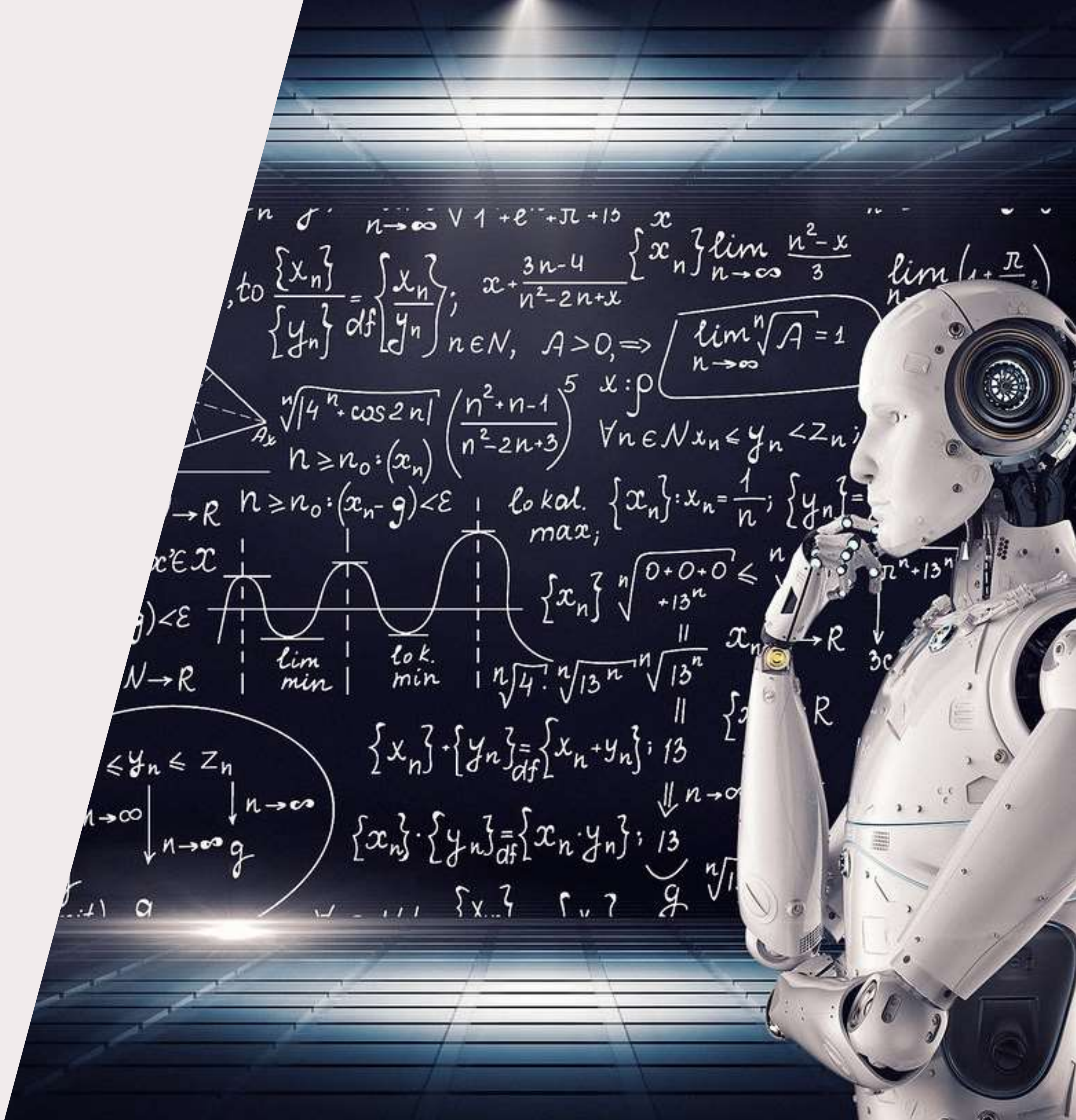
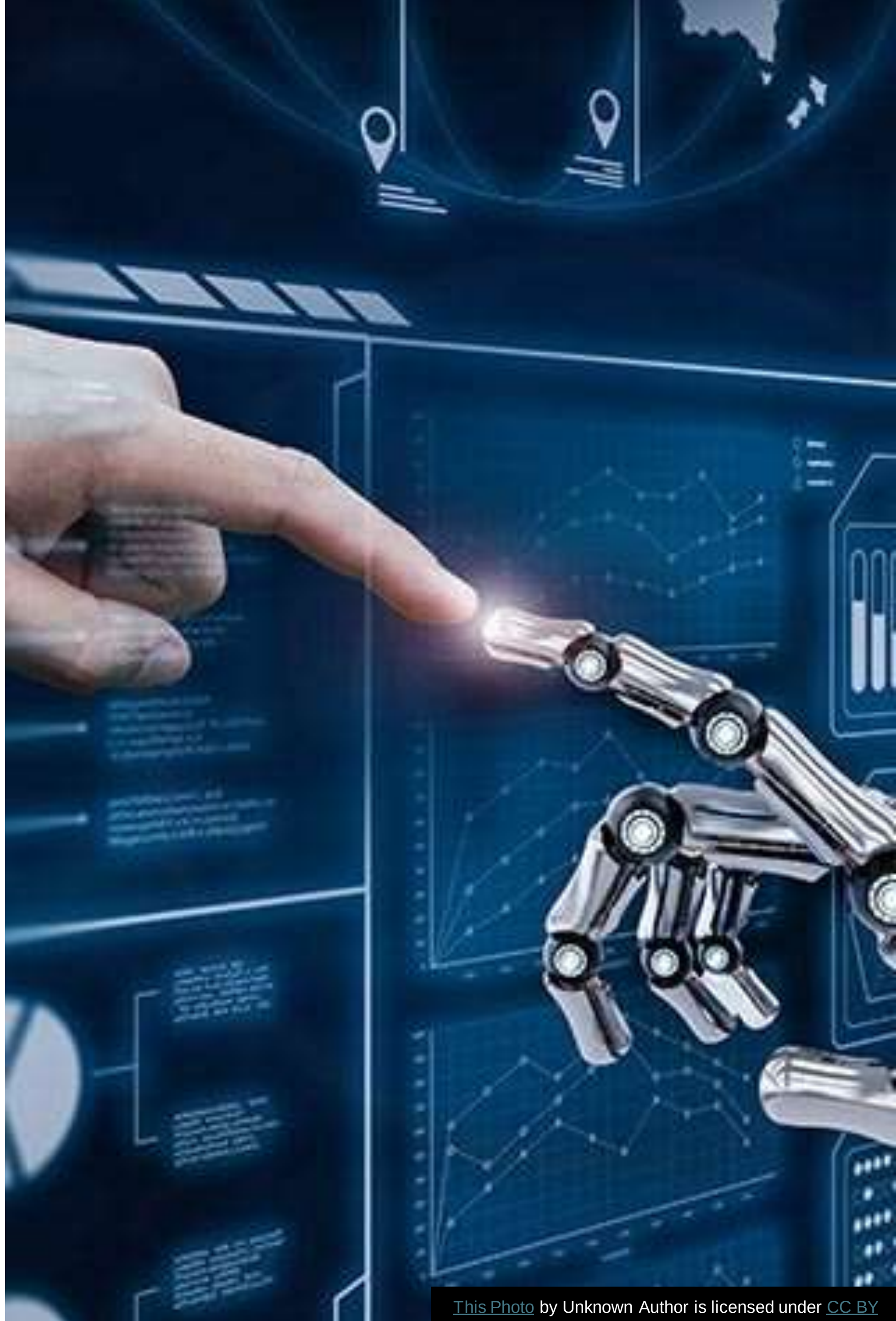
DEEP LEARNING PROJECT

