Configuration Manual

MSc Research Project
MSc in Data Analytics

Saiharsha Gurijala
Student ID: 23194341

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saiharsha Gurijala
Student ID:	23194341
Programme:	MSc in Data Analytics **Year:** 2024
Module:	MSc Research Project
Lecturer:	Aaloka Anant
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual: Enhancing Financial Forecasting through Transformer Models Using Social Media and News Insights
Word Count:	1864 **Page Count:** 14

Signature: Saiharsha Gurijala

Date: 12/12/2024

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

Saiharsha Gurijala
Student ID: 23194341

1 Introduction

This configuration manual provides a comprehensive guide to replicate the project titled **Enhancing Financial Forecasting through Transformer Models Using Social Media and News Insights.** It describes the required environment, datasets, and implementation steps for deploying the system. Since all necessary datasets and pretrained models are stored in the Stock_Web_UI folder, the key focus is executing the stock_web_UI.ipynb notebook, which consolidates the workflow for prediction and visualization. To run the website and check the results, refer to Sections 7 and 8 of this manual for detailed guidance.

2 Hardware and Software Specifications

2.1 Hardware Requirements

- **Operating System:** Windows 11
- **Processor:** 8-core CPU (Intel i7/AMD Ryzen 7 or equivalent)
- **GPU (Required):** NVIDIA RTX 3060 or higher with CUDA support
- **RAM:** 16 GB or higher
- **Storage:** 50 GB of free disk space

2.2 Software Requirements

Programming Language: Python 3.10
Execution Platform: Google Colab Pro (required for GPU acceleration)
Libraries: Install the following dependencies in Google Colab Pro using the specified commands:

Libraries	Installation Command
pandas, numpy	!pip install pandas numpy
yfinance, praw, requests	!pip install yfinance praw requests
pytorch-lightning, torch	!pip install lightning torch
pytorch-lightning, torch	!pip install pytorch-forecasting
transformers	!pip install transformers
flask, pyngrok	!pip install flask pyngrok
matplotlib, plotly, scikit-learn	!pip install matplotlib plotly scikit-learn

Table 1: About Libraries Installation

3 Cloud Storage and Execution Environment

This project leverages cloud-based tools for execution and storage:

- **Google Colab Pro:** Executes the notebooks with GPU acceleration for faster processing.
- **Ngrok:** Hosts the Flask web application and provides a public URL for accessibility.

4 Dataset Details

4.1 Sources

- **Reddit Data (2018-2024):** Extracted from publicly available stock-related discussions on [Kaggle](https://www.kaggle.com).
- **News Articles (2018-2024):** Historical news data was retrieved using the GDELT API, while the latest news data is dynamically fetched from the News API.
- **Stock Market Data (2018-2024):** Retrieved via Yahoo Finance API.

4.2 Dataset Files

Organized into the following directories:

Folder	**Files**
Initial_Datasets	stock_index.csv, posts.csv, wallstreetbets_2022.csv
Sentiment_Analysis	tesla_reddit_data_18_to_24.csv, aapl_reddit_data_18_to_24.csv, tesla_news_data_18_to_24.csv, aapl_news_data_18_to_24.csv, tesla_stocks_18_to_24.csv, aapl_stocks_18_to_24.csv
Stock_Web_UI	**Trained models:** (tesla_trained_tft_model.pth, aapl_trained_tft_model.pth), **Sentiment-enriched datasets:** (tsla_data_with_sentiment_analysis_from_18_to_24.csv, aapl_data_with_sentiment_analysis_from_18_to_24.csv) **Flask web application files:** (home.html, error.html, predict.html, strategy.html, predict_lstm.html, compare_models.html).

Table 2: Folder and File details

5 Implementation Details

5.1 Supporting Notebooks (Optional)

The following notebooks were used for data preparation and training but are not required to execute the project:

1. get_reddit_and_news_data.ipynb:

- Downloads news data and stocks data using respective APIs.
- Filters data for relevant stock tickers (Tesla and Apple).
- Stores the data in DataFrames and saves it into CSV files, which are ready for sentiment analysis.

2. sentiment_analysis.ipynb:

- Performs sentiment analysis using the FinBERT model for Reddit and news data.
- Merges all the data into a single dataset and saves it as CSV files.

