National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Naresh Kumar Satish
Student ID: 23248441

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Naresh Kumar Satish
Student ID:	23248441
Programme:	Artificial Intelligence
Year:	2024-25
Module:	MSc Research Project
Supervisor:	Anderson Simiscuka
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	340
Page Count:	7

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:	Naresh Kumar Satish
Date:	25th 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

Naresh Kumar Satish
23248441

1 Introduction

An outline about the system requirements, configurations, tools used, step by step implementation, assumptions made, and the required dependencies are described in a systematic flow of the research work. The execution of the python scripts, environment utilized, and the packages used are well described in the below sections. The manual gives an overall visualization on the datasets used, data preprocessing stages, feature engineering techniques, model training and evaluation.

2 System Configurations and Setup

2.1 Hardware Requirements

The processor and the memory utilized was 12th Generation with 10 cores, 12 Logical Processors, 1300Mhz, 8GB RAM.

2.2 Software Environment

The research work was scripted using as the Python primary language in its latest version - 3.12.4, and JupyterLab is the primary IDE created from Anaconda distribution running on Windows 11 environment with 64-bit system type. Conda was used for the management of the libraries and packages and JupyterLab was accessed using Google Chrome browser.
The required packages and dependencies are shown in the below figure 1

```
import requests
from datetime import datetime, timedelta
import pandas as pd
import time
import praw
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
import numpy as np
from transformers import BertTokenizer, BertModel
import torch
from sentence_transformers import SentenceTransformer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import xgboost as xgb
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from xgboost import XGBRegressor
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Dropout
from tqdm import tqdm
from tensorflow.keras.callbacks import EarlyStopping
from sentence_transformers import SentenceTransformer
from xgboost import plot_importance
```

```
from sklearn.model_selection import GridSearchCV
%matplotlib inline
shap.initjs()
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
import re
from collections import Counter
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from transformers import pipeline
from transformers import pipeline
from textblob import TextBlob
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
```

Figure 1: Required Packages and Dependencies

3 Data Collection

The data was collected using four different APIs: Global Database of Events, Language, and Tone(GDELT), Mediastack, Reddit, and Kraken. The necessary data was retrieved and was downloaded in a ".csv" format file using Pandas package.

1. GDELT: https://github.com/gdelt retrieval.csv
2. Mediastack: https://github.com/mediastack retrieval.csv
3. Reddit: https://github.com/reddit retrieval.csv
4. Kraken: https://github.com/eth daily prices sept 2023 to oct 2024.csv

