088 3. Open: 0.001758 4. Volume: 0.001030	39.03848342
nvidiadf	{'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'ball_tree', 'p': 1}	0.1428625974	1. High: 0.001536 2. Low: 0.001461 3. Open: 0.001002 4. Volume: 0.000953	6.076913521

The last splitting for K-NN Regression was 70:30. Both of the highest and lowest MSE was achieved from the microsoft (highest MSE) and the nvidia (lowest MSE) dataset. The nvidia achieved the lowest MSE of 0.143 with the average prediction of 6.077 and the average actual value of 6.087. Microsoft achieved the lowest MSE of 0.497 with the average prediction of 39.038 and the average actual value of 39.056. There is no difference MSE between splitting 70:30 and 80:20 for the nvidia dataset. This 70:30 splitting also decreased the MSE value of the microsoft dataset by 0.03 point. Amazon achieved the lowest MSE of 0.395 with the average value prediction of 21.752 and the average of the actual value is 21.801. Apple has the lowest MSE of 0.361 with the average prediction of 30.401 with the average actual value of 30.423. Google has the lowest MSE of 0.474 with the average prediction of 37.008 and the average actual value of 37.029.

The feature importance for each dataset for this splitting has a very unique result. If the most important column is filled with the volume column, then the least important column is filled with the open price column. If the top order is filled with the highest price column, then the last order is filled with the volume column. The importance column value for each dataset is not have a very big difference, this could be means that the column will not really affected the process of training process.

```
[0.420069542888, 0.3724456643399999, 0.3740295049]
```



Gambar 3. Chart KNN Splitting

Based on the chart., the researcher now begins to use the chart to see from the three splitting which has the lowest MSE. This will be done by taking the average MSE for each splitting from 90:10, 80:20, and 70:30. The chart showed that splitting 80:20 has the lowest average MSE compared to the other with the value of 0.372, while 90:10 has the average MSE value of 0.420 and 70:30 has the average MSE value of 0.374. With the difference of 0.002 splitting 80:20 is lower than the 70:30. Based on the experiment, the researcher also found that more neighbors creates a larger MSE value.

Table 5. SVR Split 90:10

SVR SPLIT 90:10				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.1776541957	1. Low: 4.035104e-01 2. High: 3.803245e-01 3. Open: 2.843765e-02 4. Volume: 1.106510e-07	21.04668678

appledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.3452256831	1. Low: 3.490715e-01 2. High: 3.482489e-01 3. Open: 6.386655e-02 4. Volume: 9.579132e-09	31.36192963
googledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.193030105	1. Low: 4.411789e-01 2. High: 3.818021e-01 3. Open: 1.512573e-02 4. Volume: -1.916336e-07	35.53468015
microsoftdf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.1181653236	1. Low: 6.381247e-01 2. High: 6.312745e-01 3. Open: 2.910788e-02 4. Volume: 5.140592e-07	40.55245333
nvidiadf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.02549972998	1. Low: 5.700931e-01 2. High: 4.468522e-01 3. Open: 1.788423e-04 4. Volume: -2.650884e-09	5.39012087

Moving on to the SVR algorithm result, with the same splitting process. The researcher found that the nvidia dataset achieved the lowest MSE with the value of 0.025 with the average prediction of 5.390 and the average actual value of 5.404. The highest MSE value was achieved by apple with the MSE value of 0.345 with the average prediction of 31.362 with the average actual value of 31.417. The second lowest MSE was achieved by microsoft dataset with the MSE of 0.118 with the average prediction of 40.552 and the average actual value of 40.559. The amazon dataset comes in third with the value MSE of 0.177 with the average value prediction of 21.047 and the average of the actual value is 21.011. The second highest MSE was achieved by google dataset with the value of 0.193 with the average prediction of 35.535 and the average actual value of 35.525.

Different from the KNN Regression, the SVR has a very unique feature importance. SVR turns out to return the same order for each of the dataset. The order is lowest price, highest price, open price, and the volume. The importance value for each column is relatively small. This value returned because of the scalar transformation.