3. train_telsa_tft_model.ipynb & train_apple_tft_model.ipynb:

- Preprocesses stock, sentiment data for Tesla and Apple.
- Trains Temporal Fusion Transformer (TFT) models and saves them as trained files.
- Visualizes predictions versus actual values and feature importance.
- Tests the LSTM model, evaluates model metrics, and simulates a trading strategy for validation.

Note: The outputs from these four notebooks (processed datasets and trained models) are already stored in the Stock_Web_UI folder. You do not need to rerun these notebooks.

5.2 Primary Notebook for Execution

Stock_web_UI.ipynb file:

The key notebook for deployment and execution is stock_web_UI.ipynb, which:

- Provides a Flask-based web interface for stock price predictions, trading strategy simulation, and model comparison.
- Integrates pretrained Temporal Fusion Transformer (TFT) models (tesla_trained_tft_model.pth and aapl_trained_tft_model.pth) and LSTM models for predictions.
- Dynamically updates the dataset by integrating live data from APIs (Yahoo Finance, News API, and Reddit).
- Generates interactive plots (using Plotly) to compare actual and predicted stock prices, as well as trading signals (Buy/Sell markers).
- Includes a public URL for external access via Ngrok, enabling seamless deployment and usability.

6 Evaluation

Metrics:

- **Accuracy Metrics:** Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score.

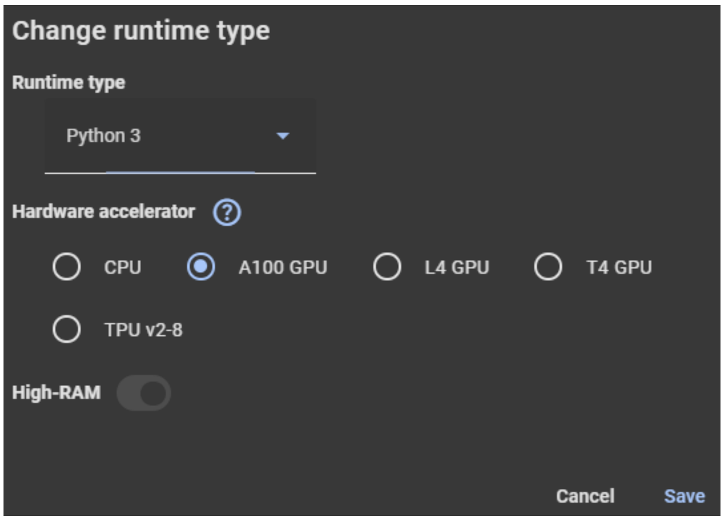
Visualizations:

- Actual vs Predicted Prices (Plotly graphs).
- Model Comparison (Bar charts for metrics).
- Trading Signals (Buy/Sell markers on price charts).

7 Instructions for Running the Project

7.1 Environment Setup:

- Open **stock_web_UI.ipynb** in Google Colab.
- To enable **GPU acceleration** in Google Colab, go to the ‘Runtime’ menu, select ‘Change runtime type’, choose your desired ‘Hardware accelerator’ (e.g., A100 GPU), and click Save.
- Ensure that all files from the **Stock_Web_UI** folder are uploaded to the session storage.


Change runtime type

Runtime type: Python 3

Hardware accelerator: CPU [✓ A100 GPU] L4 GPU T4 GPU TPU v2-8

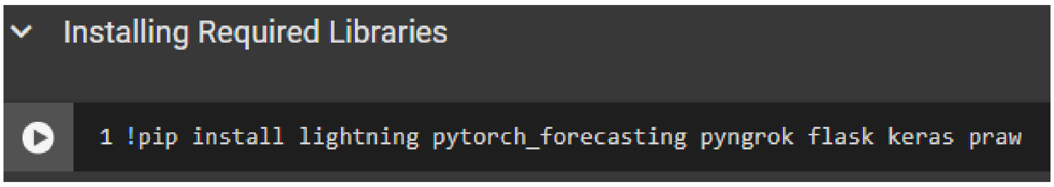
High-RAM

Fig 1: Runtime type UI

7.2 Execution Steps & Explanation of Code Workflow:

1. Installing Required Libraries:

The notebook begins by installing the necessary Python libraries such as lightning, pytorch_forecasting, pyngrok, flask, keras, and praw using pip.