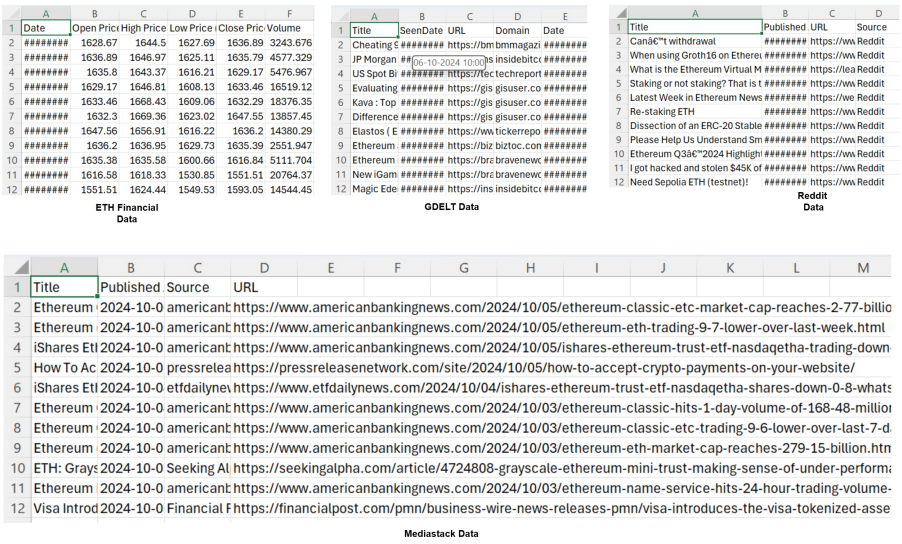

Figure 2: Datasets

4 Data Processing and Transformation

The data was preprocessed by undergoing a series of steps, by removing the URL, HTML tags, numeric characters, tokenization, stemming, lemmatization, removing the duplicate values, removing the abbreviation and was merged with the financial data of Ethereum. Figure 3 shows the script for data preprocessing.

```
[ ]  df1 = df1.drop_duplicates(subset='Title')
     df1 = df1.dropna(subset=['Title'])
```

```
     df1['Title'] = df1['Title'].str.lower()
     df1['Title'] = df1['Title'].str.replace(f'[{string.punctuation}]', '')
     df1['Title'] = df1['Title'].str.replace(r'\d+', '')
     df1['Title'] = df1['Title'].str.replace(r'\s+', ' ', regex=True)
     df1['Title'] = df1['Title'].apply(word_tokenize)
     stop_words = set(stopwords.words('english'))
     df1['Title'] = df1['Title'].apply(lambda x: [word for word in x if word not in stop_words])
     lemmatizer = WordNetLemmatizer()
     df1['Title'] = df1['Title'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
```

```
ethereum_news = pd.merge(ethereum_news, price_data, on='Date', how='inner')
```

Figure 3: Preprocessing and Transformation of the Dataset

5 Feature Engineering and Exploratory Data Analysis

The data was analyzed for its insights, and there were necessary feature engineering steps implemented for creation of new features which added more value to the dataset. The below figure 4 shows some of the main parts like the sentiment analyzers, hybrid sentiments implementations, and average sentiments in the feature engineering technique,

```
analyzer = SentimentIntensityAnalyzer()
ethereum_news['VADER Sentiment'] = ethereum_news['Tokens'].apply(lambda x: analyzer.polarity_scores(x)['compound'])
```

```
[ ] ethereum_news['Price Change (%)'] = ((ethereum_news['Close Price (USD)'] - ethereum_news['Low Price (USD)']) / ethereum_news['Low Price (USD)'])*100
```

```
def bert_sentiment(text):
    result = general_sentiment_classifier(text)
    sentiment_result = result[0]
    score = sentiment_result['score']
    label = sentiment_result['label']

    # Map the star labels to numeric scores
    star_map = {
        '1 star': -2,
        '2 stars': -1,
        '3 stars': 0,
        '4 stars': 1,
        '5 stars': 2
    }


    numeric_score = star_map.get(label, 0)

    return numeric_score, score
```

```
ethereum_news['TextBlob Sentiment'] = ethereum_news['Tokens'].apply(lambda x: TextBlob(x).sentiment.polarity)
```

```
extreme_sentiment = ethereum_news[(ethereum_news['Average Sentiment'] > 0.8) | (ethereum_news['Average Sentiment'] < 0.2)]

# correlation for extreme sentiment cases only
correlation_extreme = extreme_sentiment[['Price Change (%)', 'Average Sentiment']].corr()
print(correlation_extreme)
```

```
ethereum_news['Volume Norm'] = (ethereum_news['Volume'] - ethereum_news['Volume'].min()) / (ethereum_news['Volume'].max() - ethereum_news['Volume'].m

# Hybrid Sentiments
ethereum_news['Sentiment and Volume'] = ethereum_news['Average Sentiment'] * ethereum_news['Volume Norm']

# correlation between this new feature and price change
correlation_sentiment_volume = ethereum_news[['Price Change (%)', 'Sentiment and Volume']].corr()
print(correlation_sentiment_volume)
```

Figure 4: Feature Engineering

After performing the feature engineering on the dataset, the dataset was analyzed for its insights by visualizing some of the seasonal trends and downfall of the cryptocurrency within the short period. Some of the important exploratory data analysis which revealed the maximum insights from the data was correlation, tope 5 sources of the datasets, lagged effects of the price, rolling volatility of the price, and stationarity tests of the time series data, the below figure 5 and 6 highlights the steps mentioned

```
print("Correlation Matrix:")
print(correlation_matrix)

plt.figure(figsize=(10, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='Spectral', fmt=".2f", vmin=-1, vmax=1)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