Contributors: Avraham Rahimov, Elon Ezra



AGENDA



INTRODUCTION

This project explores the interesting field of digital money, with a special emphasis on Bitcoin, which is the first and most famous digital currency. We will be working with a dataset that contains historical data on Bitcoin prices since 2014. Our goal is to build a model that can accurately predict the "Open_Close_Gap", which is the difference between the opening and closing prices of Bitcoin on a given day.

We will be comparing the performance of three different models: a dummy model, an Artificial Neural Network (ANN), and a Linear Regression model. The dummy model serves as a baseline, providing a point of reference for the performance of the other two models. The ANN and Linear Regression models represent two different approaches to predictive modeling, with the ANN leveraging the power of deep learning.

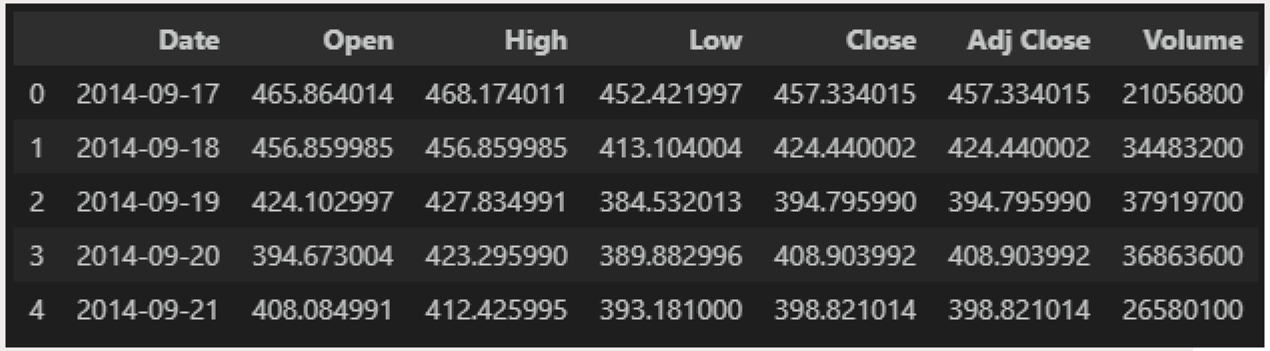
We will be using TensorFlow, a powerful library for numerical computation, to build and train our models. We will also be using other data science libraries like pandas for data manipulation and matplotlib for data visualization.

Throughout this project, we will follow a systematic approach to machine learning, starting with data preprocessing, followed by model training, and finally, model evaluation. We will also be using techniques like cross-validation and data normalization to ensure that our models are robust and reliable.

By the end of this project, we will have a clear understanding of how to build, train, and evaluate deep learning models using TensorFlow, and how these models compare to more traditional approaches like Linear Regression.