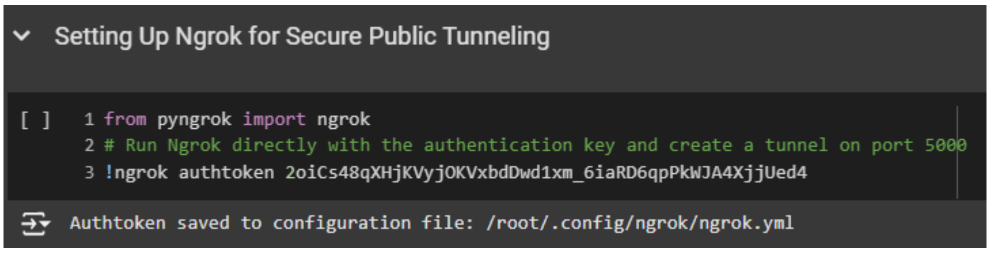

Installing Required Libraries

```
!pip install lightning pytorch_forecasting pyngrok flask keras praw
```

Fig 2: Installing Libraries

2. Setting Up Ngrok for Public Access:

The Ngrok authentication token is configured to allow secure public tunneling. This step enables the Flask application to be accessible externally via a public URL.


Setting Up Ngrok for Secure Public Tunneling

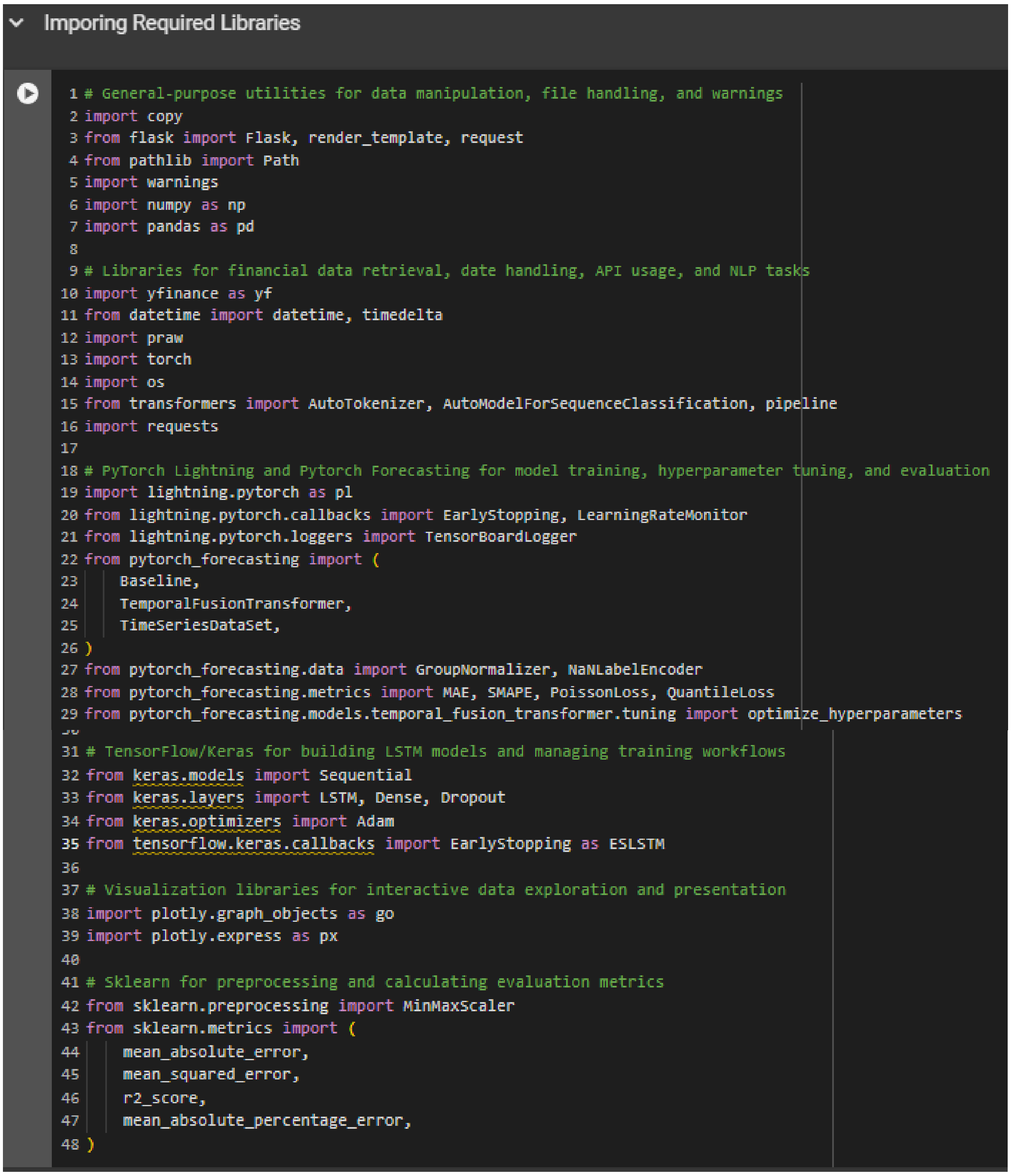
```
from pyngrok import ngrok
# Run Ngrok directly with the authentication key and create a tunnel on port 5000
!ngrok authtoken 2oiCs48qXHjKVyjOKVxbdDwd1xm_6iaRD6qpPkWJA4XjjUed4
```

```
Authtoken saved to configuration file: /root/.config/ngrok/ngrok.yml
```

