National College of Ireland

Configuration Manual

MSc Research Project
Master of Science in Fintech

John Guevara Rincón
Student ID: 23409517

School of Computing
National College of Ireland

Supervisor: Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	John Fredy Guevara Rincón
Student ID:	23409517
Programme:	MSc in Fintech **Year:** 2025
Module:	Msc Research Project
Supervisor:	Brian Byrne
Submission Due Date:	11th August 2025
Project Title:	Integrating Deep Learning and Statistical Methods for Cryptocurrency Price Forecasting: The case of Ripple
Word Count:	681 **Page Count** 10

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: John Guevara Rincón

Date: 11th August 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

John Guevara Rincon
23409517

1 Introduction

This study aims to analize and predict the closing price of the cryptocurrency XRP using two predictive models: a deep learning model (LSTM) and a statistical model (ARIMA). The configuration manual shows a set of instructions to replicate the research, from setting up the system to running the code and reproducing the evaluation results.

2 System Configuration

2.1 Hardware Specification

The analysis of this research was executed on Google Collaboratory; However, the runtime was changed to run with the local resources.

- **Processor:** Apple M4 Chip
- **RAM:** 16 GB
- **Disk:** 256 GB
- **GPU:** 8-core

2.2 Software Specification

- Google Account (required for accessing Google Drive and Google Collab)
- Google Collaboratory environment
- Python Version: 3.11or higher
- Key Python Libraries:
  - pandas 2.2.2
  - NumPy 2.0.2
  - matplotlib 3.10.0
  - seaborn
  - yfinance
  - scikit-learn
  - keras / tensorflow
  - statsmodels
  - mplfinance

3 Install Packages and Libraries

Before running the code, ensure the following packages are installed in Google Colab:

- Packages

```
[172] pip install mplfinance
```

```
Requirement already satisfied: mplfinance in /usr/local/lib/python3.11/dist-packages (0.12.10b0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from mplfinance) (3.10.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from mplfinance) (2.2.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->mplfinance) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->mplfinance) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->mplfinance) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->mplfinance) (1.17.0)
```

Figure 1 : mplfinance package

- Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
import itertools
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

import warnings
```

Figure 2 : Python Libraries

4 Data Gathering

4.1 Yahoo Finance API

The historical cryptocurrency data for XRP-USD, BTC-USD, and ETH-USD was downloaded directly using the yfinance library.

```
# For reading crypto data from yahoo
import yfinance as yf

# The cryptocurrencies I'll use for this analysis
# Using Yahoo Finance tickers for crypto (USD pairs)
crypto_list = ['XRP-USD', 'BTC-USD', 'ETH-USD']

# Set up End and Start times
end = datetime.now()
start = datetime(end.year - 3, end.month, end.day)

# Download crypto data
for crypto in crypto_list:
    globals()[crypto.replace('-USD', '')] = yf.download(crypto, start, end)
```

Figure 3 : Gathering data – Yahoo Finance API

5 Data Preparation

5.1 Data Cleaning

- Flattened multi-index column structure from Yahoo Finance

```
# Create list of dataframes and corresponding names
company_list = [XRP, BTC, ETH]
company_name = ["RIPPLE", "BITCOIN", "ETHEREUM"]

# Add company name column to each dataframe
for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

# Concatenate all dataframes
df = pd.concat(company_list, axis=0)
print("Last 10 rows of combined cryptocurrency data:")
print(df.tail(10))
```

Figure 4 : Dataframe preparation

- Filtered to ensure consistent range

```
# Grab all the closing prices for the crypto list into one DataFrame
closing_df = yf.download(crypto_list, start=start, end=end)['Close']

# Make a new crypto returns DataFrame
crypto_rets = closing_df.pct_change()
print("Cryptocurrency Returns DataFrame:")
print(crypto_rets.head())

# Display basic statistics
print("\nBasic Statistics for Cryptocurrency Returns:")
print(crypto_rets.describe())
```

Figure 5: Normalizing dataframe

5.2 Feature Engineering

- Calculated moving averages (10, 20, 50 days)