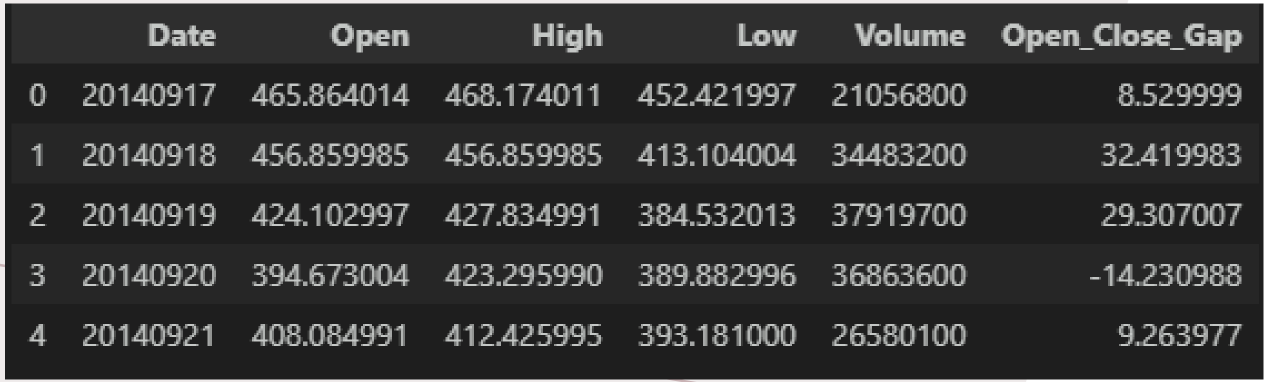
DATA LOADING AND PREPROCESSING

This is our data in the beginning, including 7 columns and 2713 lines.



	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100

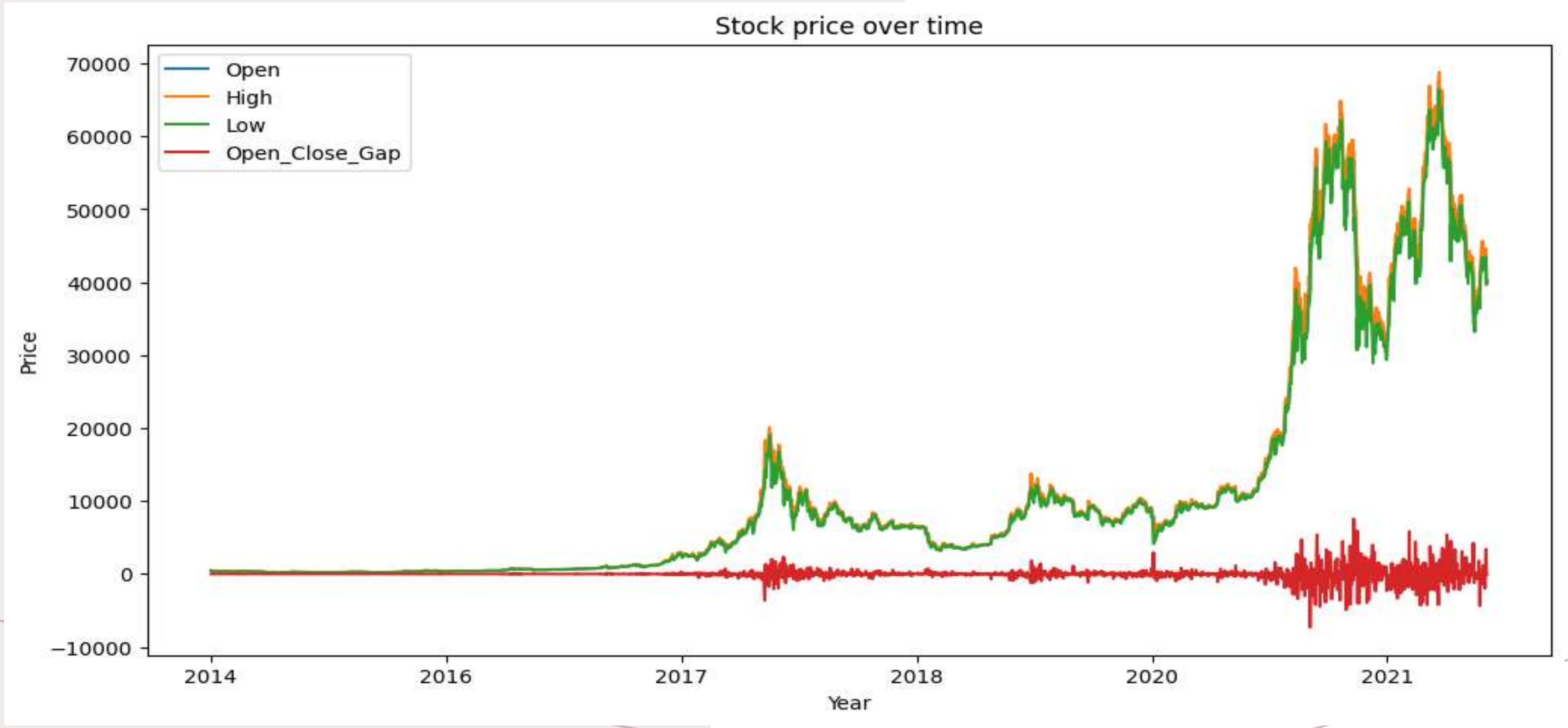
This is our data after preprocessing which includes dropping unnecessary columns, add a new column(label) and make the date as int.
We also normalize the data, but it’s done later.



