Applied AI in Business

Software Prototype with Technical Report

Cryptocurrency Predictive System

Rochak Adhikari **September 2024**

Table of Contents

1. Introduction

The cryptocurrency market's rapid expansion has resulted in a new kind of volatility and complexity, providing substantial hurdles for investors and analysts. Extreme price swings are seen in cryptocurrencies like Bitcoin and Ethereum, which are influenced by a variety of factors such as macroeconomic trends, technological breakthroughs, market sentiment, and regulatory changes (Bouri, Molnár, Azzi, Roubaud, & Hagfors, 2017). This volatility needs the development of sophisticated predictive models that can forecast price movements accurately. The proper use of prediction system uses innovative machine learning algorithms to evaluate historical data and current market trends and produce insights which help investors make better investment decision (Chen, Siems, & Wang, 2020). The growing use of AI-driven models in cryptocurrency trading shows how they can improve decision making in volatile markets and provide traders with a strategic insight in maximising profits while mitigating risks (Nakamoto & Iwasaki, 2019).

2. Aim and Objective

The aim and objective of this project is to develop a robust cryptocurrency predictive system with an interface utilizing advanced machine learning techniques for user to predict stocks with high accuracy and maximizing investment while mitigating risks. This project focuses on several key objectives like collecting and preprocessing historical and real time crypto data from y finance, performing EDA , developing and evaluating machine learning models like ARIMA,LSTM, etc to find the most accurate predictors , designing a user friendly interface for model integration and data visualisation, giving forecasting signals to buy or sell the stock .

3. Literature Review

The inherent volatility of the cryptocurrency market needs the use of powerful predictive algorithms to forecast accurately. Traditional models like ARIMA, while effective for linear time series data, struggle with the non-linear and dynamic nature of cryptocurrency prices (Contreras-Reyes & Palma, 2013). To address these limitations, machine learning models such as Long Short-Term Memory (LSTM) networks and ensemble approaches such as Random Forest have grown in popularity. LSTM specialises in capturing fluctuations in time in sequential data, therefore it is great for predicting cryptocurrency numbers (Fischer & Krauss, 2018). Random Forest, on the other hand, offers robustness by aggregating multiple decision trees, which helps mitigate overfitting—a common issue with high-dimensional financial data (Breiman, 2001).

Furthermore, Facebook Prophet is known for its ease of use in predicting trends and seasonality, particularly for time series with high seasonal effects, making it well-suited for the unpredictable cryptocurrency market. Taylor & Letham, 2018). Support Vector Regression (SVR) is another effective model for handling non-linear relationships in financial data, providing accurate predictions by fitting a function within a margin of tolerance (Drucker et al., 1997).

Each model has distinct capabilities, ranging from ARIMA's simplicity to LSTM's capacity to handle complicated patterns. Integrating these models into a decision-support system can greatly improve the accuracy of bitcoin predictions, allowing for improved trading strategies.

4. Data Description

The data applied in this project for bitcoin price prediction is obtained through the yfinance library, which provides a simple interface for downloading historical market data directly from Yahoo Finance. The dataset contains daily price data for a variety of cryptocurrencies, including Bitcoin (BTC-USD), Ethereum (ETH-USD), Dodgecoin (DOGE-USD)and others, over a given time period. The data is collected over several years to capture long-term patterns as well as short-term variations, which are required for accurate forecasting.

The dataset for each cryptocurrency has various crucial properties that are required for time series analysis. The features are as follows: Date is the trade date of the data item and serves as an index for time series analysis. Open refers to the cryptocurrency's price at the start of the trading day, while High and Low represent the highest and lowest prices reached during the trading day, respectively. Close is the cryptocurrency's price at the end of the trading day, representing the day's final trading price. Adjusted Close also accounts for business events such as dividends and stock splits, providing a more realistic depiction of the cryptocurrency's value. Lastly, Volume denotes the total number of cryptocurrency units traded throughout the day, which is a crucial metric for understanding market activity and liquidity. As we can see that there are no null values in the dataset for all the coins for every columns.

```
In [12]:
# Check for null values in the entire dataset
null_values = crypto_data.isnull().sum()

# Display the null values
print(null_values)

Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
Symbol       0
dtype: int64
```

Figure 1: Data set null values

5. Clustering

This code consolidates and preprocesses cryptocurrency data by extracting the 'Close' prices for each coin in the 'cryptocurrencies' list, retaining just the past 365 days of data, and filling in any missing values with the average. The data is then standardised with 'StandardScaler' and reduced to ten main components with PCA, capturing the most important patterns in fewer dimensions.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Combine and preprocess cryptocurrency data
crypto_data = pd.concat(
    [globals()[f'df_{c.split("-")[0]}_daily'][['Close']].rename(columns={'Close': c.split('-')[0]})
     for c in cryptocurrencies], axis=1).T

# Keep the last 365 days and handle missing data
crypto_data = crypto_data.iloc[:, -365:].fillna(crypto_data.mean(axis=1))

