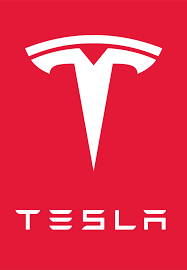
Tesla Stock Price EDA

COMP-4948 Machine Learning Assignment 2



# Victor Li

A00891907

Set U

# 

# Introduction

The objective of this Exploratory Data Analysis (EDA) is to understand the patterns and relationships in the Tesla Stock Price dataset. The dataset provides daily stock prices for Tesla from June 29, 2010, to September 3, 2021. The focus of this analysis will be on the "Adjusted Close" (Adj Close) price, which is the closing price adjusted for dividends and stock splits.

# Data Overview

The dataset contains the following columns:

* Date - The date of the stock trading in the format YYYY-MM-DD
* Open - The opening stock price on the given date
* High - The highest stock price reached on the given date
* Low - The lowest stock price reached on the given date
* Close - The closing stock price on the given date
* Volume - The number of shares traded on the given date
* Adj Close - The adjusted closing stock price on the given date (our target variable)

# Data Cleaning and Preprocessing

Before diving into the EDA, the dataset was checked for missing values, duplicates, and data type inconsistencies. No missing values or duplicates were found, and all data types were consistent with their respective columns.

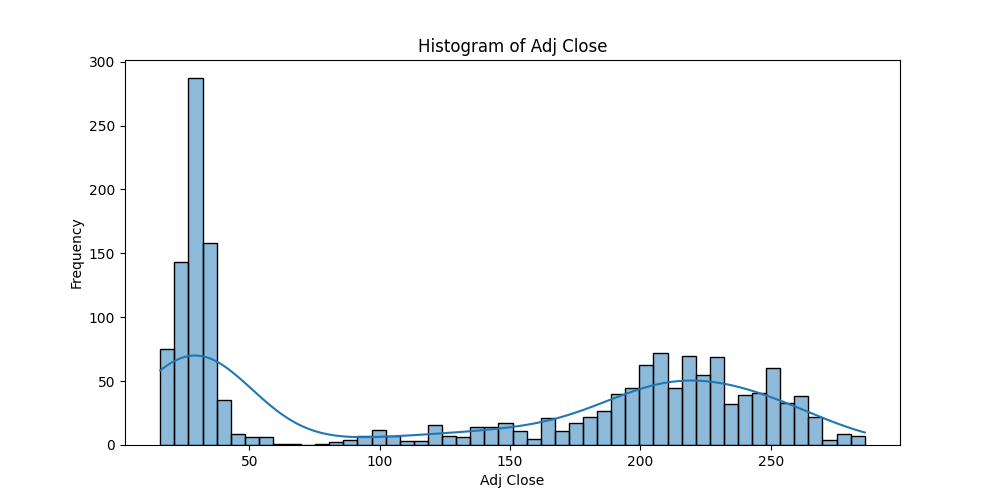
The 'Date' column was dropped as it doesn’t have any relevance to predicting the Adj Close.

