Configuration Manual for Enhancing Cryptocurrency Price Prediction Using Transformer-Based Models for Effective Time-Series Analysis

MSc Research Project
Data Analytics

Rambabu Karicheti
Student ID: 22244239

School of Computing
National College of Ireland

Supervisor: Sallar Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……………RAMBABU KARICHETI……………

Student ID: ……………………X22244239……………………..……

Programme: ……………DATA ANALYTICS…… **Year:** ……2025..

Module: ……………MSC RESEARCH PROJECT……………………….………

Lecturer: ……………………SALLAR KHAN……………….………

Submission Due Date: …………………29 JANUARY 2025……………….………

Project Title: … Enhancing Cryptocurrency Price Prediction Using Transformer Based Models for effective Time-Series Analysis ……

Word Count: ……………………………………… **Page Count:** ……………………………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………………………RAMBABU KARICHETI………

Date: ……………………29 JANUARY 2025…………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Enhancing Cryptocurrency Price Prediction Using Transformer-Based Models for Effective Time-Series Analysis

RAMBABU KARICHETI
22244239
DATA ANALYTICS
NATIONAL COLLEGE OF IRELAND

1. Introduction

This research focuses on enhancing the prediction accuracy of cryptocurrency prices using Transformer-based models, which are well-suited for time-series analysis. The goal is to apply advanced machine learning techniques to predict future cryptocurrency prices by analyzing historical data, integrating Transformer models with traditional algorithms for baseline comparisons. This manual outlines the steps and configuration needed for the study, from data acquisition to model evaluation.

2. Prerequisites

Software Requirements

- **Python (>= 3.6):** Python is the primary programming language used for this study.
- **TensorFlow:** For building and training the Transformer model and other deep learning algorithms.
- **Keras:** A high-level neural network library for easy model building.
- scikit-learn: For traditional machine learning models (e.g., SVM, Random Forest, Gradient Boosting).
- **pandas:** For data handling and preprocessing.
- **NumPy:** For numerical computations.
- **matplotlib & seaborn:** For data visualization.
- **plotly:** For interactive visualizations.
- **Jupyter Notebook/IDE:** Recommended for running and managing the code.

To install the required libraries, use the following command:

```
pip install tensorflow pandas scikit-learn matplotlib seaborn plotly
```

3. Data Preparation and Preprocessing

3.1 Data Acquisition

The cryptocurrency data must be loaded from a CSV file containing historical price data which is sourced from the kaggle. Each row should represent a specific timestamp, including columns such as the Timestamp, Open, High, Low, Close, Volume BTC, and Volume USD.

Table 1: Description of Dataset

Column	Description	Data Type
Timestamp	A numerical representation of the date and time when the data point was recorded (Unix format).	int64
Date	The date of the data entry in a human-readable format (likely "YYYY-MM-DD").	object
Symbol	The ticker symbol of the asset being traded (e.g., cryptocurrency or financial asset).	object
Open	The opening price of the asset at the beginning of the given time period.	float64
High	The highest price reached by the asset during the given time period.	float64
Low	The lowest price reached by the asset during the given time period.	float64
Close	The closing price of the asset at the end of the given time period.	float64
Volume BTC	The total amount of the asset traded, measured in Bitcoin (BTC) during the given time period.	float64
Volume USD	The total value of the asset traded, measured in US Dollars (USD) during the given time period.	float64

3.2 Data Exploration

Use Pandas to explore the dataset. This includes viewing the first and last rows, checking for basic information, and reviewing basic statistics about the data.

```python
#view the first 10 values of the attributes
crypto_data.head(10)
```

Python

	Timestamp	Date	Symbol	Open	High	Low	Close	Volume BTC	Volume USD
0	1676939580000	2023-02-21 00:33:00	BTC/USD	24859.34	24859.34	24859.34	24859.34	0.000000	0.000000
1	1676939520000	2023-02-21 00:32:00	BTC/USD	24821.96	24859.34	24821.96	24859.34	0.103099	2562.977818
2	1676939460000	2023-02-21 00:31:00	BTC/USD	24818.09	24821.96	24815.47	24821.96	0.090640	2249.866178
3	1676939400000	2023-02-21 00:30:00	BTC/USD	24812.25	24818.09	24812.25	24818.09	0.002203	54.681450
4	1676939340000	2023-02-21 00:29:00	BTC/USD	24809.27	24812.25	24809.27	24812.25	0.090675	2249.862431
5	1676939280000	2023-02-21 00:28:00	BTC/USD	24809.28	24809.28	24809.27	24809.27	0.003961	98.279938
6	1676939220000	2023-02-21 00:27:00	BTC/USD	24809.28	24809.28	24809.28	24809.28	0.000000	0.000000
7	1676939160000	2023-02-21 00:26:00	BTC/USD	24809.28	24809.28	24809.28	24809.28	0.000000	0.000000
8	1676939100000	2023-02-21 00:25:00	BTC/USD	24821.31	24821.31	24809.28	24809.28	0.001361	33.758732
9	1676939040000	2023-02-21 00:24:00	BTC/USD	24817.20	24821.31	24811.49	24821.31	0.212014	5262.474899

```python
#view the last 10 values of the attributes
crypto_data.tail(10)
```