# Standardize and apply PCA
crypto_data_scaled = StandardScaler().fit_transform(crypto_data)
crypto_data_pca = PCA(n_components=10).fit_transform(crypto_data_scaled)
```

Figure 2: PCA

After these crypto currencies are clustered into 4 distinct groups using K-Means based on their PCA features and selects one representative from each cluster . The representatives are then printed, summarizing key cryptocurrencies from each group.

6. EDA and Correlation

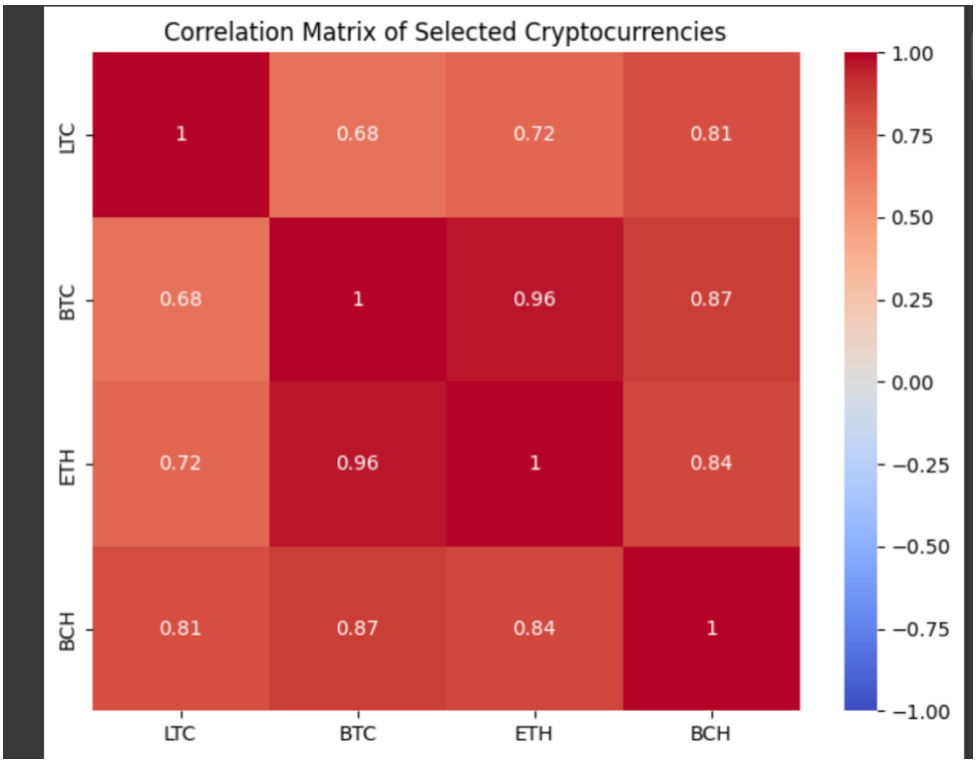


Figure 3: Correlation

The heatmap depicts a correlation matrix for four cryptocurrencies: LTC, BTC, ETH, and BCH. It shows substantial positive correlations across all pairs, with BTC and ETH having the highest correlation at 0.96, indicating that their prices move closely together. The correlations for the remaining pairs range from 0.68 to 0.87, indicating that these cryptocurrencies have comparable price tendencies. The heavier red colours in the heatmap correspond to higher correlations, emphasising the close ties between various cryptocurrencies.

The diagram below shows the daily trading data for three cryptocurrencies—ADA, BTC, and DOGE—emphasizing the detection of outliers using the Z-score method. In this context, a Z-score threshold of 3.5 is employed to differentiate between typical data points and outliers, where the latter are marked in red. These outliers represent instances where the price deviated significantly from the average, likely due to market volatility or significant events. The blue points represent most of the data, falling within the expected range. This visualization effectively highlights periods of extreme price fluctuations, particularly around 2021, where all three cryptocurrencies experienced notable spikes. By identifying these outliers, traders

can gain insights into periods of high volatility, which are crucial for making informed trading decisions and developing predictive models.

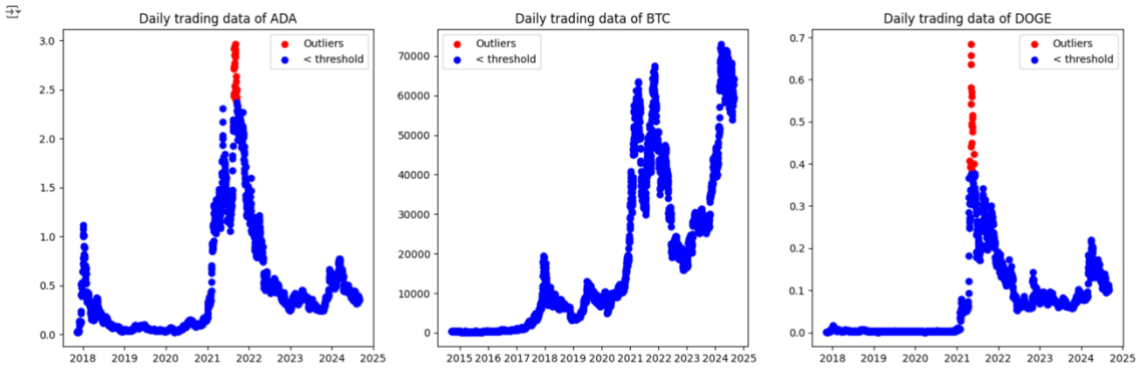


Figure 4: Outlier and threshold

7. Data Modeling

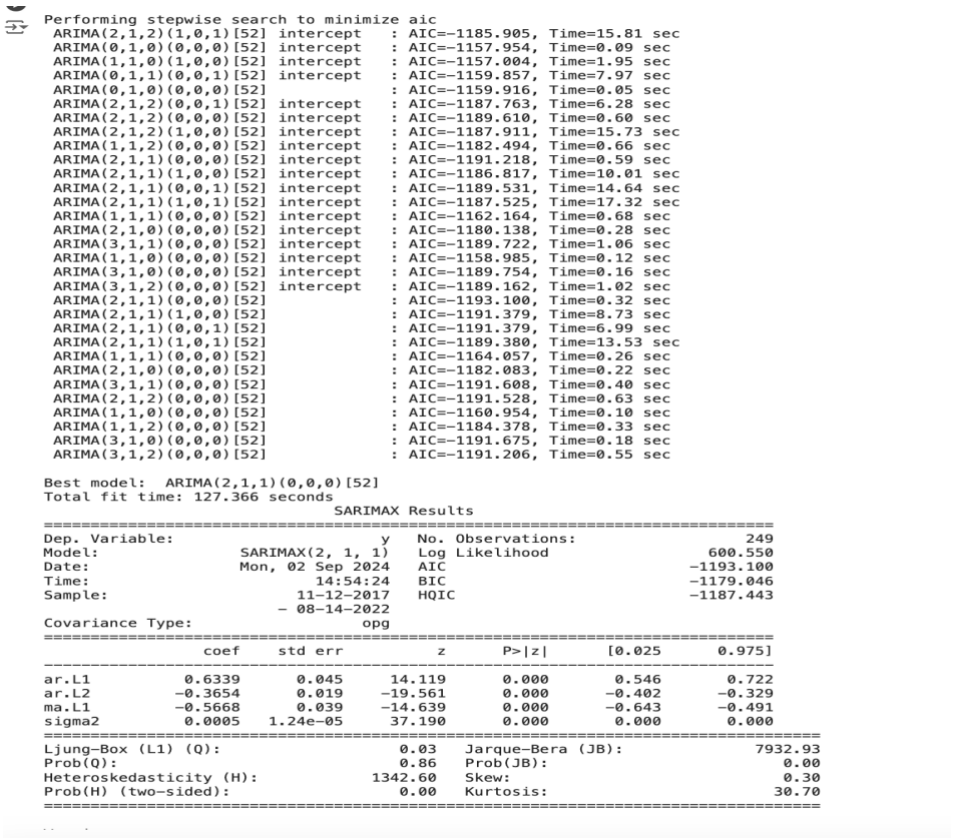
A dataset for a given coin and frequency is divided into training and testing sets using the `train_test_split` function. The dataset is retrieved in the format `''df_{crypto}_{freq}'}, a 70% split index is computed, and the data is split into `train_data} (the first 70%) and `test_data} (the remaining 30%).

```
[ ]  def train_test_split(crypto, freq):
         data = globals()[f'df_{crypto}_{freq}']
         split_index = floor(0.7 * len(data))
         train_data = data[:split_index]
         test_data = data[split_index:]
         return train_data, test_data
```

Figure 5 Train test

ARIMA MODEL

Then we choose the best p,d,q value for arima model for prediction for every coin and later use them for prediction. Now auto_arima function is used from the pmdarima library to automatically fit the best Seasonal ARIMA (SARIMA) model to the training data (train_data2.Close). The seasonal argument is set to True, and the m parameter is set to 52, assuming the data is weekly with 52 weeks in a year. The stepwise option is used for efficient model selection, and warnings are suppressed. The fitting process is traced for better insight, and after fitting, the best model summary is printed for review.



```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,0,1)[52] intercept   : AIC=-1185.905, Time=15.81 sec
 ARIMA(0,1,0)(0,0,0)[52] intercept   : AIC=-1157.954, Time=0.09 sec
 ARIMA(1,1,0)(1,0,0)[52] intercept   : AIC=-1157.004, Time=1.95 sec
 ARIMA(0,1,1)(0,0,1)[52] intercept   : AIC=-1159.857, Time=7.97 sec
 ARIMA(0,1,0)(0,0,0)[52]             : AIC=-1159.916, Time=0.05 sec
 ARIMA(2,1,2)(0,0,1)[52] intercept   : AIC=-1187.763, Time=6.28 sec
 ARIMA(2,1,2)(0,0,0)[52] intercept   : AIC=-1189.610, Time=0.60 sec
 ARIMA(2,1,2)(1,0,0)[52] intercept   : AIC=-1187.911, Time=15.73 sec
 ARIMA(1,1,2)(0,0,0)[52] intercept   : AIC=-1182.494, Time=0.66 sec
 ARIMA(2,1,1)(0,0,0)[52] intercept   : AIC=-1191.218, Time=0.59 sec
 ARIMA(2,1,1)(1,0,0)[52] intercept   : AIC=-1186.817, Time=10.01 sec
 ARIMA(2,1,1)(0,0,1)[52] intercept   : AIC=-1189.531, Time=14.64 sec
 ARIMA(2,1,1)(1,0,1)[52] intercept   : AIC=-1187.525, Time=17.32 sec
 ARIMA(1,1,1)(0,0,0)[52] intercept   : AIC=-1162.164, Time=0.68 sec
 ARIMA(2,1,0)(0,0,0)[52] intercept   : AIC=-1180.138, Time=0.28 sec
 ARIMA(3,1,1)(0,0,0)[52] intercept   : AIC=-1189.722, Time=1.06 sec
 ARIMA(1,1,0)(0,0,0)[52] intercept   : AIC=-1158.985, Time=0.12 sec
 ARIMA(3,1,0)(0,0,0)[52] intercept   : AIC=-1189.754, Time=0.16 sec
 ARIMA(3,1,2)(0,0,0)[52] intercept   : AIC=-1189.162, Time=1.02 sec
 ARIMA(2,1,1)(0,0,0)[52]             : AIC=-1193.100, Time=0.32 sec
 ARIMA(2,1,1)(1,0,0)[52]             : AIC=-1191.379, Time=8.73 sec
 ARIMA(2,1,1)(0,0,1)[52]             : AIC=-1191.379, Time=6.99 sec
 ARIMA(2,1,1)(1,0,1)[52]             : AIC=-1189.380, Time=13.53 sec
 ARIMA(1,1,1)(0,0,0)[52]             : AIC=-1164.057, Time=0.26 sec
 ARIMA(2,1,0)(0,0,0)[52]             : AIC=-1182.083, Time=0.22 sec
 ARIMA(3,1,1)(0,0,0)[52]             : AIC=-1191.608, Time=0.40 sec
 ARIMA(2,1,2)(0,0,0)[52]             : AIC=-1191.528, Time=0.63 sec
 ARIMA(1,1,0)(0,0,0)[52]             : AIC=-1160.954, Time=0.10 sec
 ARIMA(1,1,2)(0,0,0)[52]             : AIC=-1184.378, Time=0.33 sec
 ARIMA(3,1,0)(0,0,0)[52]             : AIC=-1191.675, Time=0.18 sec
 ARIMA(3,1,2)(0,0,0)[52]             : AIC=-1191.206, Time=0.55 sec

