

Configuration Manual

MSc Research Project
Programme Name

Manoj Crasta
Student ID: x23199156

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Manoj Crasta
Student ID:	x23199156
Programme:	Programme Name
Year:	2024
Module:	MSc Research Project
Supervisor:	Abdul Shahid
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	591
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:	
Date:	28th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Manoj Crasta
x23199156

1 Introduction

This configuration manual will provide detailed instructions how to setup the system or device for this study. This manual includes thorough explanation on how to carry out the research study from start to finish. It also contains the machine specifications and configurations needed to develop and execute all the models without any problems. It contains steps on how to setup the basic environment for the project, along with information on the necessary applications and packages to integrate into the working environment.

2 Project Files Detail

2.1 Datasets

This project include 4 historical dataset files and a JSON file folder consisting financial news articles encoded in JSON structure.

2.2 Google Collab File

Google collab is used for this project development. Hence one google collab file will be there in the project folder.

3 System Specification

Figure 1 shows the system specifications. And Figure 2 shows the Google Collab specifications.

4 Software Used

1. Google Drive
2. Google Collab

LAPTOP-T0F180CQ
HP Pavilion Laptop 14-ec1xxx

Device specifications

Device name

LAPTOP-T0F180CQ

Processor

AMD Ryzen 5 5625U with Radeon Graphics2.30 GHz

Installed RAM

16.0 GB (15.3 GB usable)

Device ID

4725083F-5A4E-4523-9CF1-51233D670D46

Product ID

00342-42631-83659-AAOEM

System type

64-bit operating system, x64-based processor

Pen and touch

No pen or touch input is available for this display

Related links

Domain or workgroup

System protection

Advanced system settings

Windows specifications

Edition

Windows 11 Home Single Language

Version

23H2

Installed on

27-05-2023

OS build

22631.4541

Experience

Windows Feature Experience Pack 1000.22700.1055.0

Figure 1: System Specifications

Specification	Details
Processor	Intel Xeon CPU (varies)
GPU Instance	NVIDIA Tesla T4
RAM	12 GB
Disk Space	100 GB (temporary storage)
Max Lifetime of VM	12 hours

Figure 2: Collab Specifications

4.1 Libraries Used

- os
- json
- datetime
- pandas
- seaborn
- matplotlib.pyplot
- tabulate
- transformers
- sklearn
- xgboost
- tensorflow.keras
- keras_tuner

5 Project Development

These steps gives a detailed guide how to implement the project in google collab.

5.1 Uploading Project Folder To Google Drive

First upload the project folder to the google drive as shown in Figure 3.

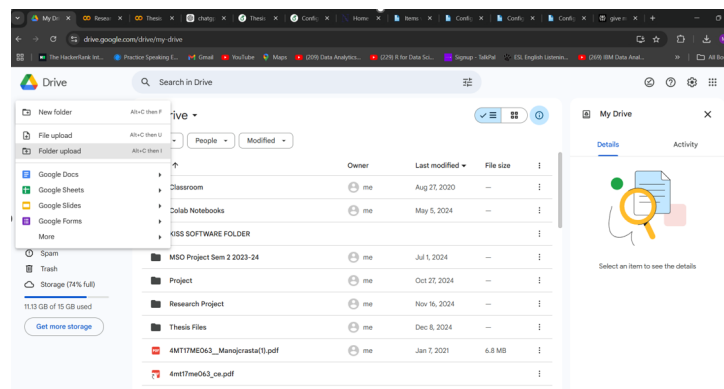


Figure 3: Uploading Project Folder

5.2 Mount Google Drive

Open the collab file from the project folder. And run the mount drive code in the first cell as shown in Figure 4.

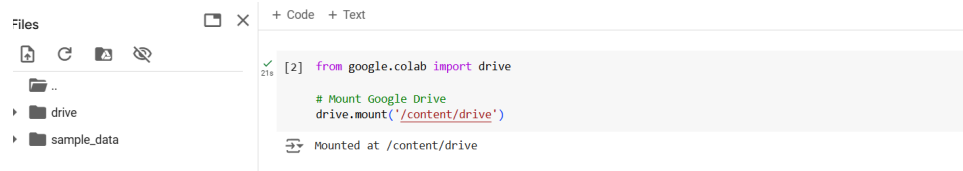


Figure 4: Mount Google Drive

5.3 Changing The File Path

Change the file paths if necessary as shown in Figure 6.