Table 6. SVR Split 80:20

SVR SPLIT 80:20				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.163655885	1. Low: 3.769894e-01 2. High: 3.462374e-01 3. Open: 4.142285e-02 4. Volume: -1.125700e-07	21.64371554
appledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.308282911	1. High: 3.302200e-01 2. Low: 3.173057e-01 3. Open: 7.553654e-02 4. Volume: 5.297395e-08	30.76592538
googledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.2184595914	1. Low: 4.201901e-01 2. High: 3.606451e-01 3. Open: 2.491814e-02 4. Volume: -6.926520e-09	36.81529712
microsoftdf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.1136475094	1. Low: 5.967120e-01 2. High: 5.895732e-01 3. Open: 1.425800e-02 4. Volume: 5.762871e-07	39.85141641
nvidiadf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.0470130312	1. Low: 5.435821e-01 2. High: 4.216750e-01 3. Open: 7.610533e-04 4. Volume: -2.383338e-08	6.061319855

Splitting 80:20 doesn't really change the value a lot. The lowest and the highest MSE was still achieved by the same dataset. While nvidia dataset makes a progressif in the MSE value, the Apple dataset makes a downfall in the MSE. The nvidia dataset achieved the value MSE of 0.047 with the average prediction of 6.061 and the average actual value of 6.067. The apple dataset achieved the MSE value of 0.308 with the average prediction of 30.766 with the average actual value of 30.790.

Beside these two dataset, the last three dataset have the same ranks in the lowest - highest MSE values. Starting from microsoft, amazon, and google dataset. Amazon achieved MSE value of 0.164 with the average value prediction of 21.644 and the average of the actual value is 21.632. Microsoft has achieved MSE of 0.113 with the average prediction of 39.851 and the average actual value of 39.858. Google achieved MSE of 0.218 with the average prediction of 36.815 and the average actual value of 36.794. The order of feature importance also does not have a significant difference, but for the apple dataset it now has the highest price as the most important column and volume is still as the last order of important column.