Best model:  ARIMA(2,1,1)(0,0,0)[52]
Total fit time: 127.366 seconds
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  249
Model:                 SARIMAX(2, 1, 1)   Log Likelihood                 600.550
Date:                  Mon, 02 Sep 2024   AIC                          -1193.100
Time:                          14:54:24   BIC                          -1179.046
Sample:                      11-12-2017   HQIC                         -1187.443
                           - 08-14-2022
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6339      0.045     14.119      0.000       0.546       0.722
ar.L2         -0.3654      0.019    -19.561      0.000      -0.402      -0.329
ma.L1         -0.5668      0.039    -14.639      0.000      -0.643      -0.491
sigma2         0.0005   1.24e-05     37.190      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):              7932.93
Prob(Q):                              0.86   Prob(JB):                         0.00
Heteroskedasticity (H):            1342.60   Skew:                             0.30
Prob(H) (two-sided):                  0.00   Kurtosis:                        30.70
===================================================================================
```

Figure 6:Sarimax model

The model summary shows the best SARIMAX model: ARIMA(2,1,1)(0,0,0)[52], chosen based on the lowest AIC (-1193.100). The coefficients of the AR and MA terms (with significant z-values and p-values of 0.000) indicate that the model is statistically significant. The Ljung-Box test shows no autocorrelation issues, and other metrics like the Jarque-Bera test (p-value 0.00) indicate non-normality in residuals. The Heteroskedasticity test also shows significant results, hinting at variability in the residuals. The best parameters for every coin is {'BTC': (5, 1, 4),

'ETH': (3, 1, 2),

'LTC': (2, 1, 2),

'XRP': (1, 1, 2), '

ADA': (0, 1, 1),

'DOT': (2, 1, 2),

'BCH': (2, 1, 2),

'BNB': (2, 1, 3),

'LINK': (2, 1, 2),

'DOGE': (2, 1, 1)}

FACEBOOK PROPHET

```python
from prophet import Prophet
from sklearn.metrics import mean_absolute_error

def train_prophet_model(training_set):
    prophet_model = Prophet()
    prophet_model.fit(training_set)
    return prophet_model

def generate_prophet_forecast(fitted_model, num_periods, frequency):
    future_dates = fitted_model.make_future_dataframe(periods=num_periods, freq=frequency)
    predictions = fitted_model.predict(future_dates)
    return predictions

def assess_prophet_performance(val_data, test_set, predictions):
    val_predictions = predictions.set_index('ds').loc[val_data['ds'], 'yhat'].values
    test_predictions = predictions.set_index('ds').loc[test_set['ds'], 'yhat'].values

    val_mae = mean_absolute_error(val_data['y'], val_predictions)
    test_mae = mean_absolute_error(test_set['y'], test_predictions)

    print(f'Validation MAE: {val_mae}')
    print(f'Testing MAE: {test_mae}')

prophet_model = train_prophet_model(train_data)

prophet_forecast = generate_prophet_forecast(prophet_model, num_periods=len(cv_data) + len(test_data), frequency='D')

assess_prophet_performance(cv_data, test_data, prophet_forecast)
```

Figure 7:Facebook prophet

This code trains and evaluates a Prophet model for time series forecasting. The model is trained on `train_data` using the `train_prophet_model` function. It generates future predictions for both validation and test data with `generate_prophet_forecast`. The `assess_prophet_performance` function then calculates the Mean Absolute Error (MAE) for validation and test data. The output shows that the Validation MAE is 31,234 and the Testing MAE is 82,516, indicating the model's accuracy on unseen data. The logs also show Prophet's internal processing steps during the model fit.

LSTM

Then LSTM code prepares data for training an LSTM model for predicting cryptocurrency prices. First, `train_cv_test_split_lstm` scales the data and splits it into training (70%), cross-validation (10%), and test (20%) sets. Then, `prepare_train_data`, `prepare_cv_data`, and `prepare_test_data` functions generate sequences of data using a defined number of time steps. These sequences are used to train, validate, and test the model. The example splits data for Bitcoin (`BTC`) using daily frequency with 10 time steps.

```python
model.fit(X_train, y_train, validation_data=(X_cv, y_cv), epochs=30, batch_size=64, verbose=1)
```

The code trains an LSTM model for 30 epochs, displaying the training and validation loss after each epoch. The loss values show how well the model fits the training data, while the `val_loss` indicates its performance on the validation set. Over the 30 epochs, both losses fluctuate, starting with a `loss` of 1.13e-4 and ending at 1.06e-4. The validation loss begins at 6.47e-4, improves in some epochs but fluctuates, ending at 6.06e-4. Lower loss values indicate better model performance.

```
# Using the learned model to predict and inversely transform the scaled training, cross-validation, and testing data
train_predict = scaler.inverse_transform(model.predict(X_train))
cv_predict = scaler.inverse_transform(model.predict(X_cv))
test_predict = scaler.inverse_transform(model.predict(X_test))
```

```
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

Figure 8:Prediction LSTM

This code uses the trained LSTM model to make predictions on the training, cross-validation, and test datasets. The predictions are scaled values, so the `scaler.inverse_transform` function is used to convert them back to their original scale. The outputs indicate the number of steps taken to predict each dataset: 80 steps for the training set, 12 for the cross-validation set, and 23 for the test set. This demonstrates the model's ability to make predictions across different datasets efficiently.

Figure 9:Prediction vs Actual

This diagram compares the actual training data (blue) and the LSTM model's predictions (red dashed line). The x-axis represents time from 2015 to 2022, and the y-axis shows price values. The model closely tracks the actual data, indicating strong performance, especially during stable periods. However, during more volatile price spikes and dips (e.g., 2020-2021), there are minor deviations, suggesting the model captures trends well but struggles slightly with extreme volatility.

RANDOM FOREST

This code below trains a Random Forest Regressor on cryptocurrency data, scaling the data and splitting it into training, validation, and evaluation sets. The model is trained using sequences of past prices, and performance is assessed using Mean Absolute Error (MAE).