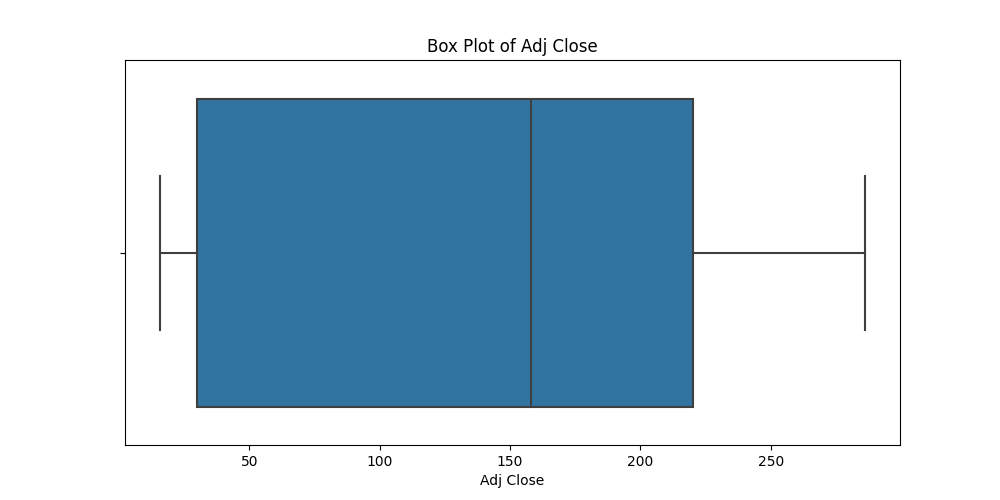
# 

# Univariate Analysis

## Distribution of the Target Variable (Adj Close)

The distribution of the target variable, "Adj Close," was analyzed using a histogram and a box plot. The histogram revealed a right-skewed distribution, indicating that the majority of the adjusted closing prices were on the lower side. The box plot showed a few outliers on the higher end of the distribution.