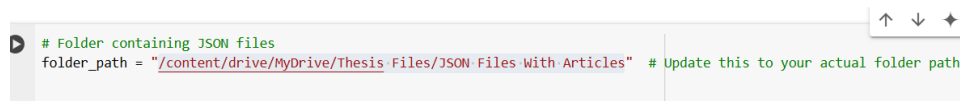


Figure 5: Changing the path of files

5.4 Importing Libraries

Importing and installing all the necessary libraries that will be used in the project.

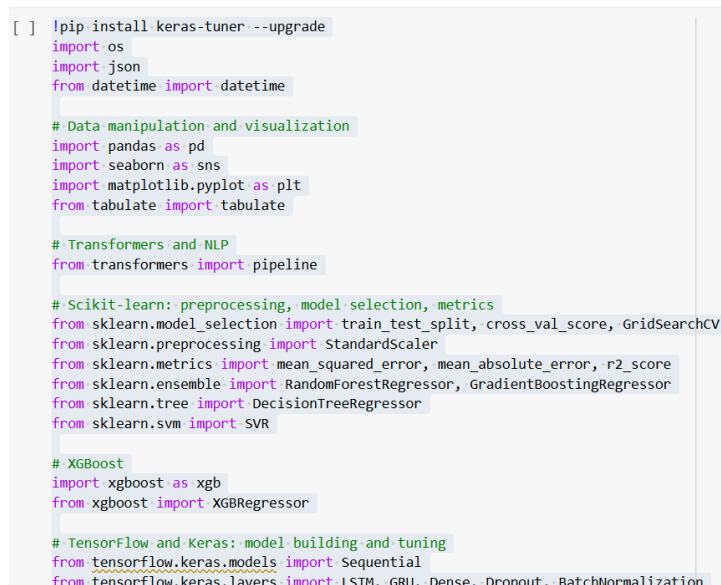
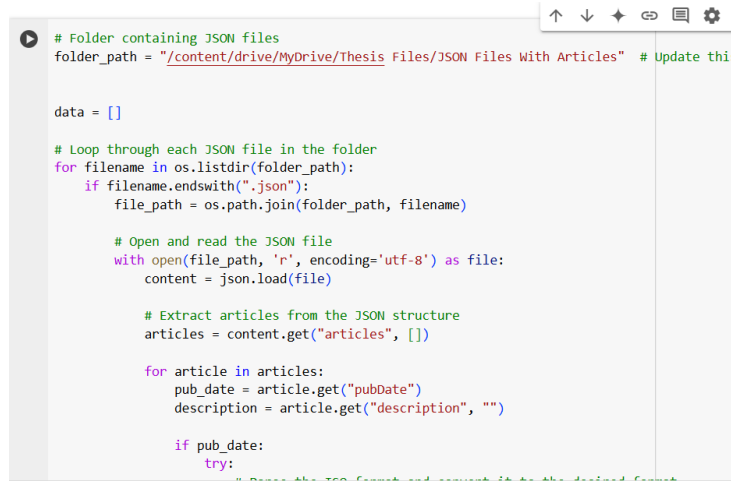


Figure 6: Importing Libraries

5.5 Pre-processing

5.5.1 Processing JSON Files

JSON files stored in the folder 'JSON Files With Articles' will be extracted one by one and processed to get an excel file with columns pub_date and description.



```

# Folder containing JSON files
folder_path = "/content/drive/MyDrive/Thesis Files/JSON Files With Articles" # Update this

data = []

# Loop through each JSON file in the folder
for filename in os.listdir(folder_path):
    if filename.endswith(".json"):
        file_path = os.path.join(folder_path, filename)

        # Open and read the JSON file
        with open(file_path, 'r', encoding='utf-8') as file:
            content = json.load(file)

        # Extract articles from the JSON structure
        articles = content.get("articles", [])

        for article in articles:
            pub_date = article.get("pubDate")
            description = article.get("description", "")

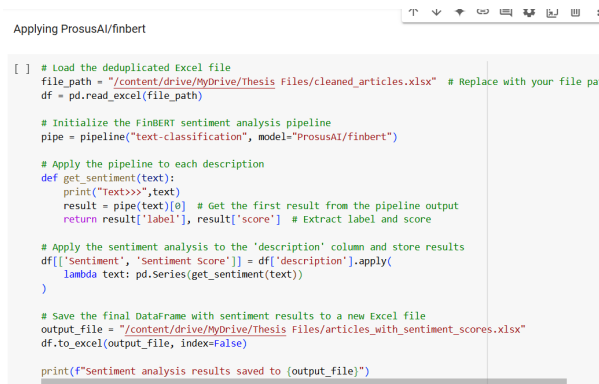
            if pub_date:
                try:
                    # Parse the pub_date and convert it to the desired format