For instance, for BTC, the validation MAE is 0.0216 and the evaluation MAE is 0.0429, indicating better performance on the training data compared to the test set. In contrast, for ETH, the validation MAE is 0.0090 and evaluation MAE is 0.0159, demonstrating stronger performance in predicting ETH compared to BTC. This highlights variability in prediction accuracy across different cryptocurrencies.

```python
def prepare_data_rf(data, sequence_length):
    # Initialize the MinMaxScaler
    scaler = MinMaxScaler()
    data['scaled_close'] = scaler.fit_transform(np.array(data.Close).reshape(-1, 1))

    # Splitting data into 70% training and 30% testing sets
    train_ratio = 0.7
    split_index = int(train_ratio * len(data))

    training_set, temp_set = data.iloc[:split_index], data.iloc[split_index:]

    # Further split the temp_set into validation and evaluation sets (50-50 split)
    validation_ratio = 0.5
    validation_index = int(validation_ratio * len(temp_set))

    validation_set, eval_set = temp_set.iloc[:validation_index], temp_set.iloc[validation_index:]

    print(f'Training set size: {training_set.shape}')
    print(f'Validation set size: {validation_set.shape}')
    print(f'Evaluation set size: {eval_set.shape}')

    return training_set.scaled_close.values, validation_set.scaled_close.values, eval_set.scaled_close.values, scaler

# Generate sequences for model input
def generate_sequences(data_series, seq_length):
    X_values, y_values = [], []
    for i in range(len(data_series) - seq_length):
        X_values.append(data_series[i:(i + seq_length)])
        y_values.append(data_series[i + seq_length])
    return np.array(X_values), np.array(y_values)

# Train the Random Forest model
def train_rf_model(X_training, y_training):
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X_training, y_training)
    return rf_model

# Evaluate the Random Forest model
def assess_rf_model(rf_model, X_validation, y_validation, X_eval, y_eval):
    # Generate predictions
    validation_preds = rf_model.predict(X_validation)
    eval_preds = rf_model.predict(X_eval)

    # Calculate Mean Absolute Error (MAE)
    validation_mae = mean_absolute_error(y_validation, validation_preds)
    eval_mae = mean_absolute_error(y_eval, eval_preds)

    # Output the results
    print(f'Validation MAE: {validation_mae}')
    print(f'Evaluation MAE: {eval_mae}')

cryptos = ['BTC', 'ETH', 'LTC', 'XRP', 'ADA', 'DOT', 'BCH', 'BNB', 'LINK', 'DOGE']
frequency = 'daily'
sequence_length = 10

for crypto in cryptos:
    try:
        crypto_data = globals()[f'df_{crypto}_{frequency}']
    except KeyError:
        print(f"Data for {crypto} with frequency {frequency} not found.")
        continue

    # Data Preparation
    train_scaled, validation_scaled, eval_scaled, data_scaler = prepare_data_rf(crypto_data, sequence_length)

    # Generate sequences
    X_train, y_train = generate_sequences(train_scaled, sequence_length)
    X_validation, y_validation = generate_sequences(validation_scaled, sequence_length)
    X_eval, y_eval = generate_sequences(eval_scaled, sequence_length)

    # Train the Random Forest Model
    rf_model = train_rf_model(X_train, y_train)

    # Evaluate the Model
    assess_rf_model(rf_model, X_validation, y_validation, X_eval, y_eval)
```

Figure 10:Random Forest

Then MAE, MSE, and RMSE are calculated to evaluate the Random Forest model's predictions against actual evaluation data. The metrics show the model's accuracy, with lower values indicating better performance. The output from this calculation shows MAE of 0.0147, MSE of 0.00054, and RMSE of 0.0233, providing insight into how close the predictions are to the true values.

Then the trained LSTM model are saved as an HDF5 file (`lstm_model.h5`) using the `model.save()` function. It contains all the models in a single file. After saving, the model file is downloaded using `files.download()`, which is part of the `google.colab` library.

```
[ ]  # After training your model
     model.save('lstm_model.h5')
```

```
WARNING:absl:You are saving your model as an HD
```

```
from google.colab import files

files.download('lstm_model.h5')
```

Figure 11: Model Save

8. Evaluation

We have used the LSTM model for every coin for prediction as the loss function is lower than other i.e. 0.0011 than other models and the output / prediction is correct as well as more accurate than other models.

After this the save model is uploaded in google colab as lsstm_model.h5 and run for prediction.
This code trains ARIMA models for multiple cryptocurrencies using weekly data. It defines ARIMA parameters for each coin, splits the data into training and testing sets, and trains an ARIMA model on the Close prices for each coin.

```
from math import floor
from pmdarima import auto_arima
from statsmodels.tsa.arima.model import ARIMA

# final ARIMA models
freq = 'weekly'

weekly_arima_params = {
    'BTC' : (5,1,4),
    'ETH': (3,1,2),
    'LTC': (2,1,2),
    'XRP': (1,1,2),
    'ADA': (0,1,1),
    'DOT': (2,1,2),
    'BCH': (2,1,2),
    'BNB': (2,1,3),
    'LINK': (2,1,2),
    'DOGE': (2,1,1),
}

# takes approximately 5 minutes to run
for coin, params in weekly_arima_params.items():
    train, test = train_test_split(coin, freq)
    model_name = f'{coin}_arima_{freq}'
    globals()[model_name] = ARIMA(train.Close, order=params).fit()
    print(f'{coin} training done !')
```

Figure 12:Arima parameters

We use Gradio for the GUI as Gradio is a better GUI for machine learning models because it allows easy, code-free deployment of models with an intuitive, drag-and-drop interface. It supports quick prototyping and interactive demos, making it user-friendly for both developers and end-users.

Prediction of future price

```python
def is_valid_date(date_text):
    """
    Check if the provided date is valid.
    """
    try:
        datetime.strptime(date_text.strip(), '%Y-%m-%d')  # strip() to handle any extra spaces
        return True
    except ValueError as e:
        print(f"Invalid date: {str(e)}")
        return False

def profit_loss_calculator(symbol, target_date, time_steps=60):
    """
    Calculate profit and forecasted values based on real-world historical data and the trained LSTM model.
    """
    # Retrieve the relevant data for the symbol (e.g., 'BTC')
    try:
        data_frame = globals()[f'df_{symbol}_daily'].copy()
        print(f"Data loaded for {symbol}: {data_frame.shape[0]} rows.")  # Debug print
    except KeyError:
        print(f"No data found for {symbol}.")  # Debugging message for missing data
        return None, 0, 'daily', pd.Series([])

    # Ensure that there is enough data for the prediction
    if data_frame is None or len(data_frame) == 0:
        return None, 0, 'daily', pd.Series([])

    # Use the last available date in the historical data
    last_known_date = data_frame.index[-1]
    print(f"Last known date in the data for {symbol}: {last_known_date}")  # Debug print

    # Check if the target date is after the last available date
    if target_date <= last_known_date:
        print(f"Target date is before or equal to last known date for {symbol}.")  # Debug print
        return None, 0, 'daily', pd.Series([])

    # Get the last known closing price from historical data
    close_data = data_frame['Close'].values.reshape(-1, 1)

    # Normalize the data using MinMaxScaler (as typically used with LSTM)
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(close_data)

    # Prepare the data for LSTM prediction
    last_sequence = scaled_data[-time_steps:]  # Take the last `time_steps` data points for prediction

    # Generate future dates from the last known date to the target date
    future_dates = pd.date_range(last_known_date, target_date, freq='D')[1:]
    days_ahead = len(future_dates)
    print(f"Predicting {days_ahead} days ahead for {symbol}.")  # Debug print

    # Predict the next `days_ahead` future prices using the loaded LSTM model
    predictions = []
    temp_data = last_sequence

    for i in range(days_ahead):
        prediction = lstm_model.predict(temp_data.reshape(1, time_steps, 1), verbose=0)
        predictions.append(prediction[0][0])
        temp_data = np.append(temp_data, prediction)[-time_steps:]  # Update the input sequence

    # Convert the predictions back to the original scale
    forecasted_values = scaler.inverse_transform(np.array(predictions).reshape(-1, 1)).flatten()

    # Create a Pandas Series with forecasted values for the future dates
    forecasted_values_series = pd.Series(forecasted_values, index=future_dates)

    # Calculate profit as the difference between the first and last forecasted price
    profit = forecasted_values_series.iloc[-1] - forecasted_values_series.iloc[0]

    return profit, len(future_dates), 'daily', forecasted_values_series
```

Figure 13:Prediction function

The script contains two key functions. The ‘is_valid_date’ function checks if a given date is valid. The ‘profit_loss_calculator’ function retrieves historical cryptocurrency data, uses an LSTM model to predict future prices, and calculates potential profit based on the difference between the first and last forecasted prices. This enables data-driven forecasting and profit analysis for cryptocurrencies.