```
top_n = 5
top_sources = difference.most_common(top_n)
sources, counts = zip(*top_sources)
colors = ['crimson', 'limegreen'] * (len(sources) // 2 + 1)

plt.figure(figsize=(5, 5))
plt.barh(sources, counts, color=colors[:len(sources)])
plt.xlabel('Count')
plt.ylabel('Source')
plt.title(f'Top {top_n} News Sources for Ethereum News')
plt.show()
```

Figure 5: Exploratory Data Analysis

```
ethereum_news['Month'] = ethereum_news['Date'].dt.to_period('M')

start_month = ethereum_news['Month'].min()
end_month = ethereum_news['Month'].max()

plt.figure(figsize=(10, 3))
plt.plot([start_month.start_time, end_month.start_time], [1, 1], color='blue', marker='o', markersize=10)
plt.text(start_month.start_time, 1.05, f'Start: {start_month}', ha='center', fontsize=12)
plt.text(end_month.start_time, 1.05, f'End: {end_month}', ha='center', fontsize=12)
plt.title('Date Range of Ethereum News Headlines (Month to Month)', fontsize=14)
plt.xlabel('Month')
plt.yticks([])
plt.grid(True)
plt.xticks(pd.date_range(start=start_month.start_time, end=end_month.start_time, freq='MS'), rotation=45)
plt.show()
```

```
ethereum_news['Daily Range (USD)'] = ethereum_news['High Price (USD)'] - ethereum_news['Low Price (USD)']
plt.figure(figsize=(10, 4))
plt.plot(ethereum_news['Date'], ethereum_news['Daily Range (USD)'], marker='o', color='blue')
plt.title('Daily Price Range (Volatility) of Ethereum')
plt.xlabel('Date')
plt.ylabel('Daily Range (High - Low) in USD')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
extreme_sentiment['Price Change Lagged'] = extreme_sentiment['Price Change (%)'].shift(-1)
correlation_lagged = extreme_sentiment[['Price Change Lagged', 'Sentiment and Volume']].corr()
print("Correlation with Lagged Price Change:")
print(correlation_lagged)
```

```
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')
adf_test(data_to_test)
```

Figure 6: Exploratory Data Analysis

6 Model Training

There were three models which was trained for a comparative analysis of ML and DL models, the implementation of Random Forest Regressor 7, XG Boost Regressor8, and LSTM 9 are shown below:

```
    # Dataset Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Random Forest Regressor
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    # time computation
    end_time = time.time()
    elapsed_time = end_time - start_time

    results[embedding] = {
        'Mean Squared Error': mse,
        'R² Score': r2,
        'Root Mean Squared Error': rmse,
        'Time Taken (seconds)': elapsed_time
    }
    print(f"{embedding} - Mean Squared Error: {mse}, R² Score: {r2}, Root Mean Squared Error: {rmse}, Time Taken: {elapsed_time:.2f} seconds")

results_df = pd.DataFrame(results).T
print("\nPerformance of different embeddings:")
print(results_df)
```

Figure 7: Random Forest Regressor

```
    # XGBoost Regressor
    modelXG = XGBRegressor(random_state=42)
    grid_search = GridSearchCV(
        estimator=modelXG,
        param_grid=param_grid,
        scoring='neg_mean_squared_error',
        cv=3,
        verbose=1
    )
    start_time = time.time()
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    end_time = time.time()
    elapsed_time = end_time - start_time
    results[embedding] = {
        'Best Parameters': grid_search.best_params_,
        'Mean Squared Error': mse,
        'R² Score': r2,
        'Root Mean Squared Error': rmse,
        'Time Taken (seconds)': elapsed_time
    }
```

Figure 8: XG Boost Regressor with Grid Search

```
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.2, random_state=42)

model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=8, verbose=0)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
end_time = time.time()
elapsed_time = end_time - start_time
results[embedding] = {
    'Mean Squared Error': mse,
    'R² Score': r2,
    'Root Mean Squared Error': rmse,
    'Time Taken (seconds)': elapsed_time
}
print(f"{embedding} - Mean Squared Error: {mse}, R² Score: {r2}, Root Mean Squared Error: {rmse}, Time Taken: {elapsed_time:.2f} seconds")
```

Figure 9: LSTM

The models were tested on various hyperparameter tuning like adjusting the maximum depth of the trees, nodes, learning rate, epochs, optimizers, dropout rates, number of layers and the results produced are the best hyperparameters used in the model.

7 Evaluation

The model's were evaluated on various metrics like R-squared, Mean squared error, Root mean squared error, time complexity and visualizations of model's actual vs predicted data points.