Python

	Timestamp	Date	Symbol	Open	High	Low	Close	Volume BTC	Volume USD
3766753	1444312140	2015-10-08 13:49:00	BTC/USD	242.96	243.00	242.96	243.00	0.100000	24.300000
3766754	1444312080	2015-10-08 13:48:00	BTC/USD	242.96	242.96	242.96	242.96	0.000000	0.000000
3766755	1444312020	2015-10-08 13:47:00	BTC/USD	242.96	242.96	242.96	242.96	0.000000	0.000000
3766756	1444311960	2015-10-08 13:46:00	BTC/USD	242.96	242.96	242.96	242.96	0.004000	0.971840
3766757	1444311900	2015-10-08 13:45:00	BTC/USD	242.96	242.96	242.96	242.96	0.000000	0.000000
3766758	1444311840	2015-10-08 13:44:00	BTC/USD	242.96	242.96	242.96	242.96	0.033491	8.137003
3766759	1444311780	2015-10-08 13:43:00	BTC/USD	242.95	242.96	242.95	242.96	0.010000	2.429600
3766760	1444311720	2015-10-08 13:42:00	BTC/USD	242.95	242.95	242.95	242.95	0.000000	0.000000
3766761	1444311660	2015-10-08 13:41:00	BTC/USD	242.50	242.95	242.50	242.95	0.001000	0.242950
3766762	1444311600	2015-10-08 13:40:00	BTC/USD	0.00	242.50	0.00	242.50	0.050000	12.125000

```python
#basic information about the dataset
crypto_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3766763 entries, 0 to 3766762
Data columns (total 9 columns):
 #   Column      Dtype
---  ------      -----
 0   Timestamp   int64
 1   Date        object
 2   Symbol      object
 3   Open        float64
 4   High        float64
 5   Low         float64
 6   Close       float64
 7   Volume BTC  float64
 8   Volume USD  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 258.6+ MB
```

```python
#description about the dataset attributes
crypto_data.describe()
```

Python

	Timestamp	Open	High	Low	Close	Volume BTC	Volume USD
count	3.766763e+06	3.766763e+06	3.766763e+06	3.766763e+06	3.766763e+06	3.766763e+06	3.766763e+06
mean	9.633091e+11	1.497170e+04	1.497938e+04	1.496394e+04	1.497171e+04	2.082630e+00	2.390526e+04
std	7.879469e+11	1.658607e+04	1.659550e+04	1.657636e+04	1.658607e+04	1.681797e+01	1.135481e+05
min	1.444312e+09	0.000000e+00	2.425000e+02	0.000000e+00	2.360000e+02	0.000000e+00	0.000000e+00
25%	1.500813e+09	2.750000e+03	2.750650e+03	2.749840e+03	2.750000e+03	0.000000e+00	0.000000e+00
50%	1.560047e+12	8.493230e+03	8.497840e+03	8.489520e+03	8.493240e+03	7.482203e-02	8.237107e+02
75%	1.620006e+12	2.038378e+04	2.039286e+04	2.037482e+04	2.038379e+04	9.309989e-01	1.142039e+04
max	1.676940e+12	6.899887e+04	6.900000e+04	6.878700e+04	6.899887e+04	8.264463e+03	3.749580e+07

3.3 Data Processing

- Convert the Timestmp column to a datetime format:

```python
# Convert timestamp to datetime format
crypto_data['Timestamp'] = pd.to_datetime(crypto_data['Timestamp'], unit='ms')
```

- Sort data by Timestamp and set as index:

```python
# Reset the index of the DataFrame
crypto_data = crypto_data.reset_index(drop=True)
```

```python
# Set 'Timestamp' column as index
crypto_data.set_index('Timestamp', inplace=True)
```

- Drop the irrelevant columns:

```python
# drop the irrevelant column Date
crypto_data = crypto_data.drop(columns=["Date","Symbol"])
crypto_data = crypto_data[2200000:]
```

- Handle missing values by forward filling:

```python
# Handle missing values by forward filling
crypto_data.fillna(method='ffill', inplace=True)
```

- Calculate moving averages and relative strength index (RSI):

```python
# Calculate RSI (Relative Strength Index)
delta = crypto_data['Close'].diff()
gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)

avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean()

rs = avg_gain / avg_loss
crypto_data['RSI'] = 100 - (100 / (1 + rs))
```

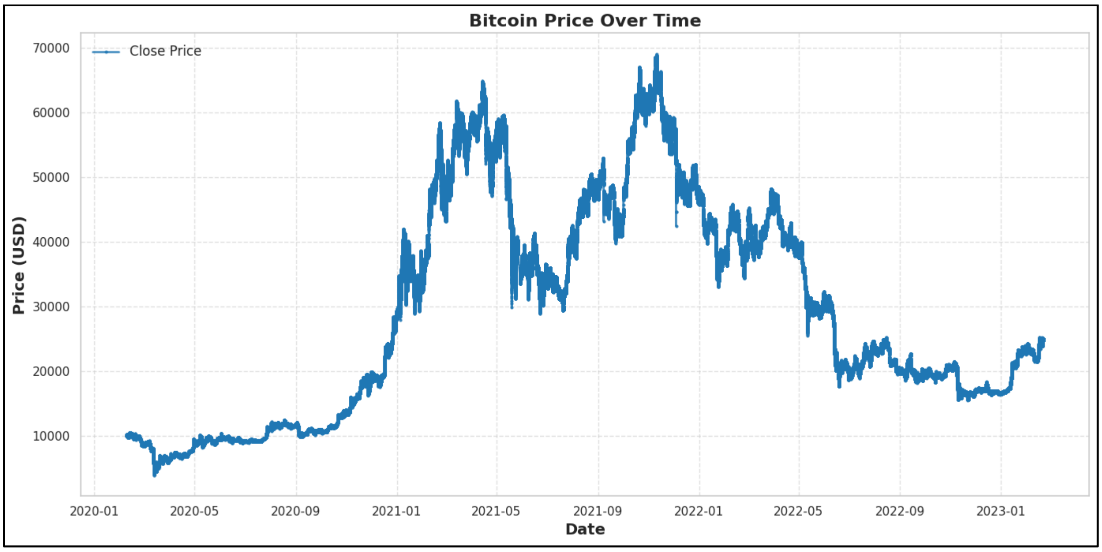
- Preprocessed Dataset:

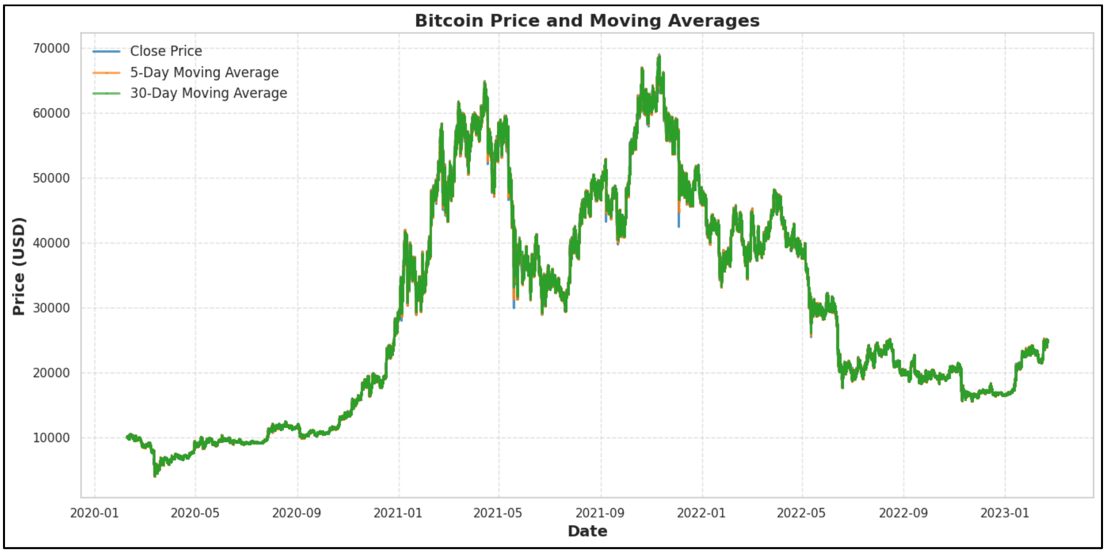
```python
# have a look to the processed dataframe attributes
crypto_data
```

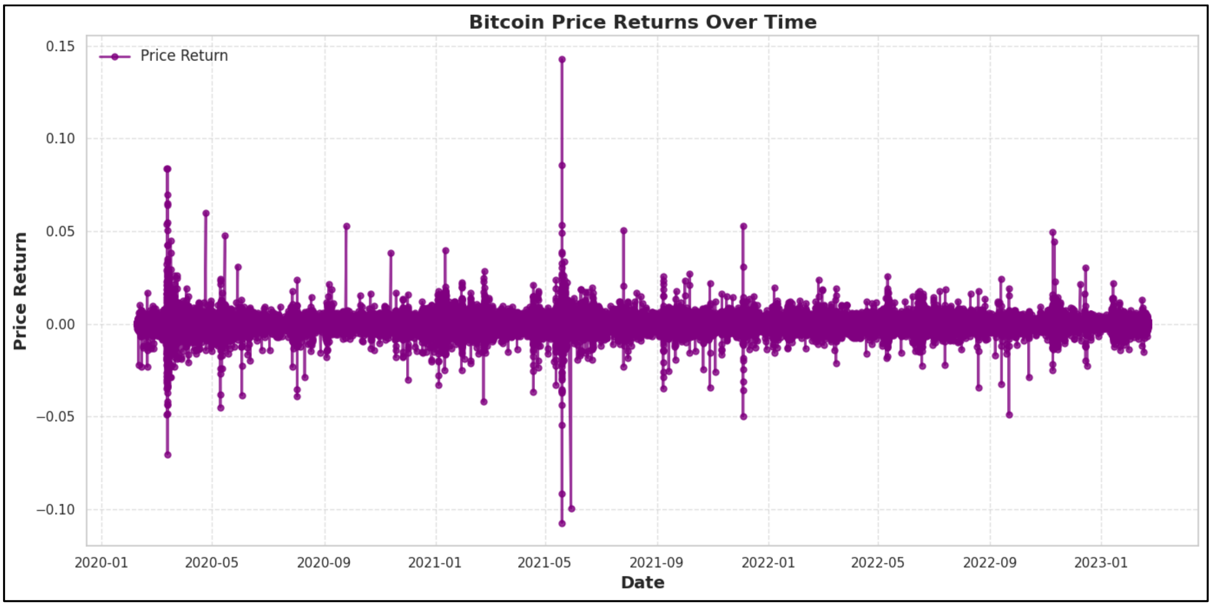
Timestamp	Open	High	Low	Close	Volume BTC	Volume USD	MA_5	MA_30	Price_return	RSI
2021-11-27 20:02:00	54984.80	55039.19	54984.80	55039.19	0.036675	2018.571650	54991.284	54980.239333	0.000989	65.096672
2021-11-27 20:03:00	55039.19	55100.00	55038.98	55100.00	1.791091	98689.138895	55016.408	54984.605333	0.001105	70.098338
2021-11-27 20:04:00	55100.00	55104.01	55071.30	55104.00	7.893659	434972.180577	55039.176	54989.104667	0.000073	69.575868
2021-11-27 20:05:00	55104.00	55104.00	55046.29	55092.18	1.231642	67853.822926	55064.034	54994.072000	-0.000215	64.683101
2021-11-27 20:06:00	55092.18	55100.47	55040.86	55040.86	0.470982	25923.262581	55075.246	54997.943333	-0.000932	54.657792
...	...	...	...	...	...	...	...	...	...	...
2023-02-21 00:29:00	24809.27	24812.25	24809.27	24812.25	0.090675	2249.862431	24809.872	24816.564667	0.000120	44.770528
2023-02-21 00:30:00	24812.25	24818.09	24812.25	24818.09	0.002203	54.681450	24811.634	24815.658333	0.000235	68.127097
2023-02-21 00:31:00	24818.09	24821.96	24815.47	24821.96	0.090640	2249.866178	24814.170	24814.881000	0.000156	73.954922
2023-02-21 00:32:00	24821.96	24859.34	24821.96	24859.34	0.103099	2562.977818	24824.182	24815.692000	0.001506	85.644851
2023-02-21 00:33:00	24859.34	24859.34	24859.34	24859.34	0.000000	0.000000	24834.196	24816.600333	0.000000	86.288578