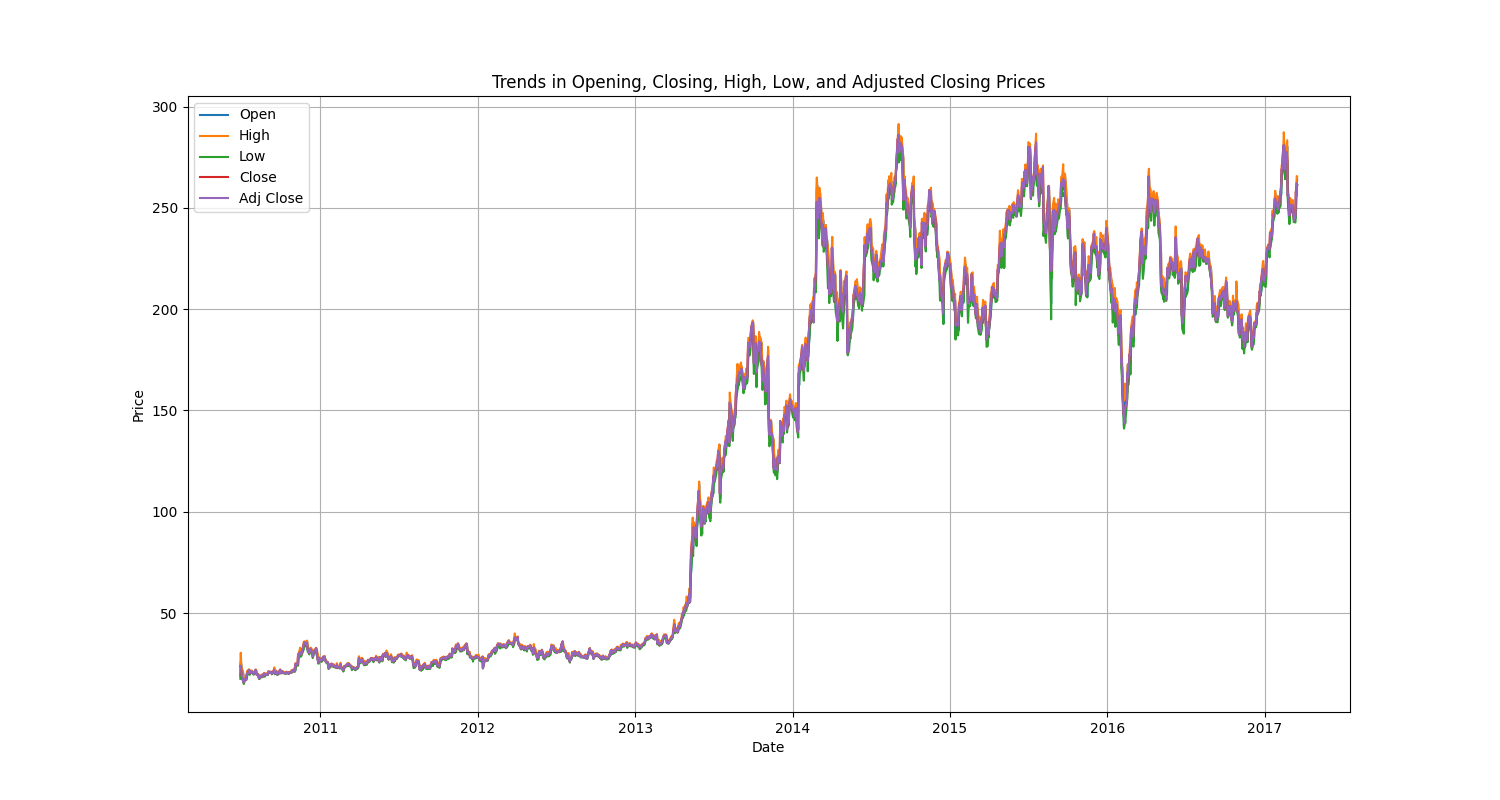




## 

## Stock Price Trends

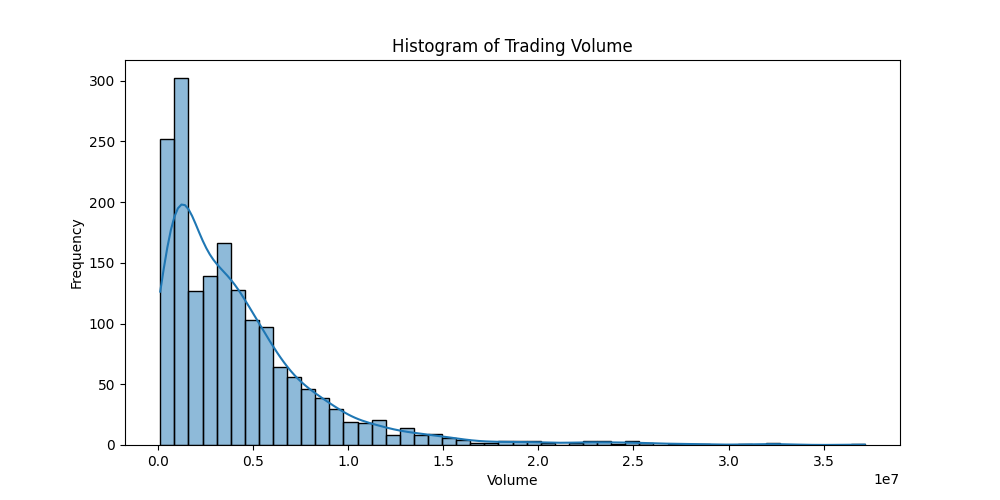
Line plots were used to visualize the trends in the opening, closing, high, low, and adjusted closing prices. The plots revealed a strong upward trend in the stock prices over time, with occasional periods of high volatility.