	Date	Open	High	Low	Volume	Open_Close_Gap
0	20140917	465.864014	468.174011	452.421997	21056800	8.529999
1	20140918	456.859985	456.859985	413.104004	34483200	32.419983
2	20140919	424.102997	427.834991	384.532013	37919700	29.307007
3	20140920	394.673004	423.295990	389.882996	36863600	-14.230988
4	20140921	408.084991	412.425995	393.181000	26580100	9.263977

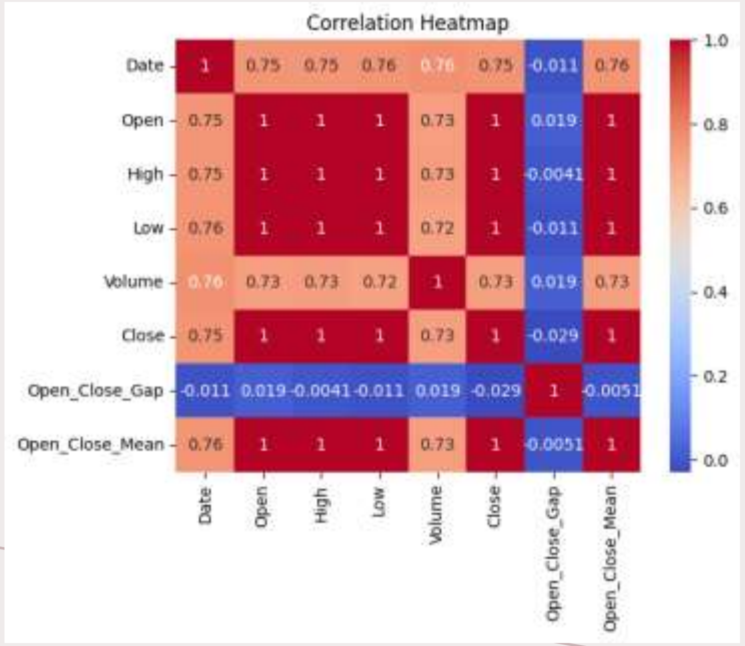
EXPLORATORY DATA ANALYSIS (EDA)

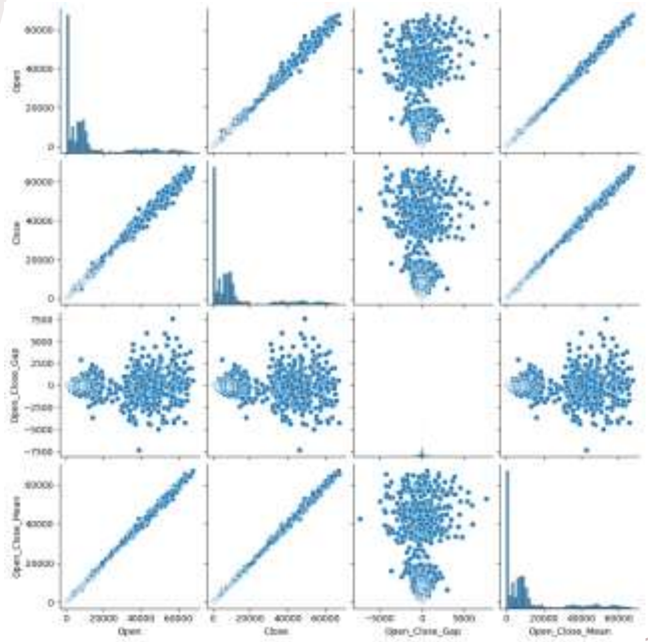
Plot the data to see the stock price over time(by years only)
We can see the list in the top left corner which indicates the column for each color.
We also can see that the label(Open_Close_Gap) is around the zero line which we can infer that the open and close values are the same with a little difference.