```

Figure 7: Extracting pub_date and description from the JSON files

5.5.2 Sentiment Analysis

This below code uses ProsusAI/FinBERT model for sentiment analysis. And then single sentiment score will be generated in the place of 2 columns sentiment label and sentiment score Figure 9.



```

Applying ProsusAI/finbert

[ ] # Load the deduplicated Excel file
file_path = "/content/drive/MyDrive/Thesis Files/cleaned_articles.xlsx" # Replace with your file path
df = pd.read_excel(file_path)

# Initialize the FinBERT sentiment analysis pipeline
pipe = pipeline("text-classification", model="ProsusAI/finbert")

# Apply the pipeline to each description
def get_sentiment(text):
    print("Text>>>", text)
    result = pipe(text)[0] # Get the first result from the pipeline output
    return result['label'], result['score'] # Extract label and score

# Apply the sentiment analysis to the 'description' column and store results
df[['Sentiment', 'Sentiment Score']] = df['description'].apply(
    lambda text: pd.Series(get_sentiment(text))
)

# Save the final DataFrame with sentiment results to a new Excel file
output_file = "/content/drive/MyDrive/Thesis Files/articles_with_sentiment_scores.xlsx"
df.to_excel(output_file, index=False)

print(f"Sentiment analysis results saved to {output_file}")

```

Figure 8: Sentiment Analysis

5.5.3 Processing of Historical datasets

Below codes performs the preprocessing of historical hourly datasets such as XAU/USD rate, Crude Oil, SP 500 and VIX.

5.5.4 Data Splitting and Scalling

Data is split into 80% training and 20% test set. And then StandardScaler was used for scaling Figure 15.

This script combines sentiment label and score multiple into a single sentiment score column

```
[ ] input_file = "/content/drive/MyDrive/Thesis Files/articles_with_sentiment_scores.xlsx" # Update this

df = pd.read_excel(input_file)

# Transform the Sentiment Score based on Sentiment type
df["Sentiment Score"] = df.apply(
    lambda row: row["Sentiment Score"] if row["Sentiment"] == "positive"
    else -row["Sentiment Score"] if row["Sentiment"] == "negative"
    else 0,
    axis=1
)

# Select only the required columns
new_df = df[["pub_date", "description", "Sentiment Score"]]

# Save the resulting table to a new Excel file
output_file = "/content/drive/My Drive/Thesis Files/single_sentiment_score_per_article.xlsx"
new_df.to_excel(output_file, index=False)

print("Transformation complete. Output saved to:", output_file)
```

Figure 9: Generating single sentiment score

This block of code merges the separate date and time columns from the hourly XAU/USD dataset into a single timestamp column, enabling seamless integration with the sentiment score dataframe for further analysis.

```
[ ] # Load the Excel file
file_path = "/content/drive/My Drive/Thesis Files/xauusd_hourly.xlsx" # Update this path to your Excel
df = pd.read_excel(file_path)

# Combine DATE and TIME columns into a new TIMESTAMP column
df['TIMESTAMP'] = pd.to_datetime(df['DATE'].astype(str) + ' ' + df['TIME'].astype(str), format='%Y.%m.%d %H:%M')

# Drop the original DATE and TIME columns
df.drop(['DATE', 'TIME', 'TICKVOL', 'HIGH', 'LOW', 'OPEN', 'SPREAD', 'VOL'], axis=1, inplace=True)

# Reorder columns to place TIMESTAMP at the beginning
df = df[['TIMESTAMP', 'CLOSE']]

# Display the final DataFrame
print("Processed DataFrame with TIMESTAMP:")
print(df.head())

# Save the result to a new Excel file
output_file_path = "/content/drive/My Drive/Thesis Files/cleaned_hourly_xauusd_data.xlsx" # Specify ,
df.to_excel(output_file_path, index=False)

print(f"Processed data with TIMESTAMP has been saved to {output_file_path}")
```

