National College of Ireland

Configuration Manual

MSc Research Project
FinTech

Tze Yeng Chong
Student ID: X21231869

School of Computing
National College of Ireland

Supervisor: Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Tze Yeng Chong ……

Student ID: …… X21231869 ……..

Programme: …… MSc FinTech …… **Year:** …. 2022/23 ….

Module: …… MSc Research Project ……

Lecturer: …… Brian Byrne ……

Submission Due Date: …… 14.08.2023 ……

Project Title: …… Unravelling Cryptocurrency and Stock Market Dynamics: Predictive Models and Macroeconomic Implications ……

Word Count: …… 930 …… **Page Count:** …… 16 ……

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: …… *tzeyeng* ……

Date: …… 11.08.2023 ……

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

Tze Yeng Chong
X21231869

1 Introduction

This configuration manual provides detailed information on the system setup, software, and hardware specifications, as well as the steps involved in implementing the Research Project: "Unravelling Cryptocurrency and Stock Market Dynamics: Predictive Models and Macroeconomic Implications".

Google Colab is chosen as the development environment, the submitted CSV files and Jupyter notebooks can be viewed on the author's Github profile: https://github.com/NCI-tyc/ResearchProject.git

2 System Configuration

2.1 Hardware

- Model: Asus Vivobook, 500GB
- Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- RAM: 8 GB

2.2 Software

- Valid Gmail account for access to Google products, particularly Google Drive.
- Google Colab, a cloud based Jupyter notebook environment.

3 Project Development

This project utilises the Knowledge Discovery in Database (KDD) approach to accomplish its research objectives. KDD is employed to extract knowledge from the data. The process encompasses various stages, including data collection, preprocessing, transformation, data mining, pattern interpretation, and knowledge representation.

3.1 Google Colab

The Google Colab environment is utilised to conduct the experiments. To utilise Google Colab, it is necessary to have access to Google Drive, which in turn requires a valid Gmail account. Upon navigating to the provided URL, a code will be displayed, which can be utilised to gain authorisation for utilising Google Drive for the purpose of mounting files and data. Python 3 notebooks were utilised for all experiments.

3.2 Data Collection

- Step 1: Download the S&P Cryptocurrency Broad Digital Market Index data from the official site (https://www.spglobal.com/spdji/en/indices/digital-assets/sp-cryptocurrency-broad-digital-market-index/#overview). The maximum time span is selected and data is downloaded into an excel file, which is then converted into a CSV file.

```
# Load the S&P Cryptocurrency BDM Index data
Crypto = pd.read_csv('/content/PerformanceGraphExport2.csv')
```

- Step 2: Download the S&P500 and exchange rates data from Yahoo Finance using yfinance package on Python.

```
[3]  # Download S&P500 data using yfinance
     sp500_data = yf.download('^GSPC', start='2017-02-01', end='2023-05-01')
     dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')

     # Extract the 'Close' prices
     sp500_close = sp500_data['Close']
```

```
[*********************100%***********************]  1 of 1 completed
```

```
[4]  # Download the exchange rates data using yfinance
     tickers = ['GBPJPY=X', 'EURCAD=X', 'AUDCNY=X']
     FX_rates = yf.download(tickers, start='2017-02-01', end='2023-05-01')
     dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
```

```
[*********************100%***********************]  3 of 3 completed
```

```
[5]  # Extract the 'Close' prices
     GBP_JPY = FX_rates['Close']['GBPJPY=X']
     EUR_CAD = FX_rates['Close']['EURCAD=X']
     AUD_CNY = FX_rates['Close']['AUDCNY=X']
```

```
[6]  # Merge the S&P500 data with Exchange rates data based on the date index
     merged_data = pd.merge(sp500_close, GBP_JPY, left_index=True, right_index=True)
     merged_data = pd.merge(merged_data, EUR_CAD, left_index=True, right_index=True)
     merged_data = pd.merge(merged_data, AUD_CNY, left_index=True, right_index=True)
```

- Step 3: Download short-term interest rates and inflation data from FRED. The inflation data is upscaled from monthly to daily frequency.

```
[8]  Int_rate = pdr.data.DataReader('DTB3', 'fred', start='2017-02-28', end='2023-05-01')
     Int_rate.columns = ['Int_rates']
     Int_rate = Int_rate.dropna()
     Int_rate.tail()
```

```
[9]  Inflation = pdr.data.DataReader('CPALTT01USM657N', 'fred', start='2017-02-28', end='2023-05-01')
     Inflation.columns = ['Inflation']
     Inflation = Inflation.dropna()
     Inflation.tail()
```

```
[10] # Resampling the CPI from annually to daily frequency
     Inflation_d = Inflation.resample('D').interpolate(method='linear')
     Inflation_d.tail()
```