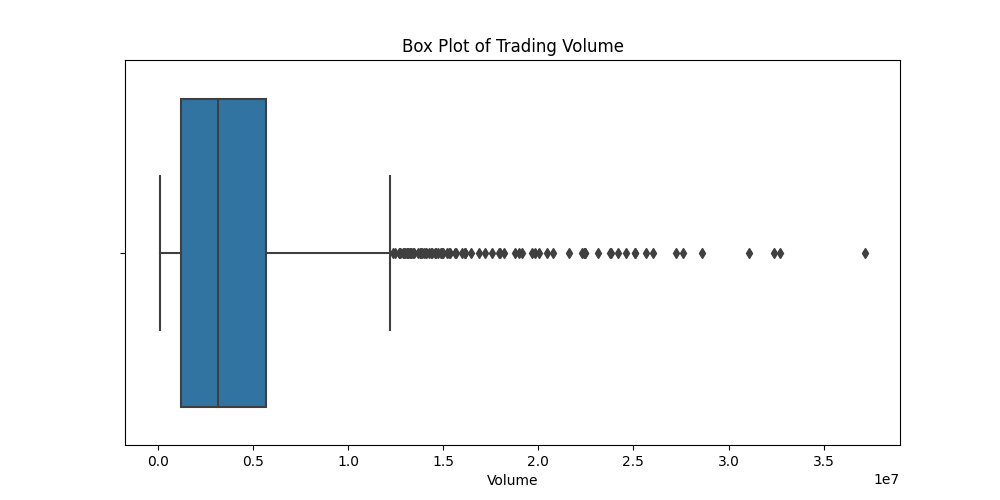


## 

## Trading Volume Analysis

A histogram and a box plot were used to analyze the distribution of the trading volume. The histogram indicated a right-skewed distribution, while the box plot showed the presence of several outliers on the higher end of the distribution. This suggests that there were days with exceptionally high trading volumes.



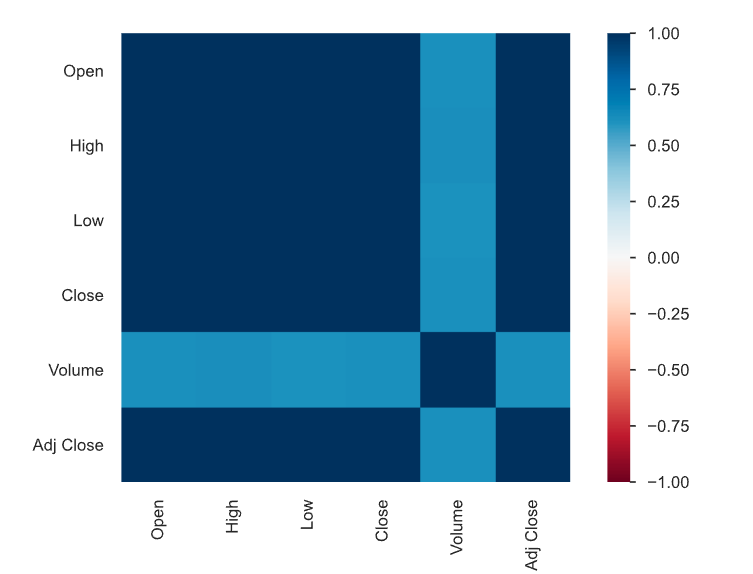


# 

# Bivariate Analysis

## Correlations between Features

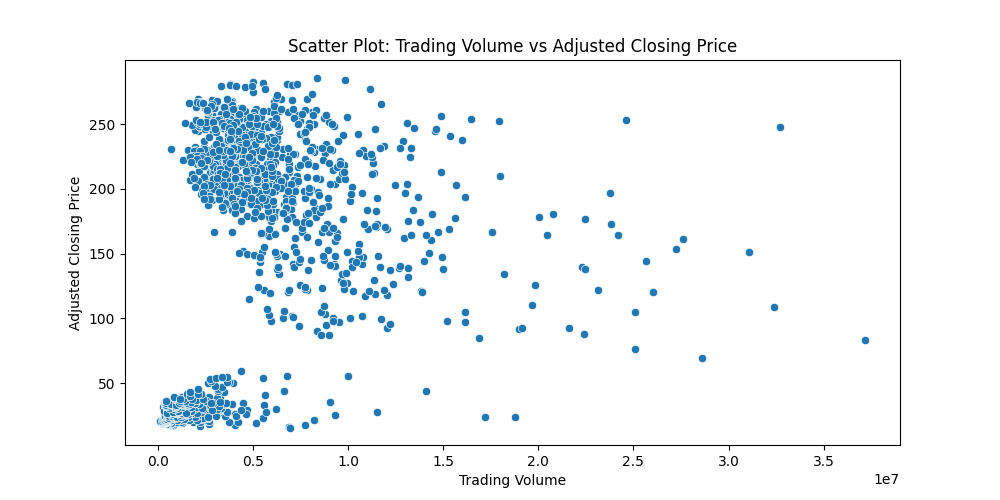
A correlation matrix was created to examine the relationships between the features. The matrix shows strong positive correlations between the opening, closing, high, low, and adjusted closing prices. This is expected, as these variables are closely related in the stock market.