Figure 10: XAU/USD pre-processing

This code processes crude oil hourly csv data by combining date and time into a single datetime column, filtering it for a specified date range, and retaining only the datetime and close price columns. The cleaned and filtered data is then saved to an Excel file for further analysis.

```
[ ]

# Load the CSV file, specifying the delimiter and column names
data = pd.read_csv('/content/drive/My Drive/Research Project/crudeOil.csv', delimiter=';',
    names=['date', 'time', 'open', 'high', 'low', 'close', 'volume'])

# Combine date and time columns into a single datetime column
data['datetime'] = pd.to_datetime(data['date'] + ' ' + data['time'], format='%d/%m/%Y %H:%M:%S')

# Drop the original date, time, and unnecessary columns, keeping only datetime and close
data = data[['datetime', 'close']]

# Filter data for the specified date range
start_date = '2020-07-01'
end_date = '2023-12-31'
filtered_data = data[(data['datetime'] >= start_date) & (data['datetime'] <= end_date)]

# Check the processed data
print(filtered_data.head())

# Save the filtered data to a new CSV file
filtered_data.to_excel('/content/drive/My Drive/Thesis Files/crudeOil_cleaned.xlsx', index=False)
```

Figure 11: Crude Oil pre-processing

This code processes S&P 500 hourly csv data by creating a single datetime column from separate date and time columns, filtering the data for a specific date range, and retaining only the datetime and close price columns. The cleaned data is then saved as an Excel file for further use.

```
[ ] # Load the CSV file, specifying the delimiter and column names
data = pd.read_csv('/content/drive/MyDrive/Research Project/es-1h.csv', delimiter=';',
names=['date', 'time', 'open', 'high', 'low', 'close', 'volume'])

# Combine date and time columns into a single datetime column
data['datetime'] = pd.to_datetime(data['date'] + ' ' + data['time'], format='%d/%m/%Y %H:%M:%S')

# Drop the original date, time, and unnecessary columns, keeping only datetime and close
data = data[['datetime', 'close']]

# Filter data for the specified date range
start_date = '2020-07-01'
end_date = '2023-12-31'
filtered_data = data[(data['datetime'] >= start_date) & (data['datetime'] <= end_date)]

# Check the processed data
print(filtered_data.head())

# Save the filtered data to a new CSV file
filtered_data.to_excel('/content/drive/MyDrive/Thesis Files/S&P500_cleaned.xlsx', index=False)
```

Figure 12: SP 500 pre-processing

```
# Load the CSV file, specifying the delimiter and column names
data = pd.read_csv('/content/drive/My Drive/Research Project/vix-1h.csv', delimiter=';',
names=['date', 'time', 'open', 'high', 'low', 'close', 'volume'])

# Combine date and time columns into a single datetime column
data['datetime'] = pd.to_datetime(data['date'] + ' ' + data['time'], format='%d/%m/%Y %H:%M:%S')

# Drop the original date, time, and unnecessary columns, keeping only datetime and close
data = data[['datetime', 'close']]

# Filter data for the specified date range
start_date = '2020-07-01'
end_date = '2023-12-31'
filtered_data = data[(data['datetime'] >= start_date) & (data['datetime'] <= end_date)]

# Check the processed data
print(filtered_data.head())

# Save the filtered data to a new CSV file
filtered_data.to_excel('/content/drive/MyDrive/Thesis Files/vix_cleaned.xlsx', index=False)
```

Figure 13: VIX pre-processing

Scripts merges the all the individual datasets into one using the date column

```
[ ] # Load the data
crude_data = pd.read_excel('/content/drive/My Drive/Thesis Files/crudeOil_cleaned.xlsx', parse_dates=[
xau_usd_data = pd.read_excel('/content/drive/My Drive/Thesis Files/cleaned_xauusd_sentiment_data.xlsx')
vix_data = pd.read_excel('/content/drive/MyDrive/Thesis Files/vix_cleaned.xlsx', parse_dates=['datetime'])
sp_data = pd.read_excel('/content/drive/MyDrive/Thesis Files/S&P500_cleaned.xlsx', parse_dates=['datetime'])

# Preprocess the datasets
xau_usd_data['datetime'] = pd.to_datetime(xau_usd_data['TIMESTAMP'])
crude_data.rename(columns={'close': 'crude_close'}, inplace=True)
vix_data.rename(columns={'close': 'vix_close'}, inplace=True)
sp_data.rename(columns={'close': 'sp_close'}, inplace=True)

# Merge XAU/USD with crude oil data
merged_data = pd.merge(xau_usd_data, crude_data[['datetime', 'crude_close']], on='datetime', how='inner')
merged_data.drop('TIMESTAMP', axis=1, inplace=True)

# Merge the result with VIX data
final_merged_data = pd.merge(merged_data, vix_data[['datetime', 'vix_close']], on='datetime', how='inner')

# Merge the result with S&P 500 data
finally_merged_data = pd.merge(final_merged_data, sp_data[['datetime', 'sp_close']], on='datetime', how='inner')

# Check for null values
null_counts = finally_merged_data.isnull().sum()
print("Null values in each column:")
print(null_counts)
```