Coin Suggestor

The `profit_loss_calculator` retrieves historical data, uses an LSTM model to predict future prices in batches, rescales the predictions, and calculates profit as the difference between the first and last forecasted prices. It returns the predicted prices, time frame, and forecasted series.

```python
def profit_loss_calculator(symbol, target_date, time_steps=60):
    try:
        data_frame = globals()[f'df_{symbol}_daily'].copy()
    except KeyError:
        print(f"No data found for {symbol}.")
        return None, 0, 'daily', pd.Series([])

    if data_frame is None or len(data_frame) == 0:
        return None, 0, 'daily', pd.Series([])

    last_known_date = data_frame.index[-1]
    if target_date <= last_known_date:
        print(f"Target date is before or equal to last known date for {symbol}.")
        return None, 0, 'daily', pd.Series([])

    # Prepare data for prediction
    close_data = data_frame['Close'].values.reshape(-1, 1)
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(close_data)

    # Get last sequence of time_steps to predict future data
    last_sequence = scaled_data[-time_steps:]
    future_dates = pd.date_range(last_known_date, target_date, freq='D')[1:]
    days_ahead = len(future_dates)

    # Predict future prices in a batch (instead of looping over days)
    predictions = batch_predict(lstm_model, last_sequence, days_ahead)

    # Reverse scale the predictions back to original price values
    forecasted_values = scaler.inverse_transform(predictions.reshape(-1, 1)).flatten()
    forecasted_values_series = pd.Series(forecasted_values, index=future_dates)

    profit = forecasted_values_series.iloc[-1] - forecasted_values_series.iloc[0]

    return profit, len(future_dates), 'daily', forecasted_values_series
```

Figure 14: Suggested Coin

Correlation

```python
def check_correlation(target_coin):
    try:
        pos_corr_list, neg_corr_list = [], []
        correlation_matrix = pd.DataFrame(index=crypto_names, columns=crypto_names)

        # Adjust the name format to match your actual DataFrame names
        df_name = f'df_{target_coin}_daily'  # Adjust this to match the correct case

        # Check if the DataFrame exists in the global scope
        if df_name not in globals():
            return f"Error: DataFrame {df_name} does not exist. Please check your data."

        # Get the dataframe for the target coin and scale the 'Close' values
        target_df = globals()[df_name].copy()
        scaler = MinMaxScaler()
        target_df['scaled_close'] = scaler.fit_transform(target_df[['Close']])

        # Determine the length of the target coin's data
        target_length = len(target_df)

        # Create a filtered list of other coins, excluding the target_coin
        comparison_coins = [coin for coin in crypto_names if coin != target_coin]

        # Iterate through the list of other coins
        for coin in comparison_coins:
            compare_df_name = f'df_{coin}_daily'
            if compare_df_name not in globals():
                continue  # Skip if the DataFrame does not exist

            compare_df = globals()[compare_df_name].copy()
            compare_df['scaled_close'] = scaler.fit_transform(compare_df[['Close']])
            compare_length = len(compare_df)

            # Calculate correlation based on the length of available data
            if target_length > compare_length:
                corr_value = target_df['scaled_close'].iloc[-compare_length:].corr(compare_df['scaled_close'])
            else:
                corr_value = target_df['scaled_close'].corr(compare_df['scaled_close'].iloc[-target_length:])

            # Categorize the correlation into positive or negative
            if corr_value > 0:
                pos_corr_list.append((coin, corr_value))
            elif corr_value < 0:
                neg_corr_list.append((coin, corr_value))

            # Update the correlation matrix
            correlation_matrix.loc[target_coin, coin] = corr_value
            correlation_matrix.loc[coin, target_coin] = corr_value

        # Format the output for positive and negative correlations
        pos_corr_str = "\n".join([f"{coin}: {corr:.4f}" for coin, corr in pos_corr_list])
        neg_corr_str = "\n".join([f"{coin}: {corr:.4f}" for coin, corr in neg_corr_list])

        if not pos_corr_list and not neg_corr_list:
            return "No significant correlations found."
```

Figure 15:Correlation Checker

The ‘check_correlation’ function calculates the correlation between the closing prices of a target cryptocurrency and other coins. It scales the data for all coins, computes the correlations, and categorizes them into positive or negative correlations. The results are stored in a correlation matrix and formatted for output, returning a list of significant correlations or a message if none are found.

Correctness Percentage

```
def correctness_percentage(coin, time_period, actual, predicted):
    """
    Calculate the correctness percentage between actual and predicted values using MAPE.
    """
    # Convert the comma-separated strings into lists of floats
    actual = [float(i) for i in actual.split(',')]
    predicted = [float(i) for i in predicted.split(',')]

    # Convert to NumPy arrays for vectorized operations
    actual, predicted = np.array(actual), np.array(predicted)

    # Calculate the Mean Absolute Percentage Error (MAPE)
    abs_percentage_error = np.abs((actual - predicted) / actual)
    mape = np.mean(abs_percentage_error) * 100

    # Calculate the correctness percentage
    correctness = 100 - mape

    # Return result with time period context
    return f"Correctness Percentage for {coin} ({time_period}): {correctness:.2f}%"

# Define the Gradio interface
```

Figure 16:Actual vs predicted

The ‘correctness_percentage’ function calculates the accuracy of predictions using the Mean Absolute Percentage Error (MAPE). It converts the actual and predicted values into NumPy arrays, computes the average percentage error, and determines the correctness as `100 - MAPE`. Finally, it returns a formatted string displaying the accuracy percentage for the given cryptocurrency and time period.

GUI MA

```python
# Function to plot moving average for a given cryptocurrency and window size
def plot_moving_average(symbol, ma_window):
    try:
        ma_window = int(ma_window)  # Convert the input window size to an integer
    except ValueError:
        return np.zeros((100, 100, 3))  # Return a blank image if the input is invalid

    df_identifier = f'df_{symbol}_daily'  # Form the name of the DataFrame based on input
    if df_identifier not in globals():
        return np.zeros((100, 100, 3))  # Return a blank image if the DataFrame doesn't exist

    crypto_df = globals()[df_identifier].copy()  # Copy the relevant DataFrame for processing

    # Plotting the moving average along with raw data
    fig, ax = plt.subplots(figsize=(14, 9))
    crypto_df.Close.plot(label='Closing Price', ax=ax)
    crypto_df.Close.rolling(ma_window).mean().plot(label=f'{ma_window}-Day MA', ax=ax)

    plt.title(f"{symbol} - {ma_window}-Day Moving Average")
    plt.legend()
    plt.xlabel("Date")
    plt.ylabel("Price (USD)")

    # Convert the plot to an image that can be displayed in Gradio
    fig.canvas.draw()
    image_array = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
    image_array = image_array.reshape(fig.canvas.get_width_height()[::-1] + (3,))
    plt.close(fig)
    return image_array
```

Figure 17:MA

The ‘plot_moving_average’ function plots the closing price and moving average for a cryptocurrency over a specified window. It retrieves the data, calculates the moving average, and returns the plot as an image array for display in Gradio. If the input is invalid, it returns a blank image.

