National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Salman Ahmed
Student ID: 23379987

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Salman Ahmed……

Student ID: ……23379987……..……

Programme: ……Artificial Intelligence…… **Year:** ……2024-25….

Module: …… MSc Research Project ……..……

Lecturer: …… Anderson Simiscuka ……..……

Submission Due Date: ……11/08/2024……..……

Project Title: …… Configuration Manual ……..……

Word Count: ……~400…… **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: …… Salman Ahmed ……

Date: ……11/08/2024……

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

Salman Ahmed
23379987

1 Introduction

This is the outline of the system requirements, Programming environment used, implementation details, libraries and dependencies used. Necessary stuff needed to run the code. The complete overview of dataset, preprocessing, models, EDA, and result evaluation is included.

2 Software and Hardware Setup & Requirements

2.1 Hardware Used

Apple Silicon M1, 8GB ram.

2.2 Software Used

IDE: Jupyter Notebook
Python Version: 3.10
Library Management: Pip, Anaconda.
Browser: Google Chrome

Libraries used:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tsa.arima.model import ARIMA
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import DataLoader, Dataset, RandomSampler, SequentialSampler
from sklearn.model_selection import train_test_split
```

```
import pandas as pd
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 matplotlib.pyplot as plt
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 numpy as np
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
```

3 How to execute code:

3.1 Driver Notebook

This notebook executes all the notebooks from preprocessing, transformation, and model training and output. It is created to not have to manually run each cell in each notebook. Change paths and Variables to execute for each cryptocurrency.

3.2 Paths

Driver_notebook:

```
# CHANGE PATH TO YOUR LOCAL
preprocessing_nb = "Desktop/Thesis/Ethereum Project/Code Artifacts/Pre_processing,_Feature_Engineering,_EDA.ipynb"
preparation_nb = "Desktop/Thesis/Ethereum Project/Code Artifacts/Data_Preparation_for_Modelling_.ipynb"
training_nb = "Desktop/Thesis/Ethereum Project/Code Artifacts/Model_Training_and_Evaluation_.ipynb"
```

Pre_processing,_Feature_Engineering,_EDA:

Cell 9:

```
df = pd.read_csv(f"Desktop/Thesis/Ethereum Project/Code Artifacts/news/news_{crypto_name}.csv")
df.head()
```

Cell 20:

```
df1 = pd.read_csv(f"Desktop/Thesis/Ethereum Project/Code Artifacts/gdelt_docapi_full/{crypto_name}_gdelt_docapi.csv")
df1.head()
```

Cell 28:

```
df2 = pd.read_csv(f"Desktop/Thesis/Ethereum Project/Code Artifacts/reddit/reddit_{crypto_name}.csv")
df2.head()
```

Cell 44:

```
price_data = pd.read_csv(f"Desktop/Thesis/Ethereum Project/Code Artifacts/prices/{symbol}_daily_prices_2024-06-01_to_2025-07-01.csv")
price_data.head()
```

3.3 Variable Changes

Driver Notebook:

```
cryptos = [
    "bitcoin" #*****Run step by step*****#
    # "ethereum"
    # "cardano"
    # "solana"
    # "ripple"
    # "avalanche"
    # "chainlink"
    # "polkadot"
    # "litecoin"
    # "tron"
]
```

Pre_processing,_Feature_Engineering,_EDA:

```
# Parameters
crypto_name = "bitcoin"
```

Data_Preparation_for_Modelling_:

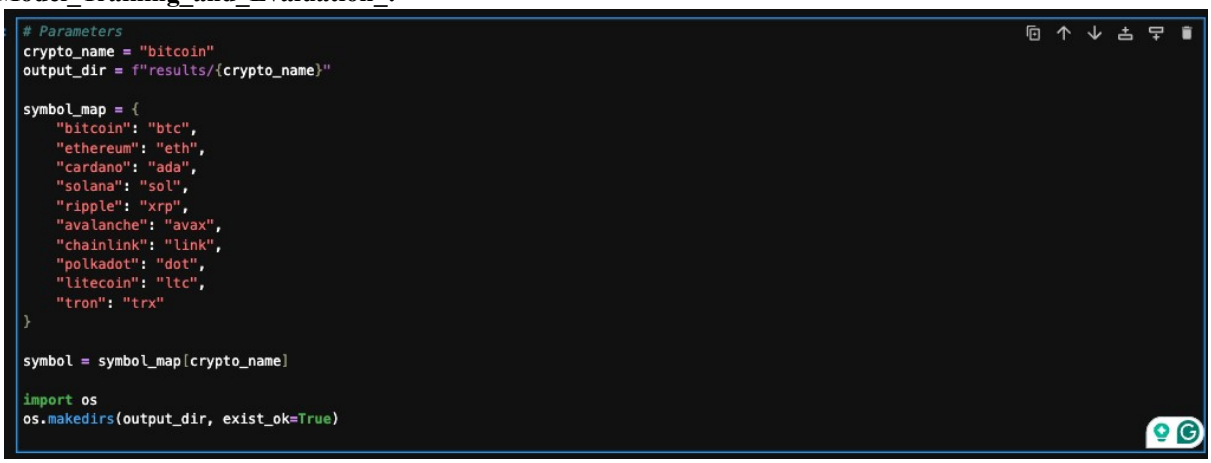
```python
# Parameters
crypto_name = "bitcoin"
output_dir = f"results/{crypto_name}"

symbol_map = {
    "bitcoin": "btc",
    "ethereum": "eth",
    "cardano": "ada",
    "solana": "sol",
    "ripple": "xrp",
    "avalanche": "avax",
    "chainlink": "link",
    "polkadot": "dot",
    "litecoin": "ltc",
    "tron": "trx"
}

symbol = symbol_map[crypto_name]

import os
os.makedirs(output_dir, exist_ok=True)
```

Model_Training_and_Evaluation_:



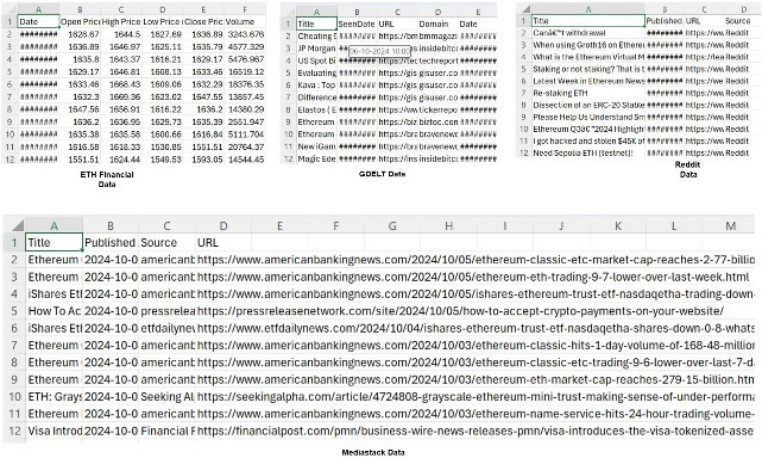
4 Data Sources & Collection

There are 4 data sources:

1. News Article: MediaStack: http://api.mediastack.com/v1/news
2. Daily Crypto Prices: Kraken: https://api.kraken.com/0/public/OHLC
3. Social News: Reddit: Credentials and details embedded in code.
4. News: GDELT: https://api.gdeltproject.org/api/v2/doc/doc

Simply run each cell in Data_Retrival notebook and it will save data in local drive with segregated folders.

Sample Data:



ETH Financial Data

GDELT Data

Reddit Data

Title	Published	Source	URL
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/05/ethereum-classic-etc-market-cap-reaches-2-77-billio
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/05/ethereum-eth-trading-9-7-lower-over-last-week.html
iShares Et	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/05/ishares-ethereum-trust-etf-nasdaqetha-trading-down
How To Ac	2024-10-0	pressrelea	https://pressreleasenetwork.com/site/2024/10/05/how-to-accept-crypto-payments-on-your-website/
iShares Et	2024-10-0	etfdailyne	https://www.etfdailynews.com/2024/10/04/ishares-ethereum-trust-etf-nasdaqetha-shares-down-0-8-whats
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/03/ethereum-classic-hits-1-day-volume-of-168-48-millio
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/03/ethereum-classic-etc-trading-9-6-lower-over-last-7-d
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/03/ethereum-eth-market-cap-reaches-279-15-billion.htn
ETH: Gray	2024-10-0	Seeking Al	https://seekingalpha.com/article/4724808-grayscale-ethereum-mini-trust-making-sense-of-under-perform
Ethereum	2024-10-0	americant	https://www.americanbankingnews.com/2024/10/03/ethereum-name-service-hits-24-hour-trading-volume-
Visa Introd	2024-10-0	Financial F	https://financialpost.com/pmn/business-wire-news-releases-pmn/visa-introduces-the-visa-tokenized-asse

Mediastack Data

5 Data Preprocessing and Transformation to fit

After going through several preprocessing stages, including deleting the URL, HTML tags, numeric characters, tokenization, stemming, lemmatization, duplicate value removal, and abbreviation removal, the data was combined with all cryptos financial data.