## 

## Relationship between Trading Volume and Adjusted Close Price

A scatter plot was created to see the relationship between trading volume and the adjusted closing price. No clear trend exists, so we can say that that trading volume does not have a strong impact on the adjusted closing price.

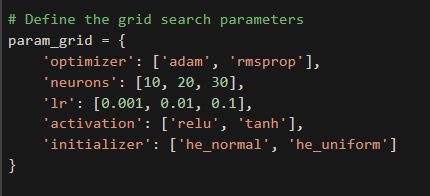


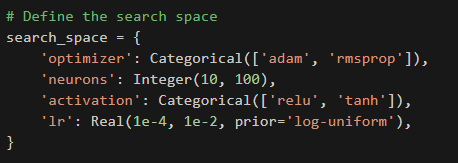
# Model Development

Models that were used include the OLS model, Neural Network model, and a Neural Network model with optimized parameters from a Grid Search.

## Tuning

Grid Search was used to obtain optimized parameters to use in conjunction with the Neural Network model.





The following are the optimized parameters:



# Model Comparison

## OLS Model and Neural Network Model

# 

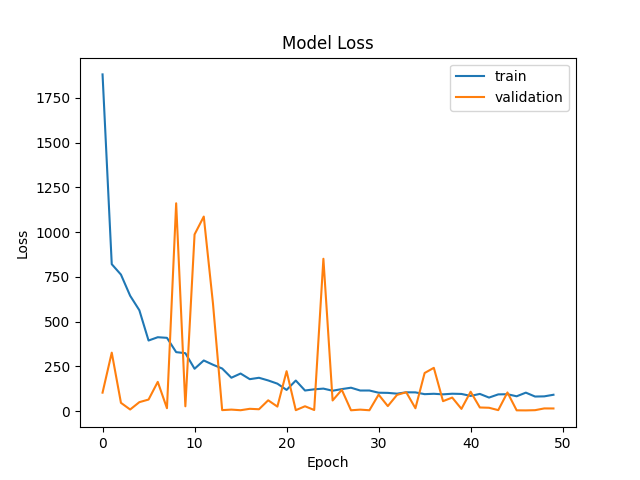
# 

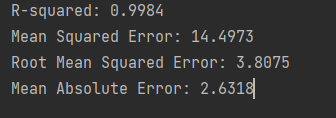
## 

## 

## 

## Neural Network Model (Optimized)





## 

## Comparison Chart

| Model | R-squared | Mean Squared Error | Root Mean Squared Error | Mean Absolute Error |
| --- | --- | --- | --- | --- |
| OLS | 0.9907628689476281 | 86.14443299430685 | 9.281402533793417 | 7.69549865663205 |
| Neural Network | **0.9996734379634916** | **1.0549810373293897** | **1.7451304791336968** | 3.0454803892014057 |
| Neural Network (Optimized) | 0.9984 | 14.4973 | 3.8075 | **2.6318** |

From creating OLS, Neural Network, and a grid search optimized parameter Neural Network model, the regular Neural Network model performs the best. The optimized version only beats it in its Mean Absolute Error metric.

# Conclusion

This exploratory data analysis investigates the Tesla Stock Price dataset, with a focus on the adjusted closing price. Findings include:

* A strong upward trend in the stock prices, with occasional periods of high volatility.
* A right-skewed distribution for both the adjusted closing price and trading volume.
* Strong positive correlations between the opening, closing, high, low, and adjusted closing prices.
* No clear relationship between trading volume and the adjusted closing price.
* No evident seasonality in the adjusted closing price.
* The presence of autocorrelation in the adjusted closing price data.