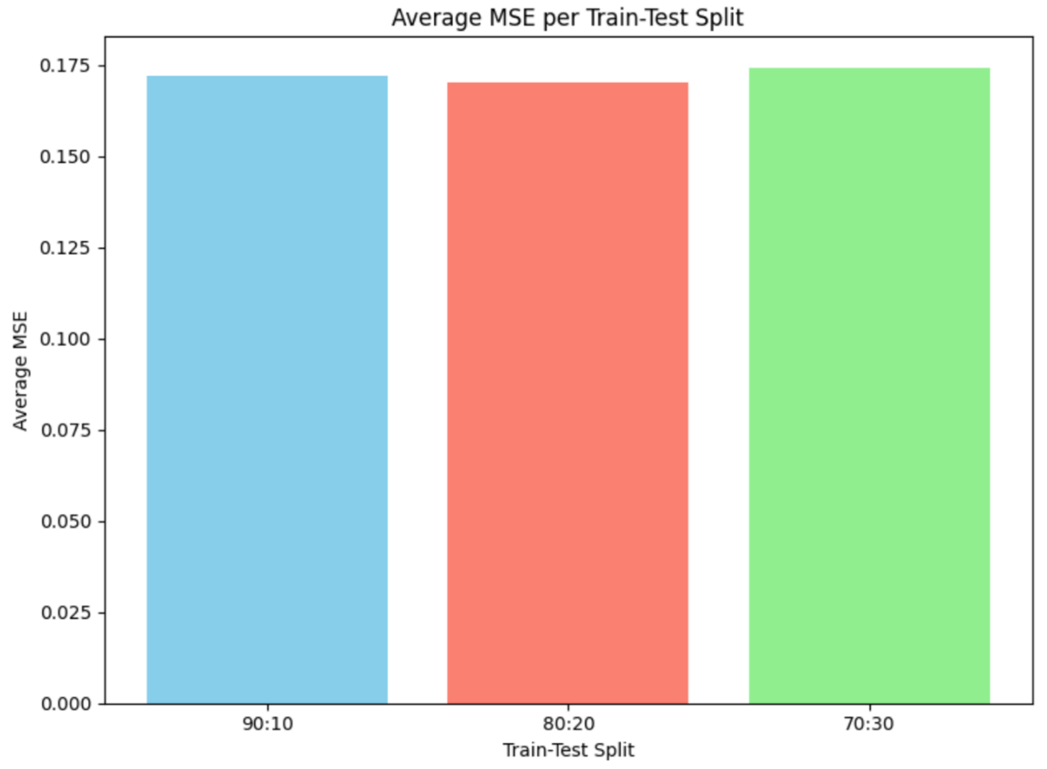
Table 7. SVR Split 70:30

SVR SPLIT 70:30				
Dataset	**Best Params**	**Lowest MSE**	**Feature Importance**	**Predictions**
amazondf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.1607077579	1. Low: 3.580864e-01 2. High: 3.473665e-01 3. Open: 5.602767e-02 4. Volume: 9.262100e-08	21.80700004
appledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.304324045	1. Low: 3.189718e-01 2. High: 3.060397e-01 3. Open: 8.901792e-02 4. Volume: 1.557502e-08	30.41496266
googledf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.2342317462	1. Low: 4.104593e-01 2. High: 3.472950e-01 3. Open: 3.580953e-02 4. Volume: 2.518317e-07	37.03370425
microsoftdf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.1188050981	1. High: 0.568704 2. Low: 0.547017 3. Open: 0.004946 4. Volume: 0.000001	39.04496115
nvidiadf	{'kernel': 'linear', 'gamma': 'scale', 'C': 1.0, 'epsilon': 0.1}	0.05233456401	1. Low: 5.424180e-01 2. High: 3.944767e-01 3. Open: 3.525991e-03 4. Volume: -4.152409e-08	6.072518284

The last split for the SVR algorithm was 70:30. The researcher found that the ranks from lowest - highest MSE were still the same with the previous splitting of 90:10 and 80:20. Starting from the lowest MSE that was achieved from the nvidia dataset with the MSE value of 0.052 with the average prediction of 6.072 and the average actual value of 6.087. The second lowest MSE was achieved from the microsoft dataset with the MSE value of 0.119 with the average prediction of 39.045 and the average actual value of 39.056. The third lowest MSE was achieved by amazon dataset with the MSE value of 0.161 with the average value prediction of 21.807 and the average of the actual value is 21.801. The fourth lowest MSE was achieved by google dataset with the MSE value of 0.234 with the average prediction of 37.034 and the average actual value of 37.029. The highest MSE value was achieved by apple with the MSE value of 0.304 with the average prediction of 30.415 with the average actual value of 30.423.

The researcher also found that there is a slight difference with the splitting of 80:20. While splitting 70:30, the main difference was achieved by the microsoft dataset with the same order from apple dataset in splitting 80:20. The order was highest price, lowest price, open price, and the volume. The importance value for each column doesn’t have a big difference.

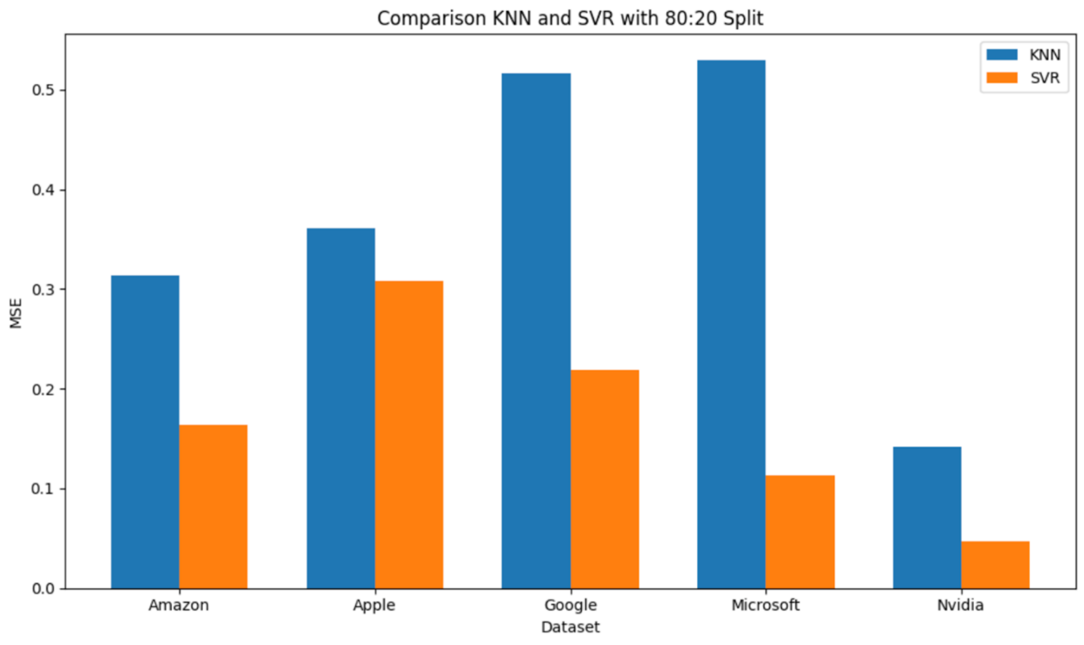
```
[0.17191500747600003, 0.1702117856, 0.174080642242]
```