Figure 14: Merging All Data

Dataset splitting into Test and Train set Using StandardScaler to normalize the data

```
[ ] # Select features and target
X = finally_merged_data[['Sentiment Score', 'crude_close', 'vix_close', 'sp_close']]
y = finally_merged_data['CLOSE']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 15: Data splitting and scalling

5.5.5 Model building

First the hyperparameter tuning will be done and then best model will predict the test set.

```
Random Forest Hyperparameter Tuning

# Define the Random Forest model
rf_model = RandomForestRegressor(
    n_estimators=100, # Number of trees in the forest
    random_state=42, # For reproducibility
    n_jobs=-1 # Use all available CPUs for training
)

# Cross-validation to get a better sense of model performance
cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=5, scoring='neg_mean_squared_error')
print(f"Cross-validation MSE scores: {cv_scores}")
print(f"Average MSE from cross-validation: {-cv_scores.mean():.4f}")

# Hyperparameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 150], # Number of trees to try
    'max_depth': [None, 10, 20, 30], # Max depth of trees
    'min_samples_split': [2, 5, 10], # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4] # Minimum number of samples required to be at a leaf node
}

grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Best parameters from grid search
print(f"Best parameters from GridSearchCV: {grid_search.best_params_}")
```

Figure 16: Hyperparameter Tuning

```
Random Forest Best Params

[ ] # Define the Random Forest model with best parameters
best_rf_model = RandomForestRegressor(
    n_estimators=150, # Best number of trees
    max_depth=30, # Best max depth
    min_samples_split=2, # Best min samples to split
    min_samples_leaf=1, # Best min samples at leaf node
    random_state=42, # For reproducibility
    n_jobs=-1 # Use all available CPUs for training
)

# Train the model
best_rf_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = best_rf_model.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"R^2 Score: {r2:.4f}")
```

Figure 17: Best Model Used for prediction

Output from hyperparameter tuning: Figure 18.

5.5.6 Model Evaluation

Evaluation Metrics of Random Forest.Figure 21

5.6 Same steps from model building will be followed to other models

5.7 R^2 Scores of Different Machine Learning Models

Below graph shows the performance of all models in terms of R^2 Scores :Figure 22.

Cross-validation MSE scores: [-226.47352602 -243.35760551 -218.30592619 -217.13246802 -208.19748727]
Average MSE from cross-validation: 222.6934
Best parameters from GridSearchCV: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}