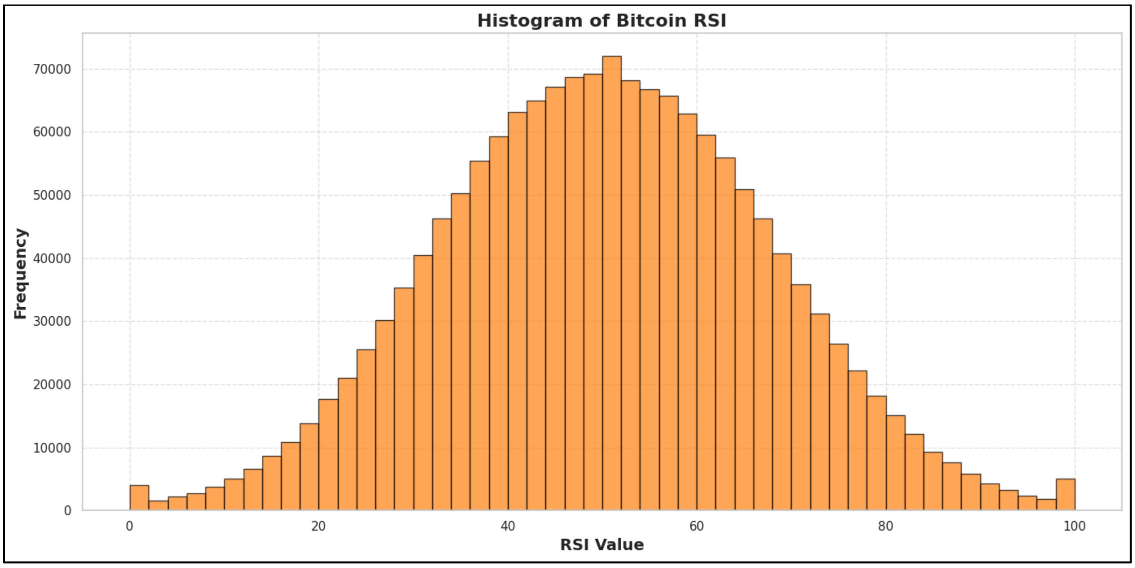
4. Exploratory Data Analysis

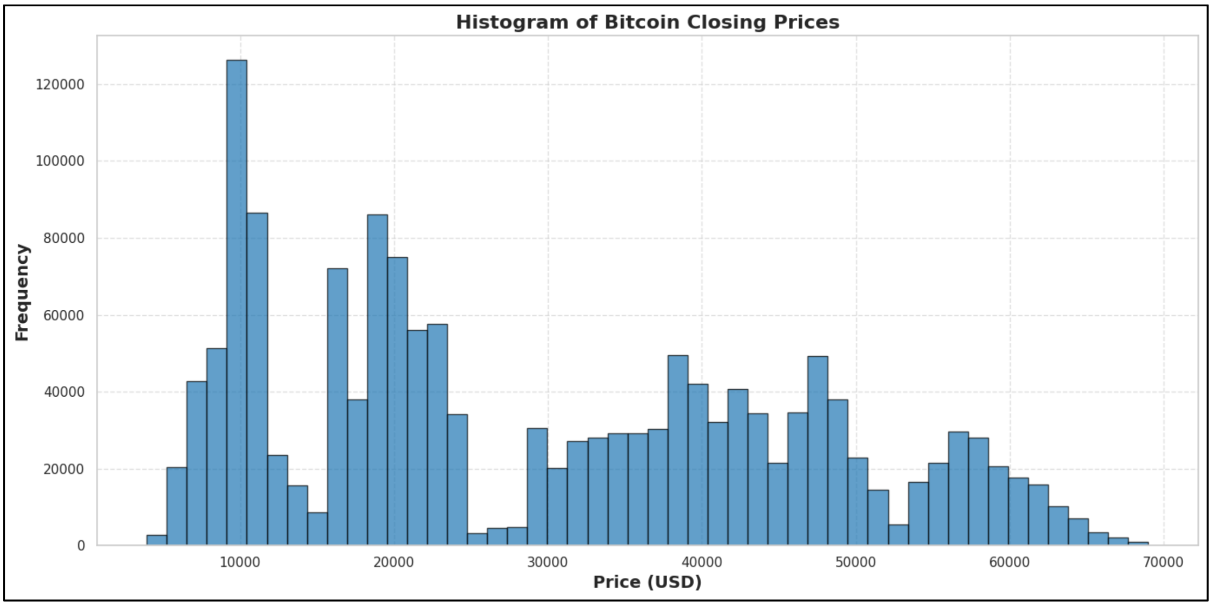
4.1. Visualizing Data

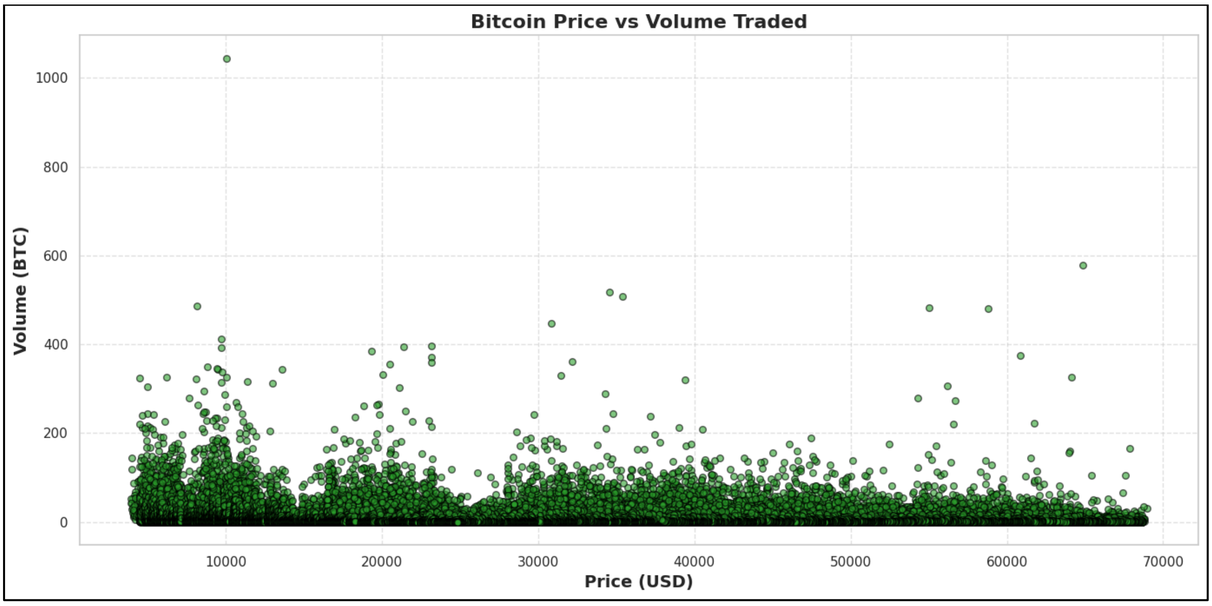



Bitcoin Price and Moving Averages
Close Price
5-Day Moving Average
30-Day Moving Average
Price (USD)
Date