CONTINUE EDA

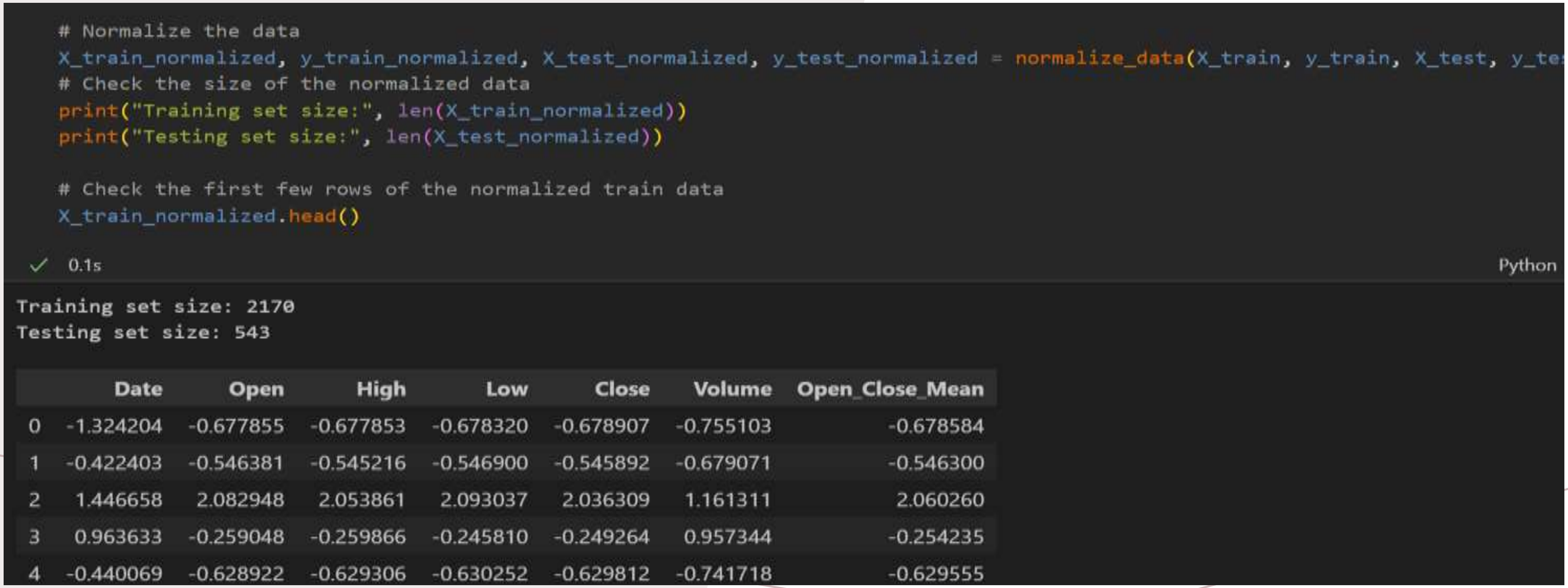
The heatmap reveals strong positive correlations among the “Open”, “Low”, “High”, “Close”, and “Open_Close_Mean” variables. Despite this, negative correlations are observed with other factors like volume traded. The pairplot further visualizes these relationships with scatter plots and histograms for individual variables. It uncovers linear relationships between opening and closing prices while showing varied distributions for other combinations of data. The Open_Close_Gap, which shows less correlation with other variables in the heatmap, appears to have a more complex, non-linear relationship with other variables in the pairplot. This supports the decision to predict the Open_Close_Gap as it seems to contain unique information not captured by the other variables.





DATA NORMALIZATION

A normalization function is created to standardize inputs by subtracting the mean and dividing by the variance’s square root. In TensorFlow, this function calculates the training data’s mean and variance for normalization. Crucially, the test data is also normalized using the training data’s statistics, not its own, to prevent test set influence during training. The training and test targets are similarly normalized using the training targets’ statistics.
Here is an example of the X_train after the normalization(we ensured that the sizes of the data is the same as before.):



```
# Normalize the data
X_train_normalized, y_train_normalized, X_test_normalized, y_test_normalized = normalize_data(X_train, y_train, X_test, y_te
# Check the size of the normalized data
print("Training set size:", len(X_train_normalized))
print("Testing set size:", len(X_test_normalized))

# Check the first few rows of the normalized train data
X_train_normalized.head()
```

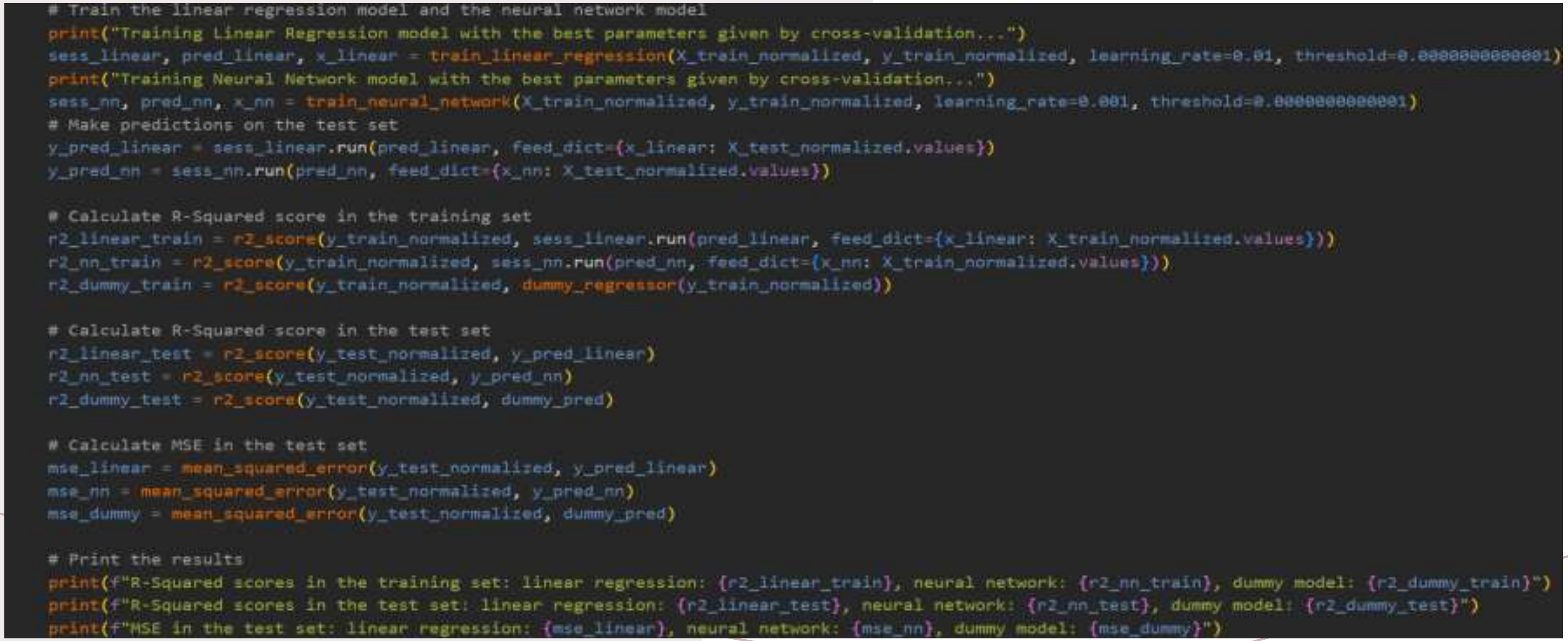
✓ 0.1s Python

```
Training set size: 2170
Testing set size: 543
```