```
# Calculate moving averages for all cryptocurrencies
ma_day = [10, 20, 50]

for ma in ma_day:
    for company in company_list:
        column_name = f"MA for {ma} days"
        company[column_name] = company['Close'].rolling(ma).mean()

# Create subplots for moving averages visualization
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# Plot XRP with moving averages
XRP[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0])
axes[0].set_title('RIPPLE (XRP-USD)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Price (USD)')
axes[0].set_xlabel('Date')
axes[0].grid(True, alpha=0.3)
axes[0].legend(loc='upper left')

# Plot BTC with moving averages
BTC[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1])
axes[1].set_title('BITCOIN (BTC-USD)', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Price (USD)')
axes[1].set_xlabel('Date')
axes[1].grid(True, alpha=0.3)
axes[1].legend(loc='upper left')

# Plot ETH with moving averages
ETH[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[2])
axes[2].set_title('ETHEREUM (ETH-USD)', fontsize=14, fontweight='bold')
axes[2].set_ylabel('Price (USD)')
axes[2].set_xlabel('Date')
axes[2].grid(True, alpha=0.3)
axes[2].legend(loc='upper left')

# Rotate x-axis labels for better readability
for ax in axes:
    ax.tick_params(axis='x', rotation=45)

plt.suptitle('Cryptocurrency Moving Averages Analysis', fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()

# Display the latest moving average values
print("\nLatest Moving Average Values:")
```

Figure 6: Moving Average

- Calculated daily returns

```
# Find the percent change for each day
for company in company_list:
    company['Daily Return'] = company['Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

XRP['Daily Return'].plot(ax=axes[0], legend=True, linestyle='--', marker='o')
axes[0].set_title('XRP')

BTC['Daily Return'].plot(ax=axes[1], legend=True, linestyle='--', marker='o')
axes[1].set_title('BITCOIN')

ETH['Daily Return'].plot(ax=axes[2], legend=True, linestyle='--', marker='o')
axes[2].set_title('ETHEREUM')

fig.tight_layout()
```

Figure 7: Daily Returns (XRP, BTC,ETH)

- Created candlestick charts and volume plots for exploratory data analysis (EDA)

```
import mplfinance as mpf

# Candle stick bar chart
xrp_data = XRP.copy()

# Check if XRP has multi-level columns (common with yfinance when downloading multiple tickers)
if isinstance(xrp_data.columns, pd.MultiIndex):
    # Flatten the multi-level columns
    xrp_data.columns = xrp_data.columns.get_level_values(0)

# Remove the company_name column if it exists (from your code)
if 'company_name' in xrp_data.columns:
    xrp_data = xrp_data.drop('company_name', axis=1)
```

Figure 8: Candlestick bar chart

5.3 Scaling and Reshaping for LSTM

- Applied MinMaxScaler to normalize the closing prices between 0 and 1

```
print("\n=== Data Preparation ===")

# Calculate split indices
train_size = int(len(data) * 0.70)  # 70% for training
val_size = int(len(data) * 0.15)    # 15% for validation
test_size = len(data) - train_size - val_size  # 15% for testing

print(f"Train size: {train_size}")
print(f"Validation size: {val_size}")
print(f"Test size: {test_size}")

# IMPORTANT: Scale data AFTER splitting to avoid data leakage
train_data = data[:train_size]
val_data = data[train_size:train_size + val_size]
test_data = data[train_size + val_size:]

# Fit scaler only on training data
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data)
val_scaled = scaler.transform(val_data)
test_scaled = scaler.transform(test_data)
```

Figure 9: Scaler the data – Min Max

- Reshaped data into 3D arrays for LSTM with a 30-day lookback window

```
def create_sequences(data, lookback=30):
    """Create sequences for LSTM"""
    X, y = [], []
    for i in range(lookback, len(data)):
        X.append(data[i-lookback:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

lookback = 30

# Create sequences for each set
X_train, y_train = create_sequences(train_scaled, lookback)
X_val, y_val = create_sequences(val_scaled, lookback)

# For test set, we need to include some data from validation set for context
combined_val_test = np.concatenate([val_scaled[-lookback:], test_scaled])
X_test, y_test = create_sequences(combined_val_test, lookback)