# Code Appendix

| import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression, ElasticNet  from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor  from sklearn.tree import DecisionTreeRegressor  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense  from keras.callbacks import EarlyStopping, ModelCheckpoint  from keras.models import load\_model  import warnings  from pathlib import Path  import statsmodels.api as sm  import numpy as np  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  from sklearn import metrics  from keras.wrappers.scikit\_learn import KerasClassifier  from sklearn.model\_selection import GridSearchCV  from keras.wrappers.scikit\_learn import KerasRegressor  warnings.simplefilter(action='ignore', category=FutureWarning)  warnings.simplefilter(action='ignore', category=UserWarning)  def plot\_loss\_and\_metrics(model\_name, y\_true, y\_pred):  print(f"====Model {model\_name} ====")  mse = mean\_squared\_error(y\_true, y\_pred)  rmse = np.sqrt(mse)  mae = mean\_absolute\_error(y\_true, y\_pred)  r2 = r2\_score(y\_true, y\_pred)  print(f"MSE: {mse:.4f}")  print(f"RMSE: {rmse:.4f}")  print(f"MAE: {mae:.4f}")  print(f'R^2 Score: {r2:.4f}')  print(f"==== # ====")  # Load and prepare dataset  PATH = Path("Tesla.csv")  df = pd.read\_csv(PATH)  df['Date'] = pd.to\_datetime(df['Date'])  df.set\_index('Date', inplace=True)  # Define target variable and features  X = df[['Open', 'High', 'Low', 'Close', 'Volume']]  y = df['Adj Close']  # Split the data  """Remove random state param when happy with RMSE score"""  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Scale the data  scaler = MinMaxScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # Create a dictionary to store the results of each model  results = {}  # # Linear Regression model  # lr\_model = LinearRegression()  # lr\_model.fit(X\_train\_scaled, y\_train)  # lr\_predictions = lr\_model.predict(X\_test\_scaled)  # results['Linear Regression'] = {  # 'R-squared': r2\_score(y\_test, lr\_predictions),  # 'RMSE': np.sqrt(mean\_squared\_error(y\_test, lr\_predictions)),  # 'MSE': mean\_squared\_error(y\_test, lr\_predictions),  # 'MAE': mean\_absolute\_error(y\_test, lr\_predictions)  # }  # Make predictions and evaluate with the RMSE.  model = sm.OLS(y\_train, X\_train\_scaled).fit()  # OLS\_model = model  predictions = model.predict(X\_test\_scaled)  # plot\_loss\_and\_metrics("MinMaxScaled OLS Model (['age', 'annual Salary', 'net worth'])",y\_test, predictions)  results['Model 1 OLS'] = {  'R-squared': r2\_score(y\_test, predictions),  'RMSE': np.sqrt(mean\_squared\_error(y\_test, predictions)),  'MSE': mean\_squared\_error(y\_test, predictions),  'MAE': mean\_absolute\_error(y\_test, predictions)  }  print(model.summary())  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))  # Neural Network model  nn\_model = Sequential()  nn\_model.add(Dense(10, activation='relu', input\_dim=5))  nn\_model.add(Dense(10, activation='relu'))  nn\_model.add(Dense(10, activation='relu'))  nn\_model.add(Dense(1, activation='linear'))  nn\_model.compile(optimizer='adam', loss='mean\_squared\_error')  history = nn\_model.fit(X\_train\_scaled, y\_train, batch\_size=16, epochs=50, validation\_split=0.2)  nn\_predictions = nn\_model.predict(X\_test\_scaled)  results['Neural Network'] = {  'R-squared': r2\_score(y\_test, nn\_predictions),  'RMSE': np.sqrt(mean\_squared\_error(y\_test, nn\_predictions)),  'MSE': mean\_squared\_error(y\_test, nn\_predictions),  