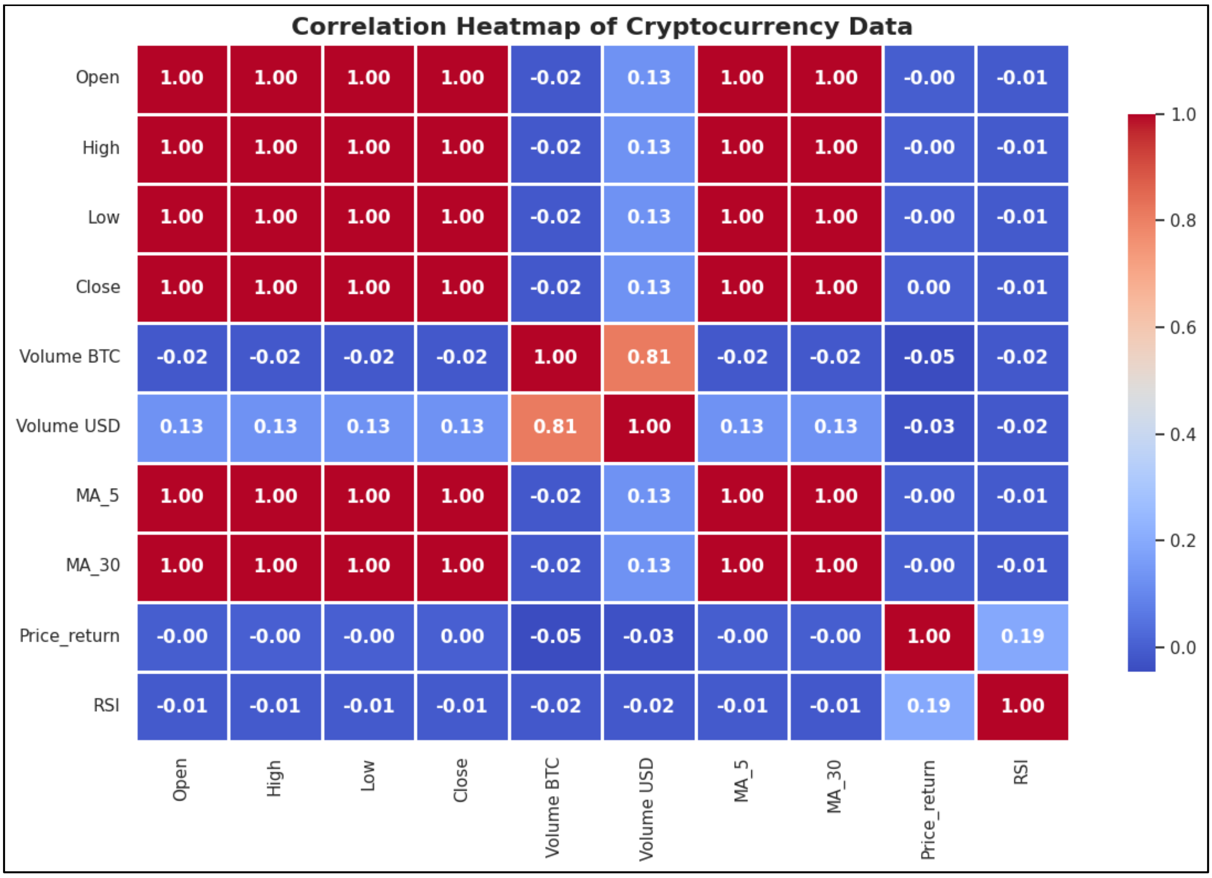

Bitcoin Price Returns Over Time
Price Return
Date


Histogram of Bitcoin RSI
Frequency
RSI Value





5. Feature Engineering and Selection



5.1. Feature Scaling

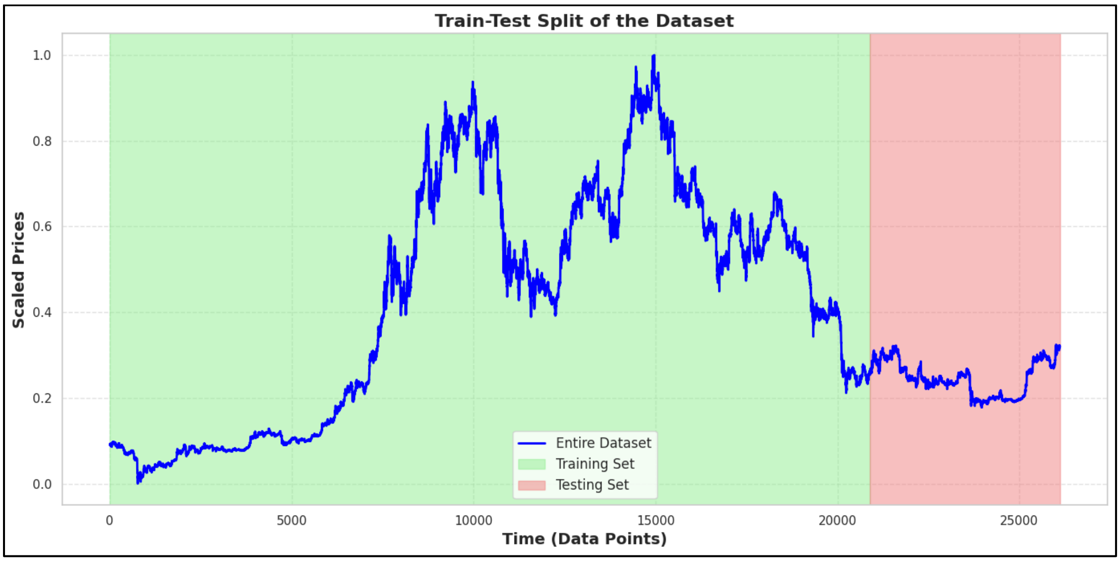
Normalize the data using MinMaxScaler for the machine learning models.

```
# Normalize the data for machine transformer model
scaler_baseline = MinMaxScaler(feature_range=(0, 1))
prices_scaled_baselines = scaler_baseline.fit_transform(prices_baselines)
```