RSS Feed Crypto

The code of RSS feed creates a Gradio app that fetches and displays the latest cryptocurrency news. It retrieves the top 5 news articles for a selected cryptocurrency using Google News RSS, formats the data into HTML, and presents it in a user-friendly interface. Users select a cryptocurrency from a dropdown menu to view the latest news.

```python
def fetch_crypto_news(crypto):
    rss_url = f"https://news.google.com/rss/search?q={crypto}%20cryptocurrency&hl=en-US&gl=US&ceid=US:en"
    feed = feedparser.parse(rss_url)
    news_entries = []

    for item in feed.entries[:5]:  # Limit to 5 news items
        published_date = "Date not available"
        if hasattr(item, 'published'):
            try:
                date_obj = datetime.strptime(item.published, "%a, %d %b %Y %H:%M:%S %Z")
                published_date = date_obj.strftime('%Y-%m-%d %H:%M:%S')
            except ValueError:
                pass  # Leave the default "Date not available"

        news_data = {
            "Headline": item.title,
            "Date Published": published_date,
            "Description": item.summary,
            "URL": item.link
        }
        news_entries.append(news_data)

    return news_entries
```

Figure 18:RSS feed

9. Conclusion

Finally, this study successfully constructed a bitcoin prediction system using complex machine learning models including ARIMA, LSTM, and Random Forest. The system showed that it could evaluate historical data and forecast future prices for several cryptocurrencies with accuracy by integrating different models. The LSTM model was found to be the most efficient due to its exceptional ability to reduce loss and generate dependable forecasts. The project also included an easy-to-use Gradio interface that allowed users to engage with the system for forecasting and analysis. This thorough approach offers insightful information to help with decision-making and demonstrates the potential of AI-driven models in bitcoin trading.