# Reshape for LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print(f"\nSequence shapes:")
print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"X_val: {X_val.shape}, y_val: {y_val.shape}")
print(f"X_test: {X_test.shape}, y_test: {y_test.shape}")
```

Figure 10: Reshape data (30 days)

6 Model Implementation

6.1 LSTM Model

- Architecture

```
print("\n=== Building Optimized LSTM Model ===")

# Clear any existing models
from tensorflow.keras import backend as K
K.clear_session()

model = Sequential([
    # First LSTM layer
    LSTM(50, return_sequences=True, input_shape=(lookback, 1)),
    Dropout(0.2),

    # Second LSTM layer
    LSTM(50, return_sequences=True),
    Dropout(0.2),

    # Third LSTM layer
    LSTM(50),
    Dropout(0.2),

    # Dense output layer
    Dense(1)
])

# Compile with appropriate learning rate
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

print(model.summary())
```

Figure 11: LSTM architecture

- Optimizer: Adam (learning rate 0.001) , Callbacks: EarlyStopping and ReduceLROnPlateau

```
print("\n=== Training Model ===")

# Callbacks
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=5,
    min_lr=0.00001,
    verbose=1
)

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=42,
    callbacks=[early_stop, reduce_lr],
    verbose=1,
    shuffle=False  # Important for time series
)
```

Figure 12: ADAM optimizer

6.2 ARIMA Model

- Seasonal and non-seasonal parameters optimized using grid search

```
print("\n=== Enhanced Time Series Analysis ===")

def check_stationarity(timeseries, title):
    """Perform Augmented Dickey-Fuller test"""
    result = adfuller(timeseries.dropna())
    print(f'\n{title}')
    print(f'ADF Statistic: {result[0]:.6f}')
    print(f'p-value: {result[1]:.6f}')
    print(f'Critical Values:')
    for key, value in result[4].items():
        print(f'\t{key}: {value:.3f}')

    is_stationary = result[1] <= 0.05
    if is_stationary:
        print("=> Series is stationary")
    else:
        print("=> Series is non-stationary")
    return is_stationary

# Check stationarity on Close prices
is_stationary = check_stationarity(train_data['Close'], 'Original Series Stationarity Test')

# Apply log transformation to stabilize variance
train_data['Close_log'] = np.log(train_data['Close'] + 1e-10)
```

Figure 13: ARIMA - Stationary

- Fitted on historical closing price of XRP

```
print("\n=== Grid Search for Optimal ARIMA Parameters ===")

# More focused parameter ranges based on ACF/PACF analysis
p_values = range(0, 5)
d_values = [optimal_d]  # Use the optimal d found above
q_values = range(0, 5)

# Store results
results = []
best_aic = float('inf')
best_params = None

# Grid search with information criteria
for p in p_values:
    for d in d_values:
        for q in q_values:
            if p == 0 and q == 0:
                continue

            try:
                # Use log-transformed data
                model = ARIMA(train_data['Close_log'], order=(p, d, q))
                model_fit = model.fit()

                results.append({
                    'params': (p, d, q),
                    'aic': model_fit.aic,
                    'bic': model_fit.bic,
                    'hqic': model_fit.hqic
                })

                if model_fit.aic < best_aic:
                    best_aic = model_fit.aic
                    best_params = (p, d, q)

            except Exception as e:
                continue

print(f"\nBest ARIMA model: ARIMA{best_params} with AIC: {best_aic:.2f}")

# Show top 5 models
results_df = pd.DataFrame(results).sort_values('aic').head(5)
print("\nTop 5 models by AIC:")
print(results_df)
```

Figure 14: ARIMA Parameters

7 Model Evaluation

7.1 Metrics

```python
# Calculate metrics
rmse = np.sqrt(mean_squared_error(y_test, predictions))
mae = mean_absolute_error(y_test, predictions)
mape = np.mean(np.abs((y_test - predictions) / y_test)) * 100
r2 = r2_score(y_test, predictions)

print(f"RMSE: ${rmse:.4f}")
print(f"MAE: ${mae:.4f}")
print(f"MAPE: {mape:.2f}%")
print(f"R² Score: {r2:.4f}")
```

Figure 15: Metrics (RMSE, MAE, MAPE, R2)

8 Plotting and Visualization