- Step 4: Similarly, the Economic Policy Uncertainty Index data is downloaded and saved into a CSV file from its official site (https://www.policyuncertainty.com/us_monthly.html). The monthly data is then upscaled to daily frequency.

```
# Load the Economic Policy Uncertainty (EPU) data
EPU = pd.read_csv('/content/US_Policy_Uncertainty_Data2.csv')
EPU['Date'] = pd.to_datetime(EPU['Date'])
EPU.set_index('Date', inplace=True)
EPU = EPU.dropna()

# Resampling the EPU index from monthly to daily frequency
EPU_d = EPU.resample('D').interpolate(method='linear')
EPU_d.tail
```

- Step 5: Merge all the variables into a single dataframe and save into a CSV file.

```
# Merge the new variables into our data frame
merged_data = pd.merge(merged_data, Int_rate, left_index=True, right_index=True, suffixes=('_merged', '_Int_rate'))
merged_data = pd.merge(merged_data, Inflation_d, left_index=True, right_index=True, suffixes=('_merged', '_Inflation_d'))
merged_data = pd.merge(merged_data, EPU_d, left_index=True, right_index=True, suffixes=('_merged', '_EPU_d'))
merged_data.tail()
```

	Close	GBPJPY=X	EURCAD=X	AUDCNY=X	S&P Cryptocurrency Broad Digital Market Index (USD)	Int_rates	Inflation	News_Based_Policy_Uncert_Index
2023-04-24	4137.040039	166.729996	1.48770	4.607537	2114.64	5.04	0.311124	195.666667
2023-04-25	4071.629883	167.584000	1.49634	4.616732	2124.01	4.97	0.302656	197.000000
2023-04-26	4055.989990	165.977005	1.49497	4.593824	2130.29	5.00	0.294187	198.333333
2023-04-27	4135.350098	166.410004	1.50546	4.573267	2249.02	5.03	0.285718	199.666667

```
# Save file to csv
merged_data.to_csv('merged_data_with_datetime.csv')
```

3.3 Diagnostics Checking

- Simple OLS regression:

```
# Perform a linear regression to explore the relationship
# Define the dependent variable (y) and independent variable(s) (X)
y = return_dataset['Crypto']
X = return_dataset.drop('Crypto', axis=1)

# Add a constant column to X for the intercept term
X = sm.add_constant(X)

# Fit the OLS regression model
model = sm.OLS(y, X)
results = model.fit()

# Get the residuals from the model
residuals = results.resid

# Print the regression results summary
print(results.summary())
```

- Check for Multicollinearity:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

X = return_dataset[['Crypto', 'SP500',  'GBPJPY', 'EURCAD', 'AUDCNY', 'Interest_Rates', 'CPI',  'EPU']]
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_data)
```

- Check for Autocorrelation:

```
import statsmodels.api as sm
import statsmodels.graphics.tsaplots as tsaplots

# Assuming 'residuals' is the residual series from the regression model
residuals = residuals.reset_index(drop=True)

# Plot autocorrelation of residuals
tsaplots.plot_acf(residuals, lags=20)

# Perform the Durbin-Watson test for autocorrelation
dw_test_result = sm.stats.durbin_watson(residuals)

print('Durbin-Watson Test Result:', dw_test_result)
```

- Check for Heteroscedasticity:

```python
from statsmodels.stats.diagnostic import het_breuschpagan

# Assuming 'residuals' is the residual series from our regression model
bp_test_results = het_breuschpagan(residuals, X)
print('Breusch-Pagan Test Results:')
print('Lagrange Multiplier Statistic:', bp_test_results[0])
print('LM-Test p-value:', bp_test_results[1])
print('F-Statistic:', bp_test_results[2])
print('F-Test p-value:', bp_test_results[3])
```

- Check for Normality:

```python
# Plot histogram
plt.hist(returns, bins=10, density=True)

# Create Q-Q plot
stats.probplot(returns, dist="norm", plot=plt)

# Perform Shapiro-Wilk test
shapiro_test_statistic, shapiro_p_value = stats.shapiro(returns)

# Perform Anderson-Darling test
anderson_test_statistic, anderson_critical_values, anderson_significance_levels = stats.anderson(returns)