```
RandomForestRegressor
```

```
RandomForestRegressor(max_depth=30, n_estimators=150, n_jobs=-1, random_state=42)
```

Figure 18: Random Forest Best parameters

Mean Squared Error (MSE): 186.9992
Mean Absolute Error (MAE): 7.2035
R² Score: 0.9782

Figure 19: Random Forest Metrics

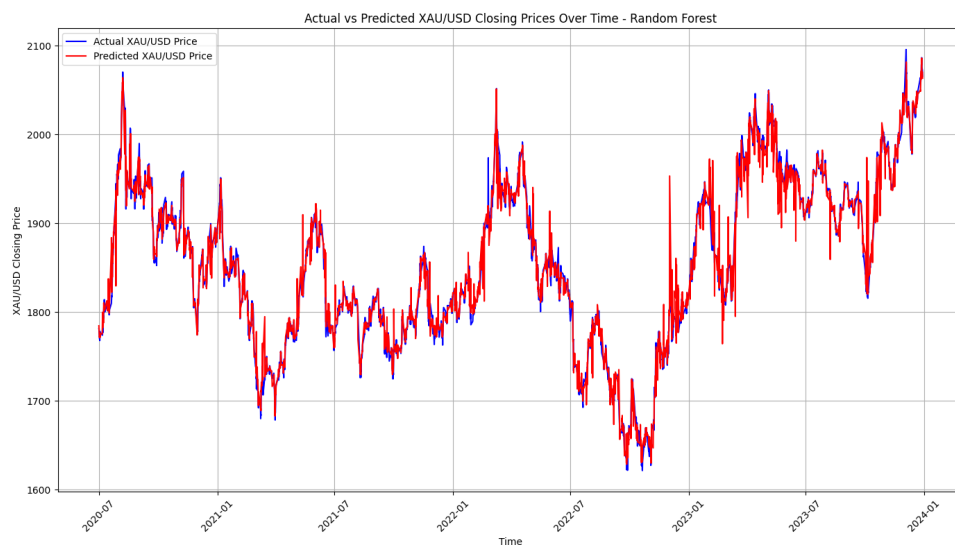


Figure 20: Predicted vs Actual XAU/USD rates over time

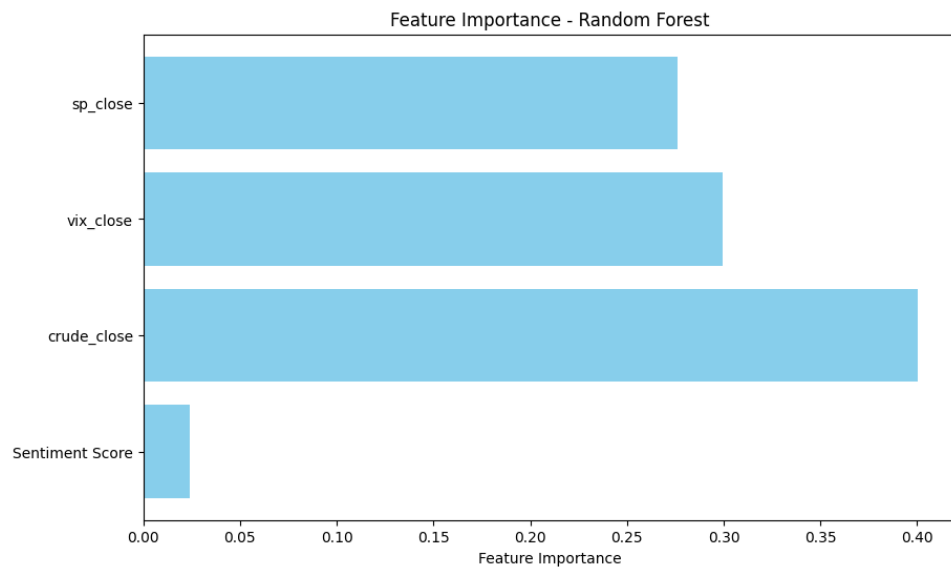


Figure 21: Feature Importance

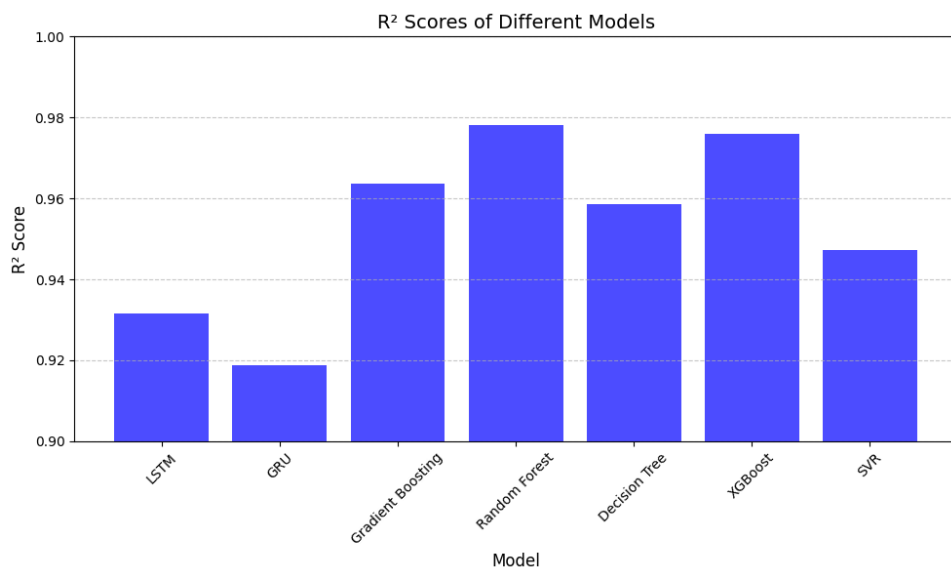


Figure 22: R² Scores of Different Machine Learning Models