	Date	Open	High	Low	Close	Volume	Open_Close_Mean
0	-1.324204	-0.677855	-0.677853	-0.678320	-0.678907	-0.755103	-0.678584
1	-0.422403	-0.546381	-0.545216	-0.546900	-0.545892	-0.679071	-0.546300
2	1.446658	2.082948	2.053861	2.093037	2.036309	1.161311	2.060260
3	0.963633	-0.259048	-0.259866	-0.245810	-0.249264	0.957344	-0.254235
4	-0.440069	-0.628922	-0.629306	-0.630252	-0.629812	-0.741718	-0.629555

MODEL TRAINING

We trained a linear regression model and a neural network model using normalized training data. After training, we used these models to make predictions on the normalized test data. We calculated the R-Squared scores for both the training and test sets for each model, including a dummy model. We also computed the Mean Squared Error (MSE) for the test set for each model. Finally, we printed the R-Squared scores and MSE for each model. The learning rates and thresholds for the models were specified in the training functions. The dummy model was used as a baseline for comparison.



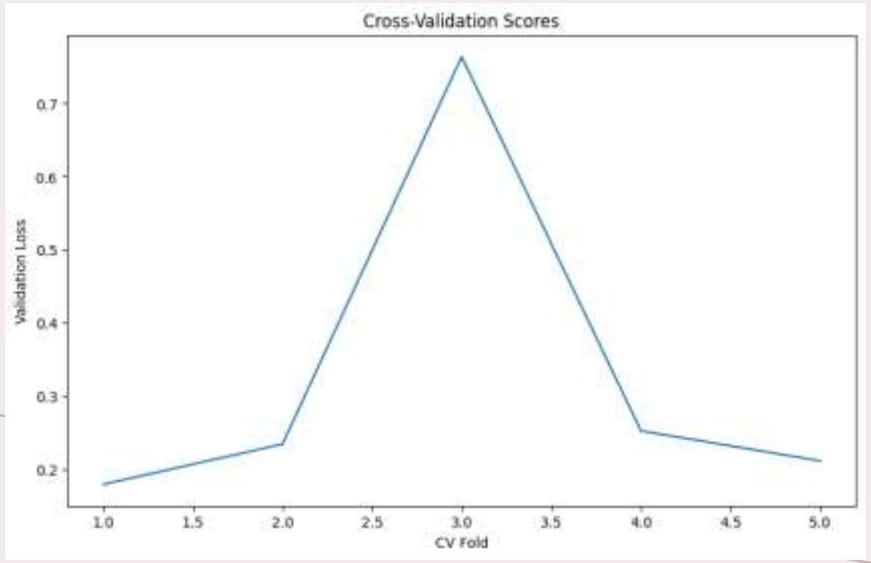
```
# Train the linear regression model and the neural network model
print("Training Linear Regression model with the best parameters given by cross-validation...")
sess_linear, pred_linear, x_linear = train_linear_regression(X_train_normalized, y_train_normalized, learning_rate=0.01, threshold=0.0000000000001)
print("Training Neural Network model with the best parameters given by cross-validation...")
sess_nn, pred_nn, x_nn = train_neural_network(X_train_normalized, y_train_normalized, learning_rate=0.001, threshold=0.0000000000001)
# Make predictions on the test set
y_pred_linear = sess_linear.run(pred_linear, feed_dict={x_linear: X_test_normalized.values})
y_pred_nn = sess_nn.run(pred_nn, feed_dict={x_nn: X_test_normalized.values})

# Calculate R-Squared score in the training set
r2_linear_train = r2_score(y_train_normalized, sess_linear.run(pred_linear, feed_dict={x_linear: X_train_normalized.values}))
r2_nn_train = r2_score(y_train_normalized, sess_nn.run(pred_nn, feed_dict={x_nn: X_train_normalized.values}))
r2_dummy_train = r2_score(y_train_normalized, dummy_regressor(y_train_normalized))

# Calculate R-Squared score in the test set
r2_linear_test = r2_score(y_test_normalized, y_pred_linear)
r2_nn_test = r2_score(y_test_normalized, y_pred_nn)
r2_dummy_test = r2_score(y_test_normalized, dummy_pred)

# Calculate MSE in the test set
mse_linear = mean_squared_error(y_test_normalized, y_pred_linear)
mse_nn = mean_squared_error(y_test_normalized, y_pred_nn)
mse_dummy = mean_squared_error(y_test_normalized, dummy_pred)

# Print the results
print(f"R-Squared scores in the training set: linear regression: {r2_linear_train}, neural network: {r2_nn_train}, dummy model: {r2_dummy_train}")
print(f"R-Squared scores in the test set: linear regression: {r2_linear_test}, neural network: {r2_nn_test}, dummy model: {r2_dummy_test}")
print(f"MSE in the test set: linear regression: {mse_linear}, neural network: {mse_nn}, dummy model: {mse_dummy}")
```