Fig 3: Setting Up Ngrok

3. Importing Required Libraries & files:

The project imports a comprehensive set of libraries for data handling, API interaction, deep learning models, visualization, and performance evaluation. Key libraries include PyTorch Forecasting for transformer-based modelling, TensorFlow/Keras for LSTM, and Plotly for interactive visualizations.

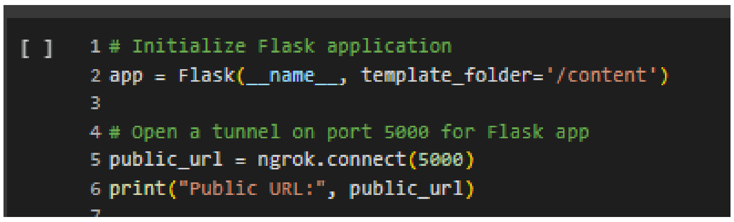


```python
# General-purpose utilities for data manipulation, file handling, and warnings
import copy
from flask import Flask, render_template, request
from pathlib import Path
import warnings
import numpy as np
import pandas as pd

# Libraries for financial data retrieval, date handling, API usage, and NLP tasks
import yfinance as yf
from datetime import datetime, timedelta
import praw
import torch
import os
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline
import requests

# PyTorch Lightning and Pytorch Forecasting for model training, hyperparameter tuning, and evaluation
import lightning.pytorch as pl
from lightning.pytorch.callbacks import EarlyStopping, LearningRateMonitor
from lightning.pytorch.loggers import TensorBoardLogger
from pytorch_forecasting import (
    Baseline,
    TemporalFusionTransformer,
    TimeSeriesDataSet,
)
from pytorch_forecasting.data import GroupNormalizer, NaNLabelEncoder
from pytorch_forecasting.metrics import MAE, SMAPE, PoissonLoss, QuantileLoss
from pytorch_forecasting.models.temporal_fusion_transformer.tuning import optimize_hyperparameters

# TensorFlow/Keras for building LSTM models and managing training workflows
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping as ESLSTM

# Visualization libraries for interactive data exploration and presentation
import plotly.graph_objects as go
import plotly.express as px

# Sklearn for preprocessing and calculating evaluation metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
    mean_absolute_percentage_error,
)
```

Fig 4: Importing Libraries

4. Initializing Flask Application:

A Flask application is initialized with a specified template folder. The Ngrok public URL is configured to provide external access to the application running locally.