```
print(f"Training Time: {training_time:.2f} seconds")
print(f"Prediction Time: {prediction_time:.2f} seconds")
print(f"RMSE: {rmse}")
print(f"MSE: {mse}")
print(f"R² Score: {r2}")

last_embedding = "TFIDF"
y_test_last = y_test
y_pred_last = model.predict(X_test)

plt.figure(figsize=(5, 5))
plt.scatter(y_test_last, y_test_last, alpha=0.6, color='blue', label='Actual Values')
plt.scatter(y_test_last, y_pred_last, alpha=0.6, color='orange', label='Predicted Values')
plt.plot([y_test_last.min(), y_test_last.max()],
         [y_test_last.min(), y_test_last.max()],
         'r--', lw=2, label='Perfect Prediction Line')
plt.title(f'Predicted vs Actual Price Change (%) using {last_embedding} and Sentiments')
plt.xlabel('Actual Price Change (%)')
plt.ylabel('Predicted Price Change (%)')
plt.legend()
plt.grid()
plt.show()
```

Figure 10: Evaluation Metrics

XG Boost model showed high and accurate results outperforming other two models due to various reasons.

8 Explainable AI

Partial Dependence Plot, Individual Conditional Expectation and Shap was implemented only on the best working model among the three, and the results were visualized for better understanding and evaluation. Figure 11 represents the XAI implemented on XG Boost

```
numeric_features = list(range(X_train.shape[1]))
core_features = [numeric_features[column_names.index(f)] for f in ['Average_Sentiment', 'Sentiment_and_Volume'] if f in column_names]
features_to_analyze = core_features
fig, ax = plt.subplots(figsize=(12, 6))
PartialDependenceDisplay.from_estimator(
    best_model, X_train, features=features_to_analyze,
    kind="average", grid_resolution=50, ax=ax
)
plt.title('Partial Dependence Plot for Average Sentiment and Sentiment-and-Volume')
plt.tight_layout()
plt.show()
```

```
numeric_features = list(range(X_train.shape[1]))
core_features = [numeric_features[column_names.index(f)] for f in ['Average_Sentiment', 'Sentiment_and_Volume'] if f in column_names]

features_to_analyze = core_features
fig, ax = plt.subplots(figsize=(12, 6))
PartialDependenceDisplay.from_estimator(
    best_model, X_train, features=features_to_analyze,
    kind="individual", grid_resolution=50, ax=ax
)

plt.title('ICE Plot for Average Sentiment and Sentiment-and-Volume')
plt.tight_layout()
plt.show()
```

```
[ ] explainer = shap.TreeExplainer(best_model)
shap_values = explainer.shap_values(X_test)
sentiment_volume_idx = X_test.columns.get_loc('Sentiment_and_Volume')
sentiment_volume_shap = shap_values[:, sentiment_volume_idx]
high_sentiment_volume = X_test['Sentiment_and_Volume'] > X_test['Sentiment_and_Volume'].quantile(0.75)
high_sentiment_shap = sentiment_volume_shap[high_sentiment_volume]

positive_contributions = (high_sentiment_shap > 0).sum()
total_contributions = len(high_sentiment_shap)

print(f"Positive SHAP contributions: {positive_contributions} out of {total_contributions}")
```

```
Positive SHAP contributions: 39 out of 40
```

```
explainer = shap.TreeExplainer(best_model)
shap_values_interaction = explainer.shap_interaction_values(X_test)
shap.summary_plot(shap_values_interaction, X_test, feature_names=X_test.columns)
```

```
explainer = shap.TreeExplainer(best_model)
shap_values = explainer.shap_values(X_test)
X_test_subset = X_test.iloc[:, :6]
shap_values_subset = shap_values[:, :6]
shap.summary_plot(shap_values_subset, X_test_subset, feature_names=X_test_subset.columns)
```

Figure 11: XAI implementation