Gambar 4. Chart SVR Split

Based on the chart, the researcher now begins to use the chart to see from the three splitting which has the lowest MSE. This will be done by taking the average MSE for each splitting from

90:10, 80:20, and 70:30. The chart showed that splitting 80:20 has the lowest average MSE compared to the other with the value of 0.170 while 90:10 has the average MSE value of 0.172 and 70:30 has the average MSE value of 0.174. With the difference of 0.002 splitting 80:20 is lower than the 90:10. The same parameter for each dataset proved that changing parameter doesn't really affect the model results.



Gambar 5. Comparison KNN and SVR

Based on figure 3 and figure 4, the researcher found that both K-NN and SVR have an average lower MSE by using the 80:20 splitting. This means that now, the researcher can compare which algorithm suits the most to predicting stock movement. Based on the chart in figure 4.5 the researcher found that between KNN and SVR the SVR dominates by having the lowest MSE for each dataset. The difference between KNN Regression and SVR is very far, this means that SVR can predict the Amazon, Apple, Google, Microsoft, and Nvidia dataset better than the KNN Regression algorithms. Amazon, if trained using KNN, achieves MSE for 0.314, while using SVR achieves 0.163. Apple, if trained using KNN, achieves MSE for 0.361, while using SVR achieves 0.308. Google, if trained using KNN, achieves MSE for 0.516, while using SVR achieves 0.218. Microsoft, if trained using KNN, achieves MSE for 0.529, while using SVR achieves 0.114. Nvidia, if trained using KNN, achieves MSE for 0.142, while using SVR achieves 0.047.

Discussion

In this study, the researcher found that SVR dominates KNN Regression for each dataset training result in predicting stock closing prices. This could happen because the process of SVR is more complicated than KNN Regression. Starting from the processing of the data, SVR could take more time to calculate every possibility than could happen for each kernel parameter. Different with KNN Regression where this algorithm only calculates from one point to another point by asking only what value that the other neighbors have.

K-Nearest Neighbors (KNN) Regression often got outperforms by Support Vector Regression (SVR) when predicting stock closing prices. This could have happened because of a better approach optimization from SVR to fine-tunes the model for the better performance, while KNN Regression used a brute-force strategy to approach the best prediction. With a complex dataset, the simplicity of KNN’s method can lead to suboptimal results.

A compatibility for each column may affect the performance of the models. Each feature column may have a different value of importance. The most value of importance marks the compatibility for each feature. The SVR algorithm have a very low importance value for each splitting and the dataset.

A notable aspect of both methods is the needs of data scaling. The SVR algorithm needs scaling. Scaling is essential as the algorithm struggles with datasets that have widely varying values. Scaling for KNN Regression helps to prevent excessive errors. However, even with scaling larger errors are more likely to come from the KNN Regression compared to SVR.

CONCLUSION

Based on the experiment, the researcher draws a conclusion that the best model to minimize error for stock closing prices resulting for the lowest MSE has been achieved from the Nvidia stocks with value of 0.114 using SVR algorithm. This means that SVR can easily predict Nvidia stock closing price with the most minimum error. This process could happen because of the superior chosen optimization from the background process from the model. Selecting the best kernel, gamma value, C, and epsilon parameter can be very impactful to do to have the best minimum error to predict. KNN Regression actually also has a very good model to predict stock closing prices, but compared to SVR in minimum error KNN Regression lost to SVR with the absolute loss.