# Perform Kolmogorov-Smirnov test
ks_test_statistic, ks_p_value = stats.kstest(returns, 'norm')

# Perform Jarque-Bera test
jarque_bera_test_statistic, jarque_bera_p_value = stats.jarque_bera(returns)
```

- Stationarity Check (Augmented Dicker-Fuller test):

```python
def adfuller_test(series, signif=0.05, name='', verbose=False):
    """Perform ADFuller to test for Stationarity of given series and print report"""
    r = adfuller(series, autolag='AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']
    def adjust(val, length= 6): return str(val).ljust(length)

    # Print Summary
    print(f'    Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'*47)
    print(f' Null Hypothesis: Data has unit root. Non-Stationary.')
    print(f' Significance Level    = {signif}')
    print(f' Test Statistic        = {output["test_statistic"]}')
    print(f' No. Lags Chosen       = {output["n_lags"]}')

    for key,val in r[4].items():
        print(f' Critical value {adjust(key)} = {round(val, 3)}')

    if p_value <= signif:
        print(f" => P-Value = {p_value}. Rejecting Null Hypothesis.")
        print(f" => Series is Stationary.")
    else:
        print(f" => P-Value = {p_value}. Weak evidence to reject the Null Hypothesis.")
        print(f" => Series is Non-Stationary.")
```

4 Modelling

In this project, the data underwent analysis with a variety of libraries apart from the aforementioned 'yfinance' package. The use of data processing libraries and machine learning methods for diverse data analysis purposes was undertaken. The absence of the library below may result in the program's failure to function.

```
!pip install statsmodels
!pip install pandas-datareader
!pip install --upgrade pandas-datareader
!pip install arch
!pip install pmdarima

# Importing libraries
import warnings
warnings.simplefilter('ignore')
import arch
from pmdarima import auto_arima
import math
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

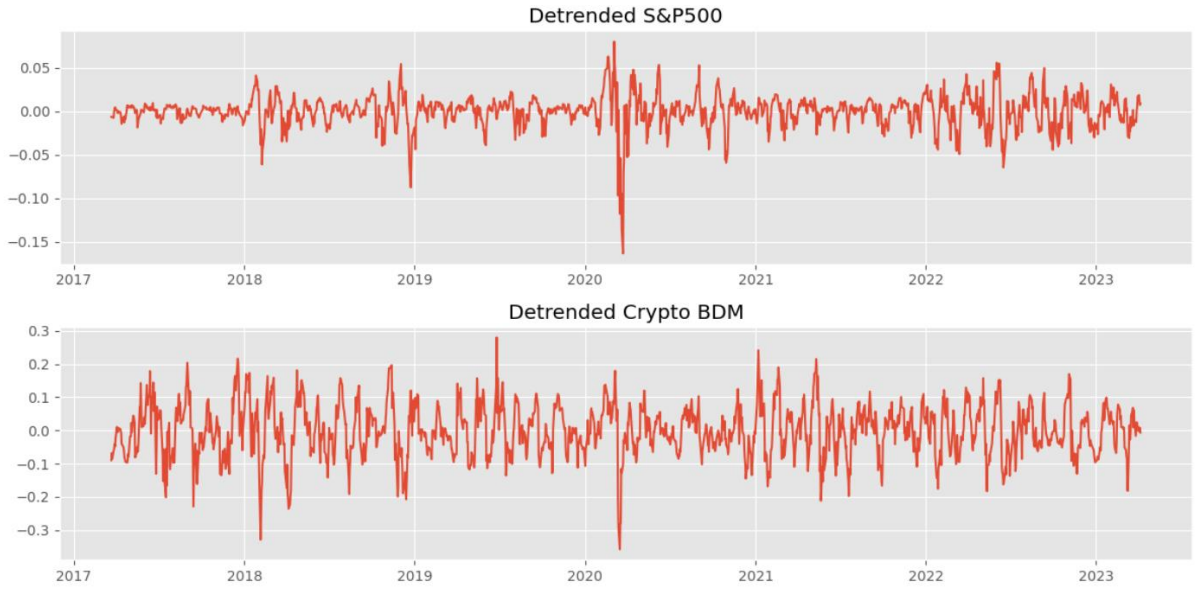
```
# Data Handling
import pandas as pd
import numpy as np

# Statistics
import statsmodels.api as sm
from statsmodels.compat import lzip
from statsmodels.api import OLS
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.graphics.tsaplots import acf as acf_func
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.vector_ar.vecm import coint_johansen
from statsmodels.tsa.vector_ar.vecm import VECM
from statsmodels.tsa.vector_ar import irf
from statsmodels.tsa.api import VAR
from statsmodels.tools.eval_measures import rmse, aic
from statsmodels.stats.stattools import durbin_watson

# Data Visualisation
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
```

4.1 Cointegration Tests

As the first objective of this project, a series of cointegration tests is used to examine the relationship of cryptocurrency and the stock market. We have employed the Johansen Cointegration test, cointegration rank, Engle-Granger test, Vector Error Correction Model (VECM) and compared the metrics to confirm the established connection between the markets.