5.2. Data Splitting

Split the data into training and testing sets. The split should be 80% for training and 20% for testing.

```
# Split the data into training and testing sets for machine learning model
baseline_train_size = int(len(prices_scaled_baselines) * 0.8)
baseline_train, baseline_test = prices_scaled_baselines[0:baseline_train_size, :], prices_scaled_baselines[b
```



6. Model Implementation

6.1. Baseline Models

6.1.1. Support Vector Machine (SVM)

```
# Define a parameter grid for simple tuning
param_grid = {
    'C': [0.01, 0.1],  # Regularization parameter
    'kernel': ['linear']  # Kernel type
}
```

```
# Set up GridSearchCV to perform simple tuning
svm_model = SVR(max_iter=60)

grid_search = GridSearchCV(svm_model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(baseline_trainX.reshape(baseline_trainX.shape[0], -1), baseline_trainY)

# Get the best model
best_svm_model = grid_search.best_estimator_

# Predict with the tuned SVM model
svm_predictions = best_svm_model.predict(baseline_testX.reshape(baseline_testX.shape[0], -1))

# Inverse transform the predictions for SVM Model
svm_predictions_inv = scaler_baseline.inverse_transform(svm_predictions.reshape(-1, 1))
baseline_testY_rescaled = scaler_baseline.inverse_transform(baseline_testY.reshape(-1, 1))
```

6.1.2. Random Forest

```python
# Define a parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [30, 70],  # Number of trees in the forest
    'max_depth': [1, 3]         # Maximum depth of each tree
}

# Set up GridSearchCV to perform tuning
rf_model = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(rf_model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(baseline_trainX.reshape(baseline_trainX.shape[0], -1), baseline_trainY)

# Get the best model
best_rf_model = grid_search.best_estimator_

# Predict with the tuned Random Forest model
rf_predictions = best_rf_model.predict(baseline_testX.reshape(baseline_testX.shape[0], -1))

# Inverse transform the predictions for Random Forest
rf_predictions_inv = scaler_baseline.inverse_transform(rf_predictions.reshape(-1, 1))
baseline_testY_rescaled = scaler_baseline.inverse_transform(baseline_testY.reshape(-1, 1))
```

6.1.3. Gradient Boosting

```python
# Define a simple parameter grid for tuning
param_grid = {
    'n_estimators': [10, 20],  # Number of boosting stages
    'max_depth': [5, 10],      # Maximum depth of individual trees
}

# Set up GridSearchCV to perform hyperparameter tuning
gbm_model = GradientBoostingRegressor(random_state=42)

grid_search = GridSearchCV(gbm_model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(baseline_trainX.reshape(baseline_trainX.shape[0], -1), baseline_trainY)

# Get the best model
best_gbm_model = grid_search.best_estimator_

# Predict with the tuned Gradient Boosting model
gbm_predictions = best_gbm_model.predict(baseline_testX.reshape(baseline_testX.shape[0], -1))

# Inverse transform the predictions for Gradient Boosting
gbm_predictions_inv = scaler_baseline.inverse_transform(gbm_predictions.reshape(-1, 1))
baseline_testY_rescaled = scaler_baseline.inverse_transform(baseline_testY.reshape(-1, 1))
```

6.1.4. Recurrent Neural Network (RNN)

```
# Recurrent Neural Network Model
rnn_model = Sequential()
rnn_model.add(SimpleRNN(units=10, return_sequences=False, input_shape=(baseline_trainX.shape[1], 1)))
rnn_model.add(Dense(units=1))  # Output layer for regression
rnn_model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Train the RNN model
rnn_model.fit(baseline_trainX, baseline_trainY, epochs=1, batch_size=128)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1734884868.579244     568 service.cc:145] XLA service 0x5d51506e81b0 initialized for platform
I0000 00:00:1734884868.579291     568 service.cc:153]   StreamExecutor device (0): Tesla T4, Compute Capab
I0000 00:00:1734884868.579295     568 service.cc:153]   StreamExecutor device (1): Tesla T4, Compute Capab
  3/161 ━━━━━━━━━━━━━━━━━━━━ 7s 47ms/step - loss: 0.0379
I0000 00:00:1734884869.314869     568 device_compiler.h:188] Compiled cluster using XLA!  This line is log
161/161 ━━━━━━━━━━━━━━━━━━━━ 6s 28ms/step - loss: 0.0086

<keras.src.callbacks.history.History at 0x7908fcb48d90>
```

```
# Predict with RNN
rnn_predictions = rnn_model.predict(baseline_testX)

# Inverse transform the RNN predictions
rnn_predictions_inv = scaler_baseline.inverse_transform(rnn_predictions)
```

6.1.4. Long Short Term Memory (LSTM)

```
# LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(units=10, return_sequences=False, input_shape=(baseline_trainX.shape[1], 1)))
lstm_model.add(Dense(units=1))  # Output layer for regression
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Train the LSTM model
lstm_model.fit(baseline_trainX, baseline_trainY, epochs=1, batch_size=128)
```