After having the experiment result, the researcher concludes that SVR has a better prediction of stock closing prices in Amazon, Apple, Google, Microsoft, and Nvidia dataset rather than KNN Regression. Even KNN Regression also achieved a low error, but compared to SVR, KNN Regression still can’t beat SVR in predicting stock closing prices. This could happen because KNN usually was utilized to create a classification rather than a regression, so this can be the main reason why KNN Regression is not suggested to become a regression model.

The experiment also answered that the Nvidia dataset achieved the lowest MSE from all of the other dataset. The feature that affects the most is the lowest price from the feature column. The other thing that is interesting are the features don't really affect the model training process because of the low value of importance. Especially for the SVR algorithm which has the best algorithm to predict stock closing price. In other words, even though the feature doesn't really affect the MSE values, the parameter for SVR definitely carried the MSE becoming low. Using a linear kernel, a 'scale' value for gamma, a regularization parameter (C) of 1.0, and the default epsilon value of 0.1 ensures a better optimization process during SVR training.

Suggestions for the next research are trying to suit the best model compatibility to create a regression type of prediction, as well as trying to handle more dataset rather than only 5 dataset. These recommendations are to identify the models that can better adapt to diverse stock patterns.

DAFTAR PUSTAKA

[1] R. K. Nayak, D. Mishra, and A. K. Rath, "A Naïve SVM-KNN based stock market trend reversal analysis for Indian benchmark indices," Appl. Soft Comput., vol. 35, pp. 670–680, Oct. 2015, doi: 10.1016/j.asoc.2015.06.040.

[2] V. Sarala, "Stock Market Trend Prediction Using K-Nearest Neighbor (KNN) Algorithm," J. Eng. Sci., vol. 13, no. 08, 2022.

[3] K. Nakagawa, M. Imamura, and K. Yoshida, "Stock Price Prediction with Fluctuation Patterns Using Indexing Dynamic Time Warping and $$k^*$$-Nearest Neighbors," in New Frontiers in Artificial Intelligence, S. Arai, K. Kojima, K. Mineshima, D. Bekki, K. Satoh, and Y. Ohta, Eds., Cham: Springer International Publishing, 2018, pp. 97–111. doi: 10.1007/978-3-319-93794-6_7.

[4] O. D. Madeeh and H. S. Abdullah, "An Efficient Prediction Model based on Machine Learning Techniques for Prediction of the Stock Market," J. Phys. Conf. Ser., vol. 1804, no. 1, p. 012008, Feb. 2021, doi: 10.1088/1742-6596/1804/1/012008.

[5] M. Ghosh and R. Gor, "STOCK PRICE PREDICTION USING SUPPORT VECTOR REGRESSION AND K-NEAREST NEIGHBORS: A COMPARISON," International Journal of Engineering Science Technologies, vol. 6, no. 4, Art. no. 4, Jul. 2022, doi: 10.29121/ijoest.v6.i4.2022.354.

[6] M. Rehman, M. Fuzail, M. K. Abid, and N. Aslam, "Financial Prices Prediction of Stock Market using Supervised Machine Learning Models," VFAST Transactions on Software Engineering, vol. 11, no. 2, Art. no. 2, May 2023, doi: 10.21015/vtse.v11i2.1439.

[7] Y. Huang, "Research on the Google Stock Price Prediction Based on SVR, Random Forest, and KNN Models," Highlights in Business, Economics and Management, vol. 24, pp. 1054–1058, Jan. 2024, doi: 10.54097/n8hxqx19.

[8] S. Islam, Md. S. Sikder, Md. F. Hossain, and P. Chakraborty, "Predicting the daily closing price of selected shares on the Dhaka Stock Exchange using machine learning techniques," SN Bus Econ, vol. 1, no. 4, p. 58, Mar. 2021, doi: 10.1007/s43546-021-00065-6.

[9] J. Kaliappan, K. Srinivasan, S. Mian Qaisar, K. Sundararajan, C.-Y. Chang, and S. C, "Performance Evaluation of Regression Models for the Prediction of the COVID-19

Reproduction Rate," Front. Public Health, vol. 9, Sep. 2021, doi: 10.3389/fpubh.2021.729795.

[10] H. Zhao and Y. Chen, "A Comparative Study for Temperature Prediction by Machine Learning and Deep Learning," in 2023 International Conference on Intelligent Computing and Control (IC&C), Feb. 2023, pp. 77–84. doi: 10.1109/IC-C57619.2023.00020.