'MAE': mean\_absolute\_error(y\_test, nn\_predictions)  }  # Print the results of each model  for model\_name, metrics in results.items():  print(model\_name)  print(metrics)  print("\n")  #  # """  # Generate a summary and plot variable graphs  # """  # from pandas\_profiling import ProfileReport  #  # # Generate the profiling report  # profile = ProfileReport(df, title="Tesla Dataset Profiling Report", explorative=True)  #  # # Save the report as an HTML file  # profile.to\_file("output.html")  #  # # Plot histogram of 'Adj Close'  # plt.figure(figsize=(10, 5))  # sns.histplot(df['Adj Close'], bins=50, kde=True)  # plt.title('Histogram of Adj Close')  # plt.xlabel('Adj Close')  # plt.ylabel('Frequency')  # plt.show()  #  # # Plot box plot of 'Adj Close'  # plt.figure(figsize=(10, 5))  # sns.boxplot(x=df['Adj Close'])  # plt.title('Box Plot of Adj Close')  # plt.xlabel('Adj Close')  # plt.show()  #  # # Plot trends for Open, High, Low, Close, and Adj Close  # plt.figure(figsize=(15, 8))  # plt.plot(df.index, df['Open'], label='Open')  # plt.plot(df.index, df['High'], label='High')  # plt.plot(df.index, df['Low'], label='Low')  # plt.plot(df.index, df['Close'], label='Close')  # plt.plot(df.index, df['Adj Close'], label='Adj Close')  #  # # Customize plot  # plt.title('Trends in Opening, Closing, High, Low, and Adjusted Closing Prices')  # plt.xlabel('Date')  # plt.ylabel('Price')  # plt.legend()  # plt.grid()  # plt.show()  #  # # Plot histogram of 'Volume'  # plt.figure(figsize=(10, 5))  # sns.histplot(df['Volume'], bins=50, kde=True)  # plt.title('Histogram of Trading Volume')  # plt.xlabel('Volume')  # plt.ylabel('Frequency')  # plt.show()  #  # # Plot box plot of 'Volume'  # plt.figure(figsize=(10, 5))  # sns.boxplot(x=df['Volume'])  # plt.title('Box Plot of Trading Volume')  # plt.xlabel('Volume')  # plt.show()  #  # # Create a scatter plot of trading volume vs adjusted closing price  # plt.figure(figsize=(10, 5))  # sns.scatterplot(x=df['Volume'], y=df['Adj Close'])  # plt.title('Scatter Plot: Trading Volume vs Adjusted Closing Price')  # plt.xlabel('Trading Volume')  # plt.ylabel('Adjusted Closing Price')  # plt.show()  """  Create NN Model  """  from keras.optimizers import Adam, RMSprop  def create\_nn\_model(optimizer='rmsprop', neurons=30, lr=0.1, activation='relu', initializer='he\_normal'):  model = Sequential()  model.add(Dense(neurons, activation=activation, kernel\_initializer=initializer, input\_dim=5))  model.add(Dense(neurons, activation=activation, kernel\_initializer=initializer))  model.add(Dense(1, activation='linear'))  if optimizer == 'adam':  opt = Adam(learning\_rate=lr)  elif optimizer == 'rmsprop':  opt = RMSprop(learning\_rate=lr)  model.compile(optimizer=opt, loss='mean\_squared\_error')  return model  # Define the grid search parameters  param\_grid = {  'optimizer': ['adam', 'rmsprop'],  'neurons': [10, 20, 30],  'lr': [0.001, 0.01, 0.1],  'activation': ['relu', 'tanh'],  'initializer': ['he\_normal', 'he\_uniform']  }  """  Perform Grid Search  """  # from skopt import BayesSearchCV  # from skopt.space import Real, Categorical, Integer  # from keras.optimizers import Adam  # from scikeras.wrappers import KerasRegressor  # from sklearn.model\_selection import RandomizedSearchCV  # from sklearn.model\_selection import GridSearchCV  #  #  # # Create a KerasRegressor model  # nn\_model = KerasRegressor(build\_fn=create\_nn\_model, epochs=50, batch\_size=16, verbose=0, optimizer='adam', neurons=10, lr=0.001, activation='relu', initializer='he\_normal')  #  #  # # Define the search space  # search\_space = {  # 'optimizer': Categorical(['adam', 'rmsprop']),  # 'neurons': Integer(10, 100),  # 'activation': Categorical(['relu', 'tanh']),  # 'lr': Real(1e-4, 1e-2, prior='log-uniform'),  # }  #  #  # # Perform the Grid search  # grid\_search = GridSearchCV(estimator=nn\_model, param\_grid=param\_grid, n\_jobs=-1, cv=3, scoring='neg\_mean\_squared\_error')  # grid\_search\_result = grid\_search.fit(X\_train\_scaled, y\_train)  #  #  # # Summarize the results  # print(f"Best: {grid\_search\_result.best\_score\_} using {grid\_search\_result.best\_params\_}")  # means = grid\_search\_result.cv\_results\_['mean\_test\_score']  # stds = grid\_search\_result.cv\_results\_['std\_test\_score']  # params = grid\_search\_result.cv\_results\_['params']  # for mean, stdev, param in zip(means, stds, params):  # print(f"{mean:.4f} ({stdev:.4f}) with: {param}")  """  Optimized parameters  Best: -1.6165573675727283 using  {'activation': 'relu', 'initializer': 'he\_uniform', 'lr': 0.1, 'neurons': 30, 'optimizer': 'rmsprop'}  Best: -2.35016944334606 using  {'activation': 'relu', 'initializer': 'he\_normal', 'lr': 0.01, 'neurons': 20, 'optimizer': 'adam'}  Best: -2.0810359138867462 using  {'activation': 'relu', 'initializer': 'he\_normal', 'lr': 0.1, 'neurons': 10, 'optimizer': 'rmsprop'}  Best: -1.6094852824378314 using  {'activation': 'relu', 'initializer': 'he\_uniform', 'lr': 0.1, 'neurons': 30, 'optimizer': 'rmsprop'}  """  """  Create an Optimized Neural Network Model  """  from keras.models import Sequential  from keras.layers import Dense  from keras.optimizers import Adam, RMSprop  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  import matplotlib.pyplot as plt  # Define the optimized parameters obtained from the grid search  optimizer = 'rmsprop'  neurons = 30  lr = 0.1  activation = 'relu'  initializer = 'he\_uniform'  # Create the optimized NN model  def create\_nn\_model(optimizer, neurons, lr, activation, initializer):  model = Sequential()  model.add(Dense(neurons, activation=activation, kernel\_initializer=initializer, input\_dim=5))  model.add(Dense(neurons, activation=activation, kernel\_initializer=initializer))  model.add(Dense(1, activation='linear'))  if optimizer == 'adam':  opt = Adam(learning\_rate=lr)  elif optimizer == 'rmsprop':  opt = RMSprop(learning\_rate=lr)  model.compile(optimizer=opt, loss='mean\_squared\_error')  return model  optimized\_model = create\_nn\_model(optimizer, neurons, lr, activation, initializer)  # Train the optimized model  history = optimized\_model.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_split=0.2)  # Predict on the test set  y\_pred = optimized\_model.predict(X\_test\_scaled)  # Calculate the mean squared error  mse = mean\_squared\_error(y\_test, y\_pred)  # Calculate the mean absolute error  mae = mean\_absolute\_error(y\_test, y\_pred)  # Calculate the R-squared score  r2score = r2\_score(y\_test, y\_pred)  # Calculate the root mean squared error  rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)  # Print the metrics  print(f"R-squared: {r2score:.4f}")  print(f"Mean Squared Error: {mse:.4f}")  print(f"Root Mean Squared Error: {rmse:.4f}")  print(f"Mean Absolute Error: {mae:.4f}")  # Plot the model loss during training  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('Model Loss')  plt.ylabel('Loss')  plt.xlabel('Epoch')  plt.legend(['train', 'validation'], loc='upper right')  plt.show() |
| --- |