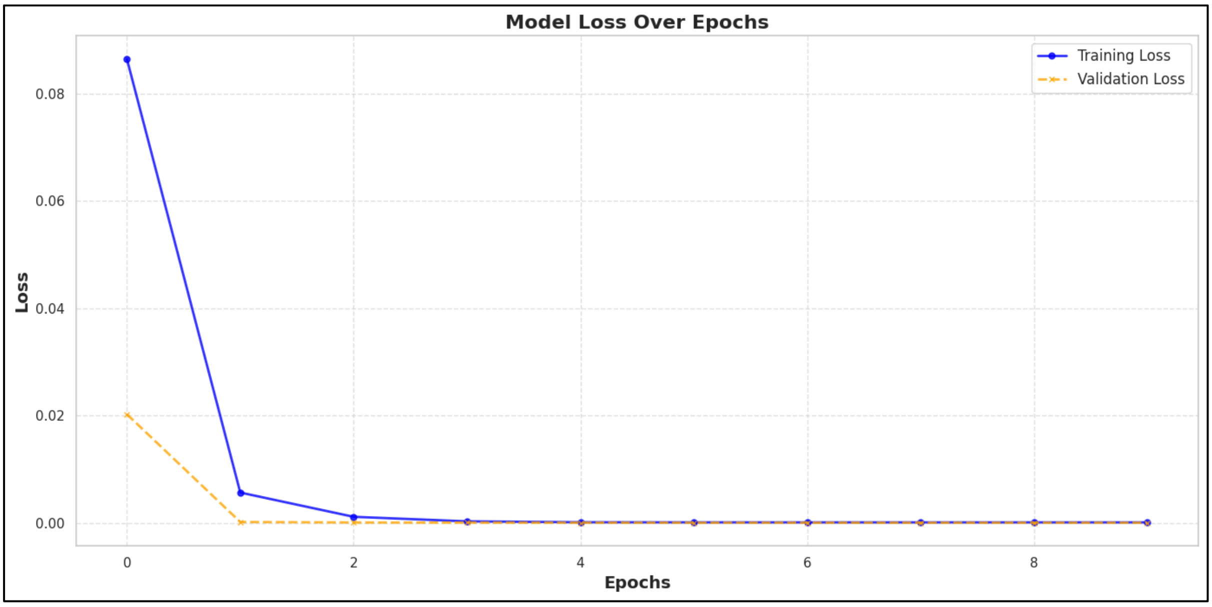
```
161/161 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.0976

<keras.src.callbacks.history.History at 0x7908f8f9f130>
```

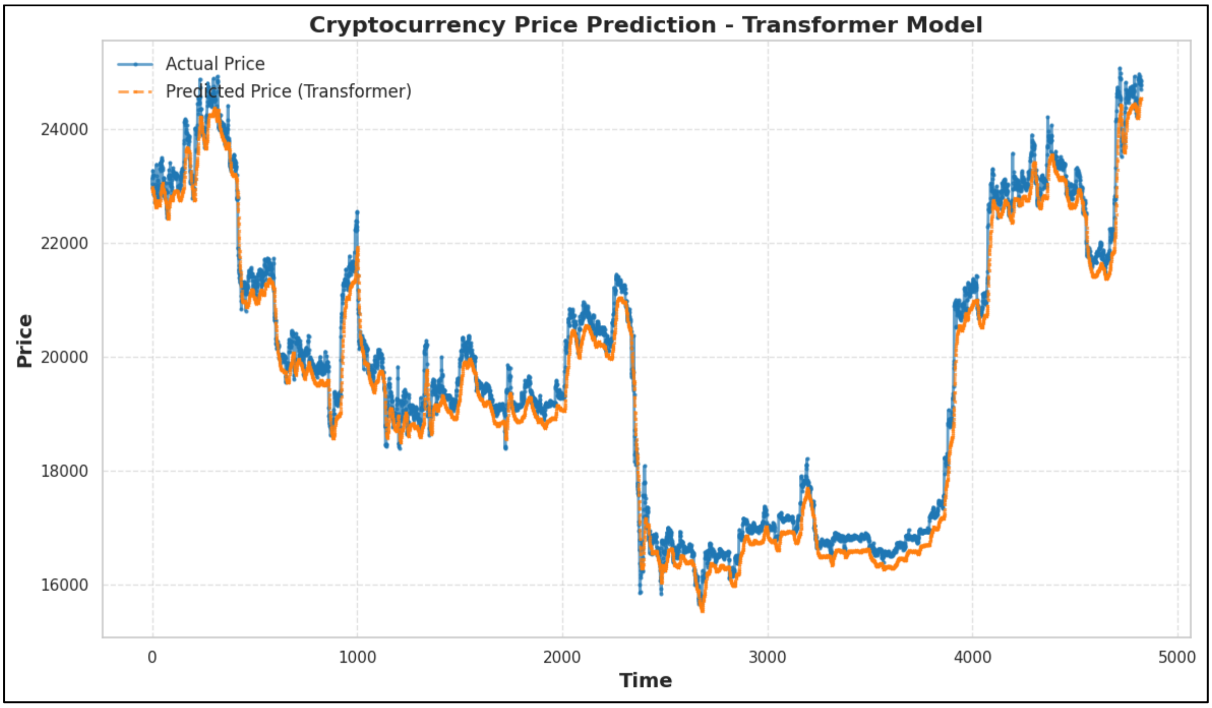
```
# Predict with LSTM
lstm_predictions = lstm_model.predict(baseline_testX)

# Inverse transform the LSTM predictions
lstm_predictions_inv = scaler_baseline.inverse_transform(lstm_predictions)
```