- First, the 'Crypto' and 'SP500' are transformed into natural log and plotted against each other.

- A separate stationary test is conducted on the indices and detrended to be used in the cointegration tests, especially the VECM.

4.2 ARIMA Model

- Step 1: The data was split into testing (10%) and training (90%) sets.

```
#Split the crypto closing prices data into train and training set

train_data = Crypto_log.iloc[:int(len(Crypto_log) * 0.9)]
test_data = Crypto_log.iloc[int(len(Crypto_log) * 0.9):]
```

- Step 2: An auto ARIMA is then used to determine the optimal order of the ARIMA model.

```
# Fit Auto ARIMA model with specified parameters
model_autoARIMA = auto_arima(train_data, start_p=0, start_q=0,
                             max_p=3, max_q=3,  # maximum p and q
                             m=1,               # frequency of series
                             d=2,               # determine 'd' 2nd diff
                             seasonal=False,    # No Seasonality
                             start_P=0,
                             D=1,
                             trace=True,
                             error_action='ignore',
                             suppress_warnings=True,
                             stepwise=False)

# Print the model summary
print(model_autoARIMA.summary())

# Plot model diagnostics
model_autoARIMA.plot_diagnostics(figsize=(15, 8))
plt.show()
```

- Step 3: Once the best model is determined, we then fit the model and plot the fitted results.

```
model = sm.tsa.ARIMA(train_data, order=(0, 2, 1))
fitted = model.fit()
print(fitted.summary())
```

```
result = fitted.get_forecast(155, alpha =0.05).summary_frame()
print(result)
fc_series = pd.Series(result['mean'].values, index=test_data.index)
lower_series = pd.Series(result['mean_ci_lower'].values, index=test_data.index)
upper_series = pd.Series(result['mean_ci_upper'].values, index=test_data.index)
```

```
# Plot
plt.figure(figsize=(10,5), dpi=100)
plt.plot(train_data, label='training data')
plt.plot(test_data, color = 'blue', label='Actual Crypto Price')
plt.plot(fc_series, color = 'orange',label='Predicted Crypto Price')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.10)
plt.title('S&P Crypto BDM Price Prediction')
plt.xlabel('Time')
plt.ylabel('Crypto Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

- Step 4: We will then perform model evaluation on the selected metrics to compare the performance of each model.

```
mse = mean_squared_error(test_data, fc_series)
print('MSE: '+str(mse))
mae = mean_absolute_error(test_data, fc_series)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_data, fc_series))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(fc_series - test_data)/np.abs(test_data))
print('MAPE: '+str(mape))
r2 = r2_score(test_data, fc_series)
print('R2: '+str(r2))
```

4.3 Vector Auto-Regressive (VAR) model

- Step 1: Test and train split.

```
nobs = 15
df_train, df_test = return_dataset[0:-nobs], return_dataset[-nobs:]

# Check size
print(df_train.shape)
print(df_test.shape)
```

- Step 2: Build the VAR model and determine the lowest AIC. After the preferred AIC is selected, we will then fit the model.

```
model = VAR(return_dataset)
x = model.select_order(maxlags=12)
x.summary()
```

```
model_fitted = model.fit(8)
model_fitted.summary()
```

- Step 3: The selected VAR model will then be used to forecast with the training dataset.

```
# Get the lag order
lag_order = model_fitted.k_ar

# Input data for forecasting
forecast_input = df_train.values[-lag_order:]
```

```
# Forecast
fc = model_fitted.forecast(y=forecast_input, steps=nobs)
df_forecast = pd.DataFrame(fc, index=dataset.index[-nobs:], columns=dataset.columns
df_forecast.tail()
```

- Step 4: Final prediction on the actual dataset is conducted. Here, we try to predict 10 days with the fitted model. We are now using the whole dataset to feed the model with the predetermined lags.