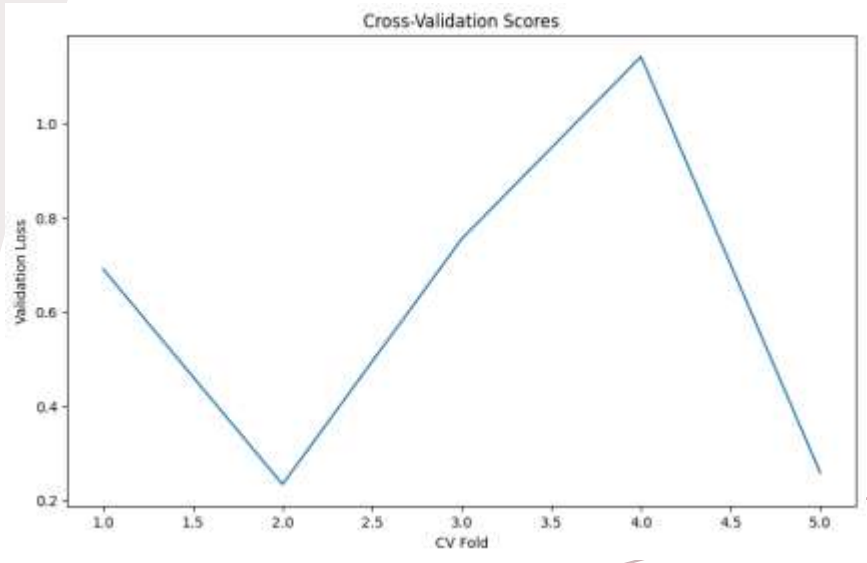
CROSS-VALIDATION

The cross-validation results show how well the Linear Regression and Artificial Neural Network (ANN) models performed across different subsets of the training data.

For the Linear Regression model, the cross-validation scores(on left) ranged from 0.1796 to 0.7636, with an average score of 0.3284. The variation in scores suggests that the model's performance was somewhat inconsistent across different subsets of the data. The early stopping epochs also varied widely, indicating that the model's convergence speed was not consistent across different subsets.

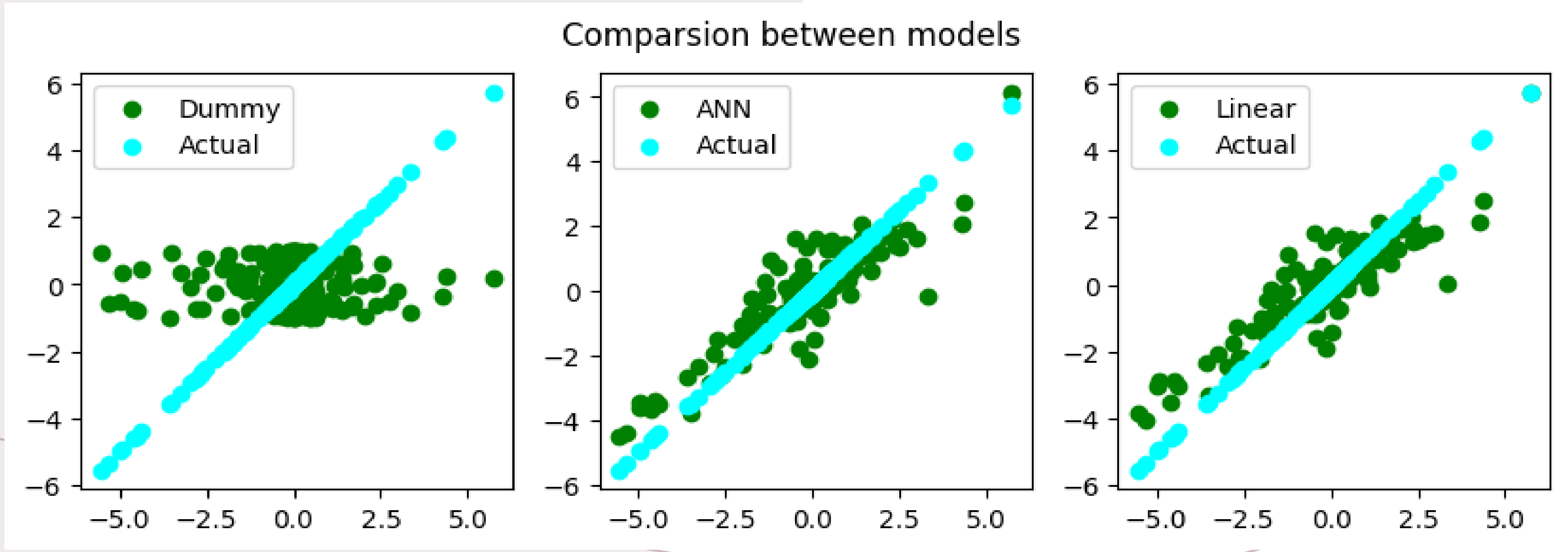
For the ANN model, the cross-validation scores(on right) ranged from 0.2334 to 1.1413, with an average score of 0.6155. This suggests that the ANN model's performance was also inconsistent across different subsets of the data. However, the average score was higher than that of the Linear Regression model, suggesting that the ANN model may have performed better overall. The early stopping epochs for the ANN model also varied, indicating that the model's convergence speed was not consistent across different subsets.