6.2. Transformer Model

Implement a Transformer-based architecture using the MultiHeadAttention layer.

```python
# Transformer Encoder Model using MultiHeadAttention (Time-Series Forecasting Transformer)
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    x = LayerNormalization(epsilon=1e-6)(inputs)
    x = MultiHeadAttention(key_dim=head_size, num_heads=num_heads, dropout=dropout)(x, x)
    x = Dropout(dropout)(x)
    res = Add()([x, inputs])

    x = LayerNormalization(epsilon=1e-6)(res)
    x = Dense(ff_dim, activation="relu")(x)
    x = Dropout(dropout)(x)
    x = Dense(inputs.shape[-1])(x)
    return Add()([x, res])

# Input shape based on the time-series data (look-back period)
input_shape = baseline_trainX.shape[1:]  # Time steps and features

# Create the input layer
inputs = Input(shape=input_shape)

# Apply transformer encoder to the inputs
x = transformer_encoder(inputs, head_size=256, num_heads=4, ff_dim=4, dropout=0.1)

# Use GlobalAveragePooling1D to pool the sequence's features into a fixed-size vector
x = GlobalAveragePooling1D()(x)

# Add Dense layers
x = Dropout(0.1)(x)
x = Dense(128, activation="relu")(x)
x = Dropout(0.1)(x)

# Output layer for regression task (predicting the Bitcoin price)
outputs = Dense(1)(x)

# Build the model
model_transformer = Sequential()
model_transformer.add(LSTM(units=50, input_shape=(1, look_back)))
model_transformer.add(Dense(units=1))

# Compile the model
model_transformer.compile(optimizer=Adam(learning_rate=0.0001), loss='mean_squared_error')

# Early stopping for model training
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```





7. Model Evaluation

Evaluate the models using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score.

Model Evaluation Metrics DataFrame:

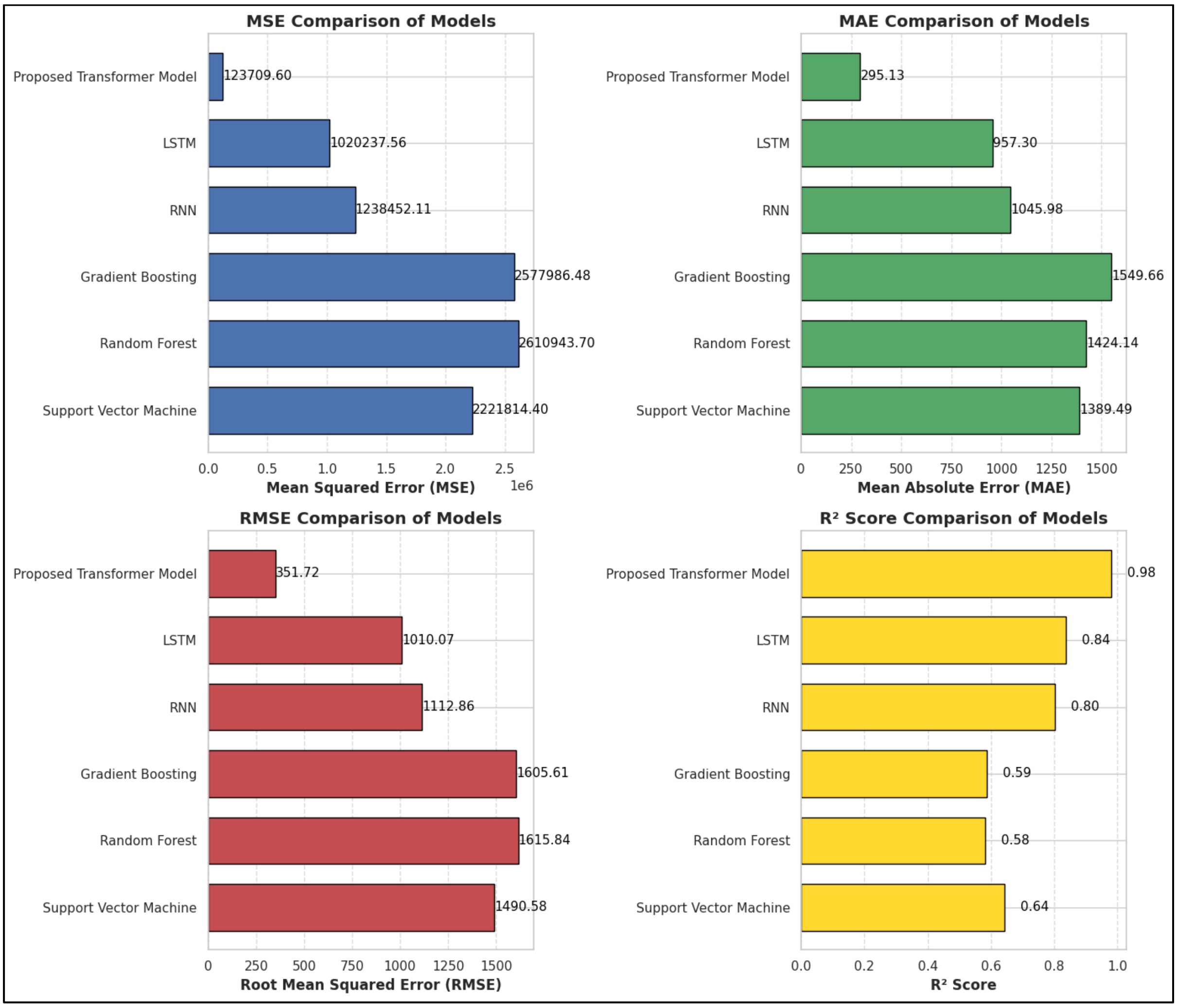
	Model	MSE	MAE	RMSE	R2
0	Support Vector Machine	2.221814e+06	1389.489556	1490.575193	0.643735
1	Random Forest	2.610944e+06	1424.142772	1615.841482	0.581338
2	Gradient Boosting	2.577986e+06	1549.664618	1605.610937	0.586623
3	RNN	1.238452e+06	1045.977701	1112.857633	0.801416
4	LSTM	1.020238e+06	957.303268	1010.068098	0.836406
5	Proposed Transformer Model	1.237096e+05	295.126108	351.723755	0.980163

8. Model Comparison

After evaluating all the models, we want to compare their performance on the various metrics (MSE, MAE, RMSE, R^2). This step helps identify which model performs best for the cryptocurrency price prediction task.

8.1 Visualization of Model Comparison

This will generate a comparison chart for the MSE, MAE, RMSE, and R^2 scores across the different models:



Conclusion

This manual provides a detailed framework for predicting cryptocurrency prices using various machine learning models, from traditional algorithms like Support Vector Machines and Random Forests to advanced models like RNNs, LSTMs, and Transformer-based architectures. The Transformer model, due to its ability to capture long-term dependencies, demonstrated superior performance in predicting cryptocurrency prices compared to other models. While traditional models not much performed well, they struggled to effectively model the time-series nature of the data. This approach offers a solid foundation for building advanced systems in cryptocurrency prediction, with potential applications in automated trading and risk management.

References

Python: https://www.python.org
Dataset Source: https://www.kaggle.com/datasets/swaptr/bitcoin-historical-data

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. A., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. NeurIPS.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780.

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. ICLR.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.