```
# Make final predictions
model = VAR(return_dataset)
```

```
# Select 10 steps (days) of prediction
steps = 10

model_fit = model.fit(8)

pred = model_fit.forecast(return_dataset.values[-model_fit.k_ar:], steps=steps)
```

```
pred_df = pd.DataFrame(pred,columns=['Crypto',
              'SP500',
              'GBPJPY',
              'EURCAD',
              'AUDCNY',
              'Interes_Rates',
              'CPI',
              'EPU'])
pred_df.index = pd.date_range(start=return_dataset.index[-1], periods=steps+1)[1:]
```

```
# Make predictions on the test dataset
forecast = model_fitted.forecast(df_test.values, len(df_test))

# Convert the forecasted values to a DataFrame
forecast_df = pd.DataFrame(forecast, index=df_test.index, columns=df_test.columns)
```

```
df = return_dataset.tail(100).copy()
fc = pred_df.copy()
dp = 'Crypto'
```

- Step 5: Plot the forecasted values and observe the trends.

```
plt.figure(figsize=(12,4))
plt.plot(df.index, df[dp], label='actual',linestyle='dashed', marker='.')
plt.plot(fc.index, fc[[dp]], label='forecast',linestyle='dashed', marker='.')
plt.title(f"{dp} Forecast")
# plt.ylim(0.0,0.9)
plt.legend();
```

- Step 6: Model evaluation is performed.

```
# Calculate the Mean Absolute Error (MAE)
mae = np.mean(np.abs(forecast_df - df_test))

# Calculate the Mean Squared Error (MSE)
mse = np.mean((forecast_df - df_test) ** 2)

# Calculate the Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Calculate the Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs(forecast_df - df_test)/np.abs(df_test))

# Calculate forecast errors
forecast_errors = forecast_df - df_test

# Calculate forecast bias
forecast_bias = forecast_errors.mean()

# Calculate forecast horizon
forecast_horizon = forecast_df.index[-1] - forecast_df.index[0]
```

```
# Calculate the total variance of the target variable (Crypto
total_variance = np.var(df_test['Crypto'])

# Calculate the residual sum of squares (RSS)
residuals = forecast_df - df_test
rss = np.sum(residuals ** 2)

# Calculate the R-squared (variance explained)
r_squared = 1 - (rss / total_variance)

# Print the R-squared value
print("\nR-squared (Variance Explained):\n", r_squared)
```

4.4 LSTM Model

- Step 1: Scaling the 'Crypto_returns' data using a MinMaxScaler transforms the data values into a range between 0 and 1. This scaling ensures that the LSTM model processes the data with improved stability and quicker convergence.

```
from sklearn.preprocessing import MinMaxScaler
# rescale the whole dataset (ideally this shoud be done seperately on the train
scaler = MinMaxScaler(feature_range=(0, 1))
# train_series = scaler.fit_transform(train_series)
return_dataset['Crypto'] = scaler.fit_transform(return_dataset[['Crypto']])
scaled_data = return_dataset['Crypto']
```

```
from sklearn.preprocessing import MinMaxScaler

# Convert 'Crypto_returns' to a 2D array with a single feature
Crypto_returns_2d = Crypto_returns.values.reshape(-1, 1)

# Initialize MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit and transform the data
scaled_data = scaler.fit_transform(Crypto_returns_2d)

# Convert back to a pandas Series (if needed)
scaled_series = pd.Series(scaled_data.flatten(), index=Crypto_returns.index)
```

- Step 2: Train and test split.

```
# Assign train data
X_train = train_data.index
y_train = train_data

# Assign test data
X_test = test_data.index
y_test = test_data

n_features = 1

train_series = y_train.values.reshape((len(y_train), n_features))
test_series = y_test.values.reshape((len(y_test), n_features))
```

- Step 3: Building the LSTM model. Here, we set the look-back window into 10 days. The 'TimeseriesGenerator' creates sequences of data with the specified look-back window and batch size, enabling the model to learn temporal patterns in the data.

```
%%time

from keras.preprocessing.sequence import TimeseriesGenerator

look_back = 10

train_generator = TimeseriesGenerator(train_series, train_series,
                                      length        = look_back,
                                      sampling_rate = 1,
                                      stride        = 1,
                                      batch_size    = 10)

test_generator = TimeseriesGenerator(test_series, test_series,
                                      length        = look_back,
                                      sampling_rate = 1,
                                      stride        = 1,
                                      batch_size    = 10)

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

n_neurons  = 4
model = Sequential()
model.add(LSTM(n_neurons, input_shape=(look_back, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse');

model.fit(train_generator,epochs=300, verbose=0);
```

- Step 4: Plot the test predictions.