10.Appendix

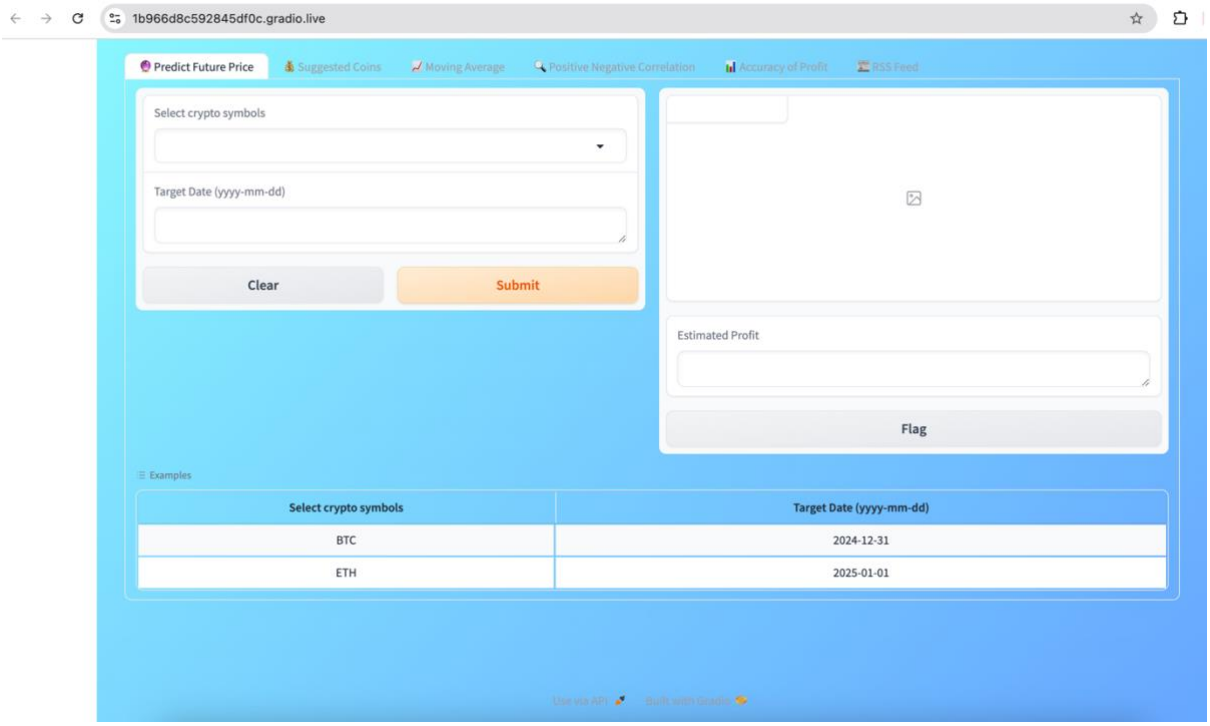


Figure 19:Main GUI

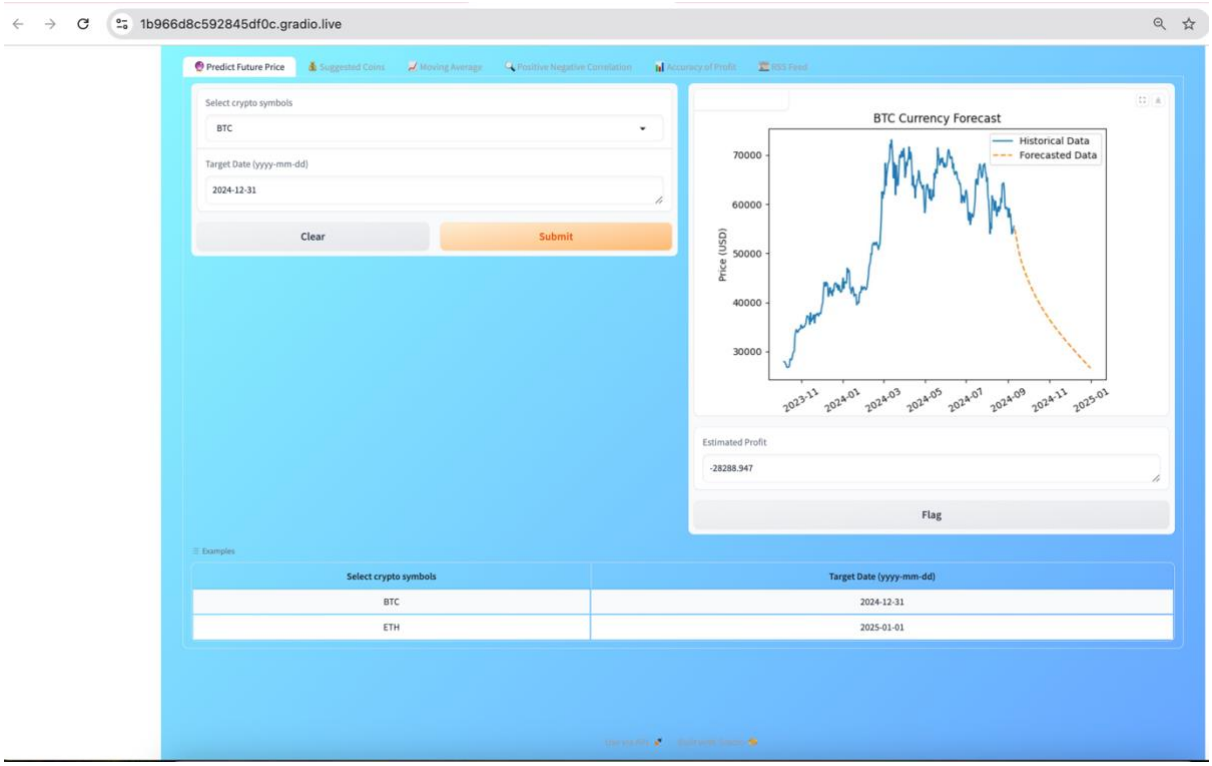


Figure 20:Future Price prediction GUI

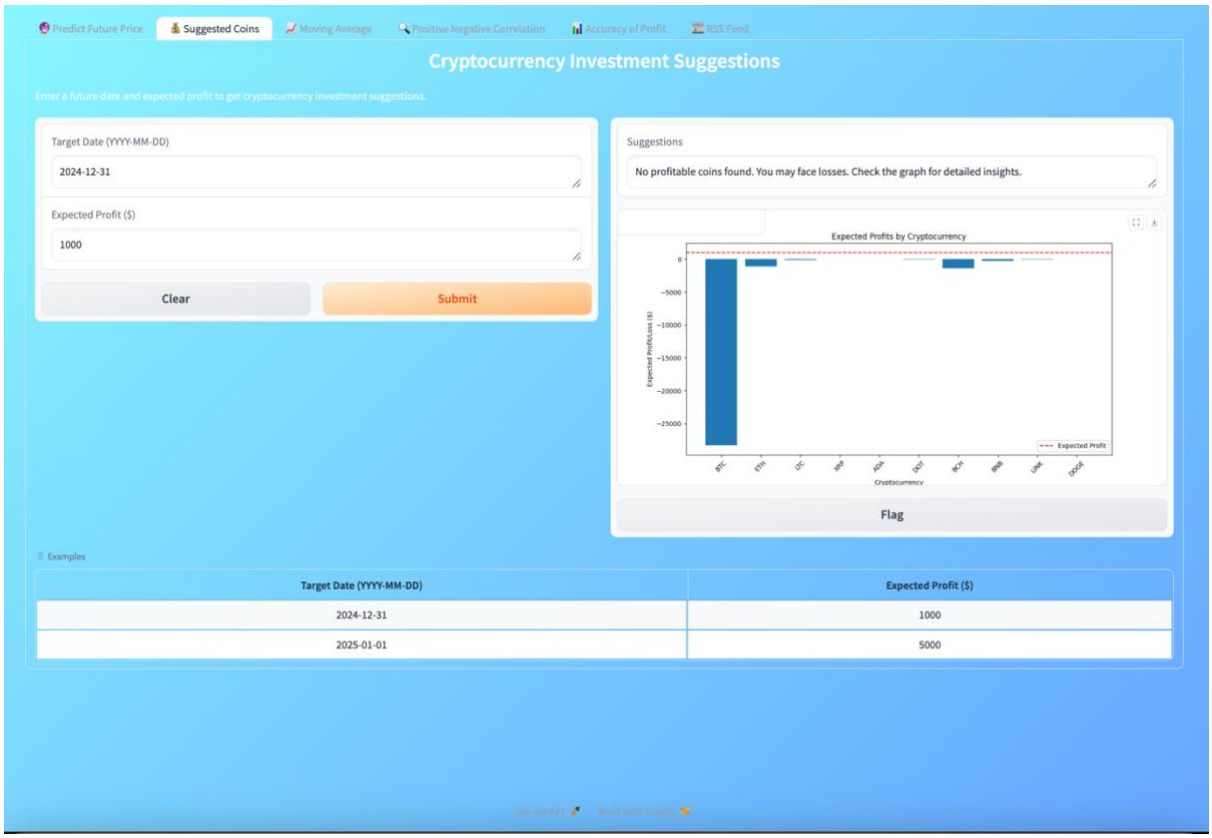


Figure 21: Suggested Coin GUI

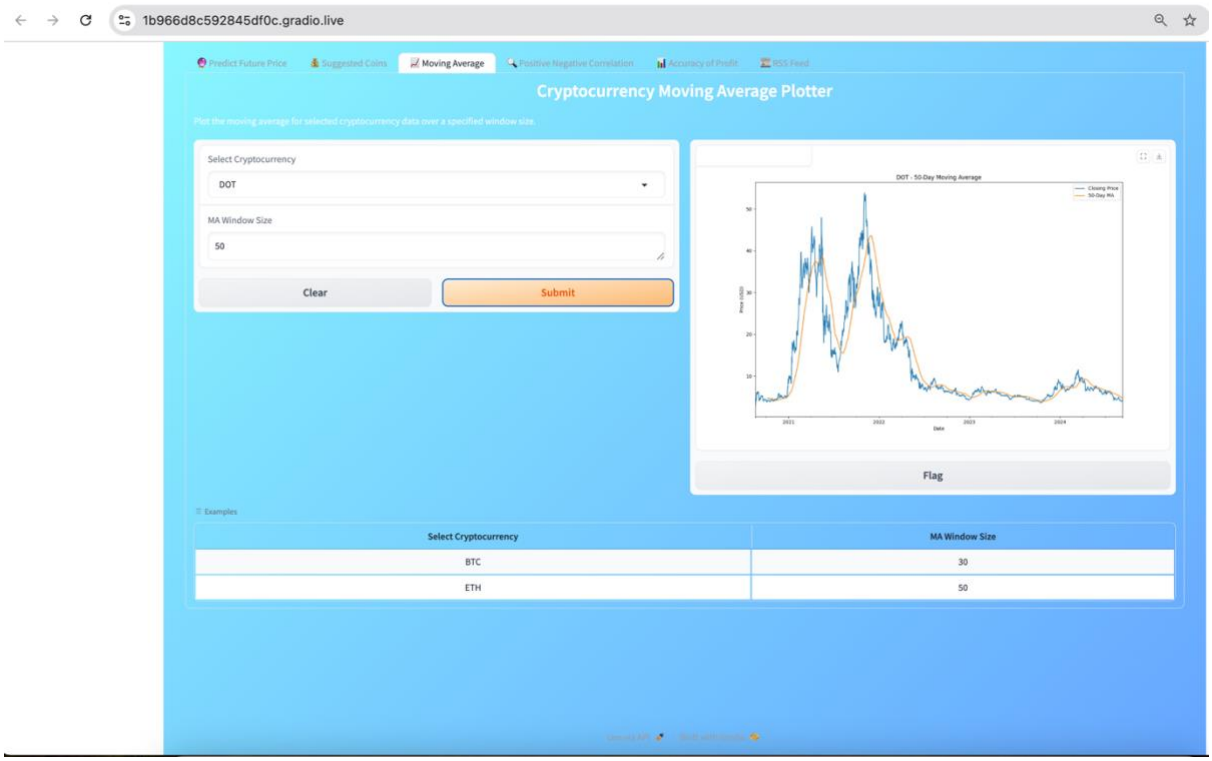


Figure 22: MA GUI

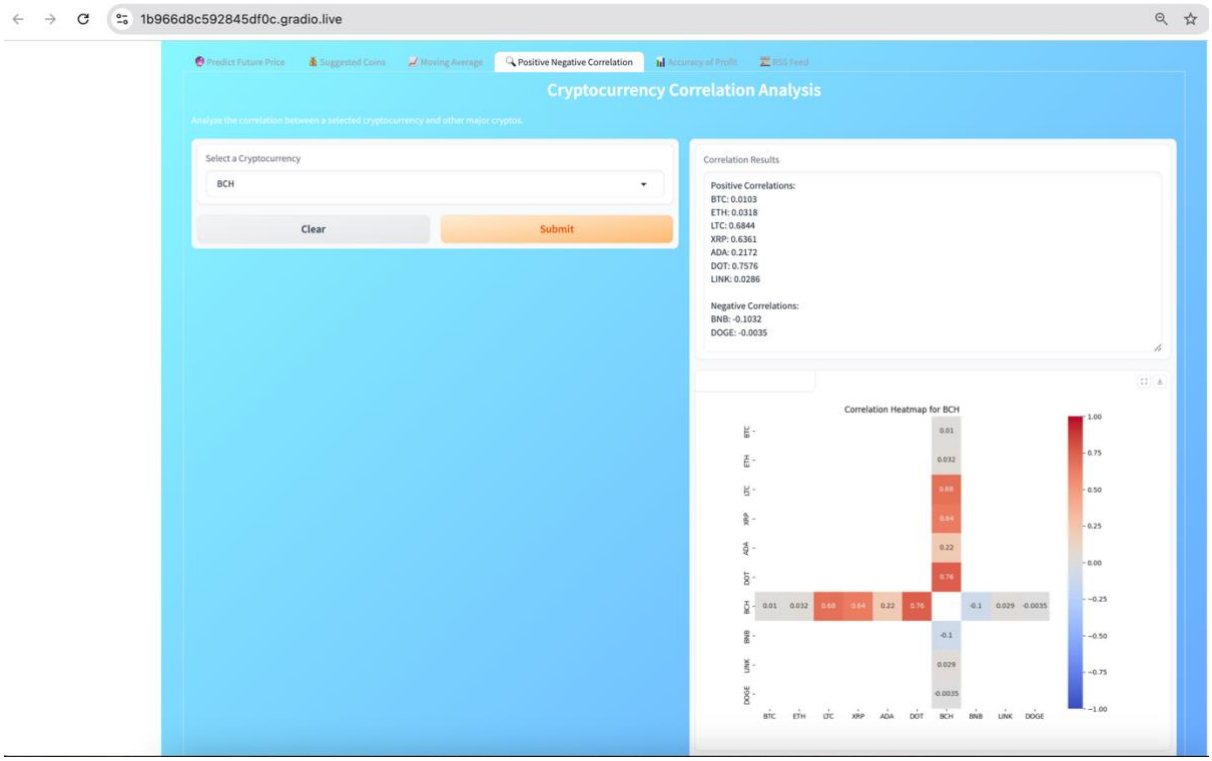


Figure 23:Correlation GUI

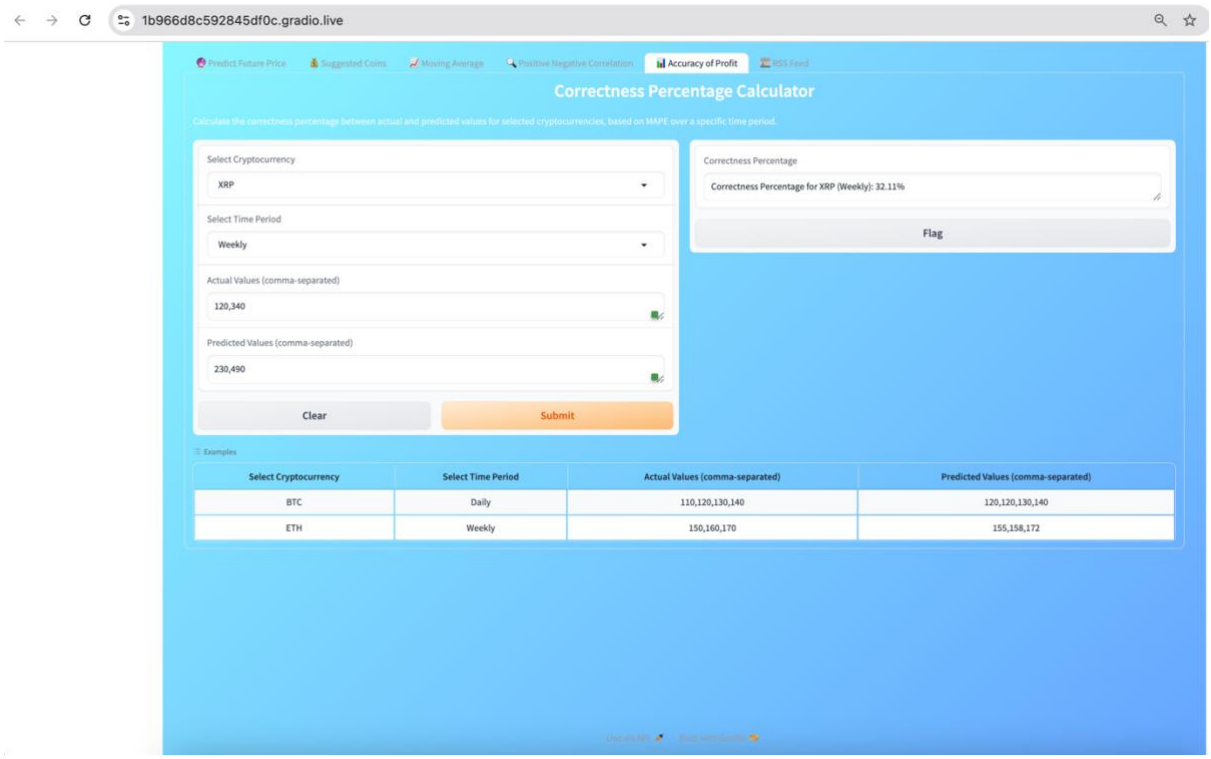


Figure 24: Accuracy Profit GUI

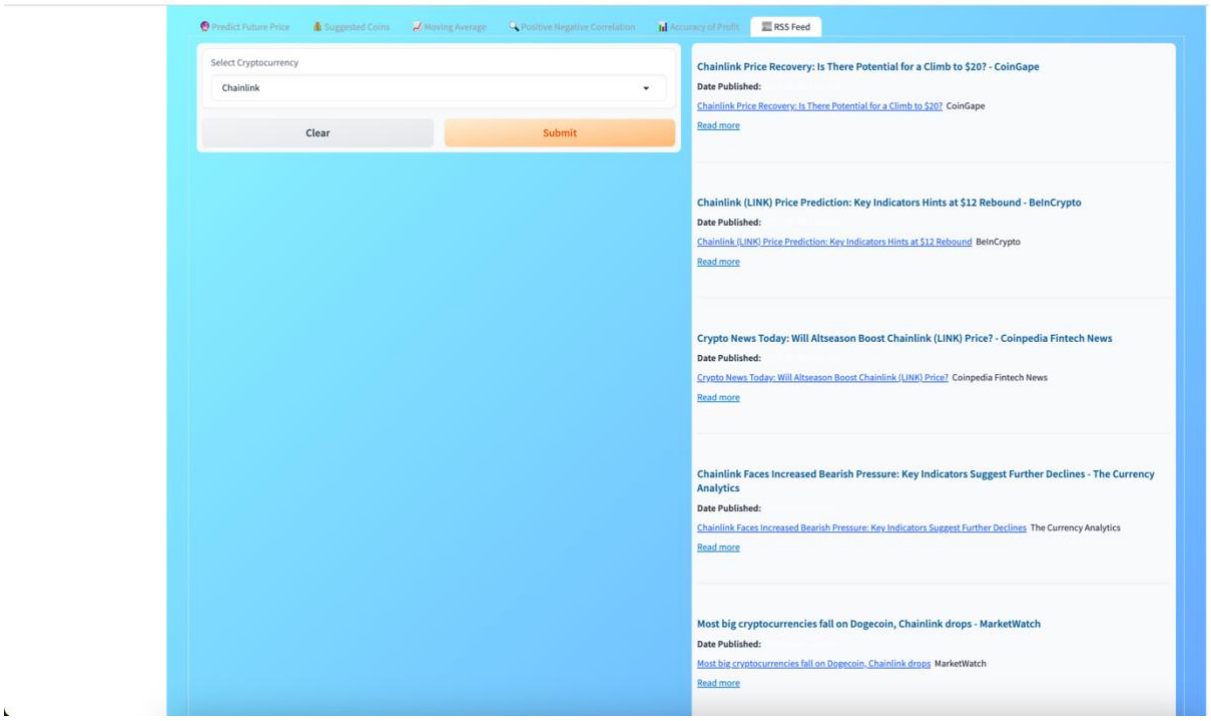


Figure 25:RSS feed GUI

10. References

- Bouri, E., Molnár, P., Azzi, G., Roubaud, D. and Hagfors, L.I., 2017. On the hedge and safe haven properties of Bitcoin: Is it really more than a diversifier? *Finance Research Letters*, 20, pp.192-198.

- Chen, Z., Siems, T.F. and Wang, L., 2020. Machine learning in cryptocurrency price prediction: Evidence from Bitcoin and Ethereum. *Journal of Risk and Financial Management*, 13(11), p.282.

- Nakamoto, S. and Iwasaki, T., 2019. Predictive modeling in cryptocurrency markets using machine learning: A comprehensive review. *Computational Economics*, 53(4), pp.1623-1643.