```
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])
```

6 EDA & Feature Engineering

6.1 Sentiment Analysis

Following code presents sentimental analysis part:

```
###VADER
```

```
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)'
```

```
ethereum_news['Cleaned Tokens'] = ethereum_news['Tokens'].apply(clean_text)
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
```

```
###TextBLob
```

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

```
# 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)
```

6.2 EDA

```
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()
```

```
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)
```

7 Models

7.1 Random Forest Regression

```
    # 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)
```

7.2 XGBoost

```
    # 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)
    y_pred = np.nan_to_num(y_pred)  # 🔥 Prevent NaN errors - to prevent NaN errors

    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
    }
    print(f"{embedding} - Best Parameters: {grid_search.best_params_}, Mean Squared Error: {mse}, R² Score: {r2}, Root Mean Squared Error: {rm

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

7.3 CatBoost

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

    model = CatBoostRegressor(verbose=0)
    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)

    print(f"✅ {embedding} → R²: {r2:.4f}, RMSE: {rmse:.4f}, MSE: {mse:.4f}")
    catboost_results[embedding] = {
        "MSE": mse,
        "RMSE": rmse,
        "R²": r2
    }
results_df = pd.DataFrame(catboost_results).T
results_df.to_csv(f"{output_dir}/{crypto_name}_catboost_results.csv")
print(f"📁 Saved to {output_dir}/{crypto_name}_catboost_results.csv")
```

7.4 LightBGM

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

    model = LGBMRegressor()
    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)

    print(f"✅ {embedding} → R²: {r2:.4f}, RMSE: {rmse:.4f}, MSE: {mse:.4f}")
    lgbm_results[embedding] = {
        "MSE": mse,
        "RMSE": rmse,
        "R²": r2
    }
results_df = pd.DataFrame(lgbm_results).T
results_df.to_csv(f"{output_dir}/{crypto_name}_lgbm_results.csv")
print(f"📁 Saved to {output_dir}/{crypto_name}_lgbm_results.csv")
```

7.5 Hist Gradient Regressor

```
    model = HistGradientBoostingRegressor(random_state=42)
    start_time = time.time()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    elapsed = time.time() - start_time

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    print(f"Time: {elapsed:.2f} seconds")
    print(f"RMSE: {rmse}")
    print(f"MSE: {mse}")
    print(f"R² Score: {r2}")
    print(f"✅ {embedding} → R²: {r2:.4f}, RMSE: {rmse:.4f}, MSE: {mse:.4f}")

    results_hgb[embedding] = {
        'Mean Squared Error': mse,
        'Root Mean Squared Error': rmse,
        'R² Score': r2,
        'Time Taken (seconds)': elapsed
    }

# Save or display results
results_hgb_df = pd.DataFrame(results_hgb).T
print("\n📊 HistGradientBoostingRegressor Results:")
print(results_hgb_df)

results_hgb_df.to_csv(f"{output_dir}/{crypto_name}_hgb_model_results.csv")
print(f"📁 Saved to {output_dir}/{crypto_name}_hgb_model_results.csv")
```

8 Results/ Evaluations

Every model was tested with four embeddings across various metrics R^2, MSE, RMSE.

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

9 Explainable Artificial Intelligence (XAI)

PDP, ICE, and SHAP was used on XGBoost with TFIDF embedding and results were presented for understanding and showcase results.

9.1 PDP

```
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()
```

9.2 SHAP

```
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)
```

9.3 ICE

```
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()
```