MODEL EVALUATION

We created a figure with three subplots to compare the predictions of our models. The figure displays scatter plots of the normalized test targets against the predictions of the dummy model, the artificial neural network (ANN), and the linear regression model. In each subplot, we also plotted the actual normalized test targets against themselves as a reference. This allows us to visually assess the performance of each model by comparing the scatter plots to the reference line. The closer the predictions are to the reference line, the better the model's performance. The labels 'Dummy', 'ANN', and 'Linear' represent the dummy model, the artificial neural network, and the linear regression model respectively. The 'Actual' label represents the actual normalized test targets.

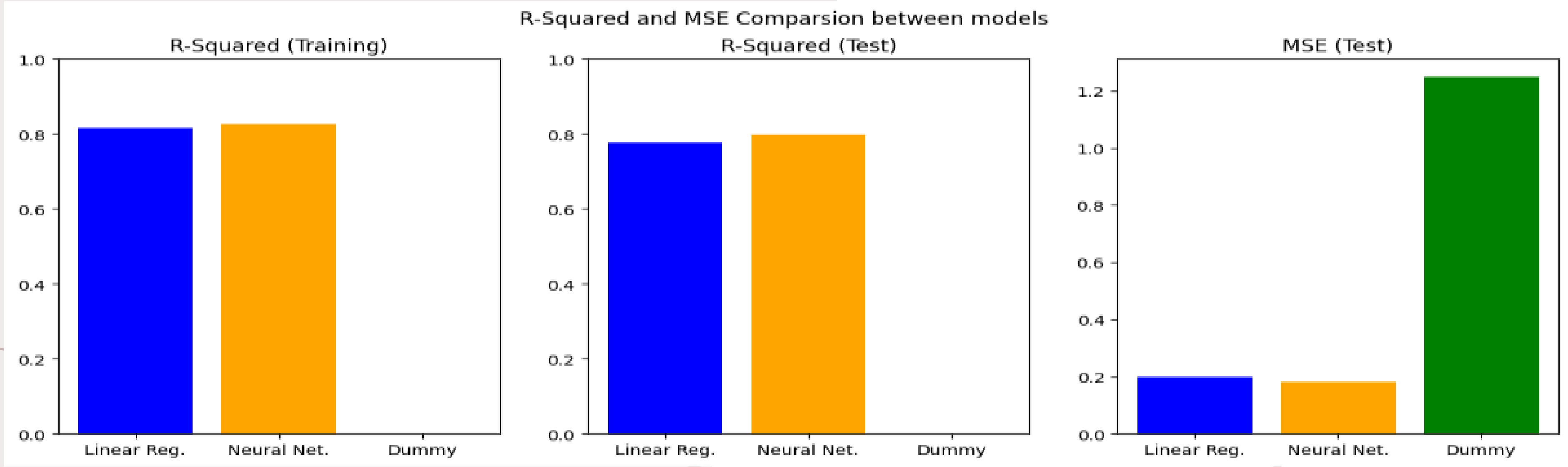


RESULTS VISUALIZATION

The R-Squared scores on the training set were 0.815 for the Linear Regression model and 0.828 for the Neural Network model, indicating that both models were able to explain a significant portion of the variance in the training data. The Dummy model, as expected, performed poorly with an R-Squared score of -0.303.

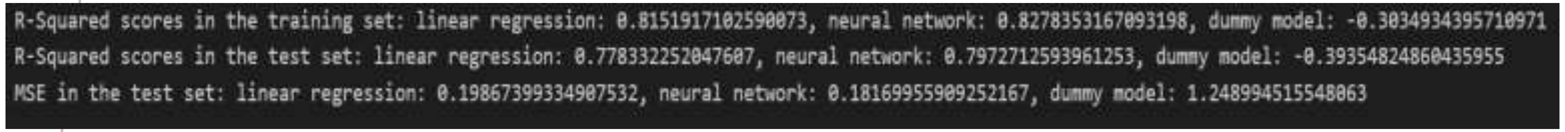
On the test set, the Linear Regression model achieved an R-Squared score of 0.778 and the Neural Network model achieved a score of 0.797, suggesting that both models generalized well to unseen data. The Dummy model had a negative R-Squared score of -0.394, indicating that it performed worse than a model that always predicts the mean of the target values.

In terms of Mean Squared Error (MSE) on the test set, the Neural Network model outperformed the Linear Regression model with an MSE of 0.182 compared to 0.199. The Dummy model had a much higher MSE of 1.249.



CONCLUSION

In conclusion, both the Linear Regression and Neural Network models showed promising results in predicting the 'Open_Close_Gap' in Bitcoin prices. The Neural Network model slightly outperformed the Linear Regression model, demonstrating the potential of deep learning for this task. However, further tuning of the model parameters and exploration of different model architectures could potentially improve these results. Despite the inherent volatility and unpredictability of Bitcoin prices, this project shows that machine learning can provide valuable insights and predictions in the world of cryptocurrency.



```
R-Squared scores in the training set: linear regression: 0.8151917102590073, neural network: 0.8278353167093198, dummy model: -0.3034934395710971
R-Squared scores in the test set: linear regression: 0.778332252047607, neural network: 0.7972712593961253, dummy model: -0.39354824860435955
MSE in the test set: linear regression: 0.19867399334907532, neural network: 0.18169955909252167, dummy model: 1.248994515548063
```