```
test_predictions  = model.predict(test_generator)
```

```
15/15 [==============================] - 1s 2ms/step
```

```
plt.figure(figsize=(24,8))
plt.plot(test_data.index, test_data, c='orange',label='true values')
plt.plot(X_test[10:],test_predictions, lw=3, c='r',linestyle = '-', label='predict
plt.legend(loc="lower left")
plt.xlabel("date", fontsize=10)
plt.ylabel("Close price (USD) (rescaled)", fontsize=16)
plt.title("Crypto Closing value", fontsize=16);
```

- Step 5: Perform model evaluation.

```
# Calculate Mean Squared Error (MSE)
mse = np.mean((test_predictions - test_series[look_back:]) ** 2)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Calculate Mean Absolute Error (MAE)
mae = np.mean(np.abs(test_predictions - test_series[look_back:]))

# Calculate Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs(test_predictions - test_series[look_back:])/np.abs(test_serie

# Flatten the test_predictions and test_series arrays
test_predictions_flattened = test_predictions.flatten()
test_series_flattened = test_series[look_back:].flatten()

# Calculate R-squared (R2)
r2 = r2_score(test_series_flattened, test_predictions_flattened)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape)
print("R-squared (R2):", r2)
```

4.5 Naïve Model

- Step 1: Create a naïve forecast based on the same training and testing data we used for the LSTM model and convert it into numeric values for evaluation.

```
# create a naïve forecast
test_data['naive'] = test_data.shift(1)
```

```
# Convert the Crypto prices to numeric
test_data = pd.to_numeric(test_data, errors='coerce')

# Remove rows with NaN values
test_data = test_data.dropna()
```

- Step 2: Generate the naïve forecast that is based on 1 lag.

```
# Extract the 'Crypto data as a numpy array
actual_prices = test_data.values

# Create a naive model by shifting the prices one step forward
naive_predictions = np.roll(actual_prices, -1)
```

- Step 3: Perform model evaluation.

```
# Calculate the metrics
naive_mse = mean_squared_error(actual_prices[:-1], naive_predictions[:-1])
naive_rmse = np.sqrt(naive_mse)
naive_mae = mean_absolute_error(actual_prices[:-1], naive_predictions[:-1])
naive_mape = np.mean(np.abs((actual_prices[:-1] - naive_predictions[:-1]) / actual_
naive_r2 = r2_score(actual_prices[:-1], naive_predictions[:-1])
```

4.6 Random Forest Regression Model

The random forest model was used to determine the significant macroeconomic variables.

- Step 1: We first define the variables considered in the features importance analysis.

```
features = ['SP500', 'GBPJPY', 'EURCAD', 'AUDCNY', 'Interest_Rates', 'CPI', 'EPU']
X = df[features]
y = df['Crypto']
```

- Step 2: Train and test split.

```
from sklearn.model_selection import train_test_split
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
```

- Step 3: Building the model by setting the random state to 1, then we will feed the training data to the random forest algorithm for it to map patterns and decision trees.

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(random_state=1)
```

```
rf_model.fit(train_X, train_y)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=1)
```

- Step 4: Getting the features importances. The importances are set into a range between 0 and 1, with closer to 0 meaning less significant and closer to 1 meaning more significant.

```
# Get feature importances
importances = rf_model.feature_importances_

# Print feature importances
for i, importance in enumerate(importances):
    print('Feature', i+1, ':', df.columns[i+1], 'Importance:', importance)
```

References

Aptech. (2020). *A Guide to Conducting Cointegration Tests*. Available at: https://www.aptech.com/blog/a-guide-to-conducting-cointegration-tests/ [Accessed 10 August 2023].

Brooks, C. (2019). *Introductory Econometrics for Finance.* 4th edn. Cambridge University Press. doi: 10.1017/9781108524872.

Hyndman, R. J., and Athanasopoulos, G. (2021). *Forecasting: principles and practice*. 3rd edn. OTexts. Available at: https://otexts.com/fpp3/ [Accessed 10 August 2023].

Kaggle. (2020). *LSTM time series prediction: sine wave example*. Available at: https://www.kaggle.com/code/carlmcbrideellis/lstm-time-series-prediction-sine-wave-example [Accessed 10 August 2023].

Keras. (2023). *Time series data loading*. Available at: https://keras.io/api/data_loading/timeseries/ [Accessed 10 August 2023].