```python
# Plot training data (blue line)
plt.plot(train_val_dates, train_val_data,
         label='Training Data',
         color='#1f77b4',  # Blue
         linewidth=1.5)

# Plot actual test data (green line)
plt.plot(test_dates, y_test_plot,
         label='Actual Price',
         color='#2ca02c',  # Green
         linewidth=1.5)

# Plot LSTM predictions (orange line)
plt.plot(test_dates, y_pred_plot,
         label='LSTM Predictions',
         color='#ff7f0e',  # Orange
         linewidth=1.5,
         alpha=0.8)

# Add vertical line to separate train/test
plt.axvline(x=df_xrp.index[train_val_size],
            color='gray',
            linestyle='--',
            alpha=0.5,
            linewidth=1)

# Formatting
plt.title('LSTM Model - XRP Price Prediction (Rolling Forecast)',
          fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('Close Price USD ($)', fontsize=12)
plt.legend(loc='upper left', fontsize=11)
plt.grid(True, alpha=0.3)
plt.ylim(0, 3.5)  # Set y-axis limits similar to your ARIMA plot

# Format x-axis dates
plt.gcf().autofmt_xdate()

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

Figure 16: LSTM Predictions Plot

```
plt.figure(figsize=(28, 12))
plt.title('ARIMA Model - XRP Price Prediction (Rolling Forecast)', fontsize=18)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Close Price USD ($)', fontsize=14)
plt.plot(train.index, train['Close'], label='Training Data', linewidth=2)
plt.plot(valid.index, valid['Close'], label='Actual Price', linewidth=2)
plt.plot(valid.index, valid['Predictions'], label='ARIMA Predictions', linewidth=2, alpha=0.8)
plt.legend(loc='upper left', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()
```

Figure 17: ARIMA Predictions Plot

```
# Plot 1: Main prediction plot with ARIMA confidence intervals
ax1 = axes[0]
ax1.plot(recent_historical.index, recent_historical.values, 'b-',
         label='Historical Price', linewidth=2)
ax1.plot(future_df.index, future_df['Predicted_Price'], 'r-',
         label='ARIMA Prediction', linewidth=3)
ax1.fill_between(future_df.index, future_df['Lower_Bound_95'], future_df['Upper_Bound_95'],
                 alpha=0.3, color='red', label='95% ARIMA Confidence Interval')
ax1.axvline(x=full_data.index[-1], color='gray', linestyle=':', alpha=0.7,
            label='Prediction Start')
ax1.set_title('XRP 30-Day ARIMA Price Prediction with Model Confidence Intervals', fontsize=16)
ax1.set_xlabel('Date', fontsize=14)
ax1.set_ylabel('Price (USD)', fontsize=14)
ax1.legend(loc='upper left', fontsize=12)
ax1.grid(True, alpha=0.3)
ax1.tick_params(axis='x', rotation=45)

# Plot 2: Comparison of confidence intervals
ax2 = axes[1]
ax2.plot(recent_historical.index, recent_historical.values, 'b-',
         label='Historical Price', linewidth=2)
ax2.plot(future_df.index, future_df['Predicted_Price'], 'r-',
         label='ARIMA Prediction', linewidth=3)
ax2.fill_between(future_df.index, future_df['Volatility_Lower'], future_df['Volatility_Upper'],
                 alpha=0.2, color='green', label='95% Volatility-Based CI')
ax2.fill_between(future_df.index, future_df['Lower_Bound_95'], future_df['Upper_Bound_95'],
                 alpha=0.3, color='red', label='95% ARIMA CI')
ax2.axvline(x=full_data.index[-1], color='gray', linestyle=':', alpha=0.7)
ax2.set_title('Comparison of Confidence Interval Methods', fontsize=16)
ax2.set_xlabel('Date', fontsize=14)
ax2.set_ylabel('Price (USD)', fontsize=14)
ax2.legend(loc='upper left', fontsize=12)
ax2.grid(True, alpha=0.3)
ax2.tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

Figure 18: Comparison Performance Models Plot