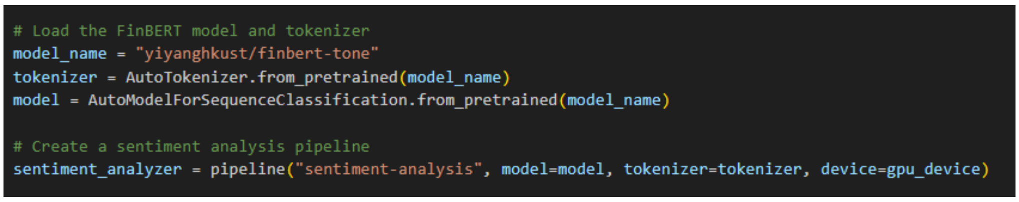


```python
# Initialize Flask application
app = Flask(__name__, template_folder='/content')

# Open a tunnel on port 5000 for Flask app
public_url = ngrok.connect(5000)
print("Public URL:", public_url)
```

Fig 5: Initializing Flask

5. Loading the FinBERT Model and Tokenizer:

The FinBERT model and its tokenizer are loaded to perform sentiment analysis on financial news and Reddit data. This model analyses sentiment (positive, neutral, negative) from textual data and assigns corresponding sentiment scores.

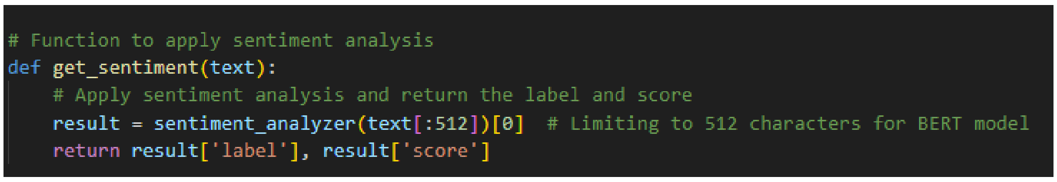


```python
# Load the FinBERT model and tokenizer
model_name = "yiyanghkust/finbert-tone"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Create a sentiment analysis pipeline
sentiment_analyzer = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer, device=gpu_device)
```

Fig 6: Runtime type change

6. Sentiment Analysis Function:

A function get_sentiment applies FinBERT to a given text, truncating it to 512 characters (model input limit) and returning the sentiment label and score.

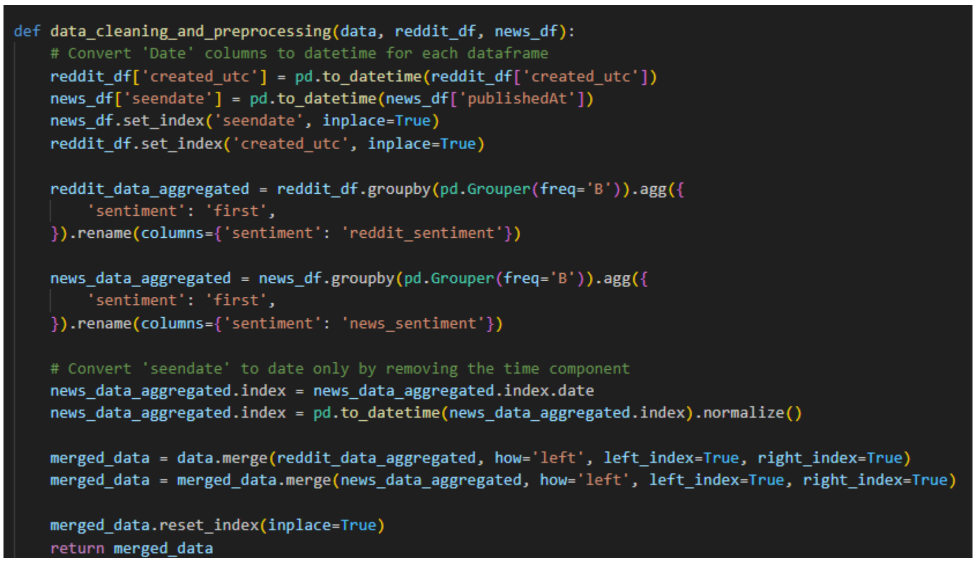


```python
# Function to apply sentiment analysis
def get_sentiment(text):
    # Apply sentiment analysis and return the label and score
    result = sentiment_analyzer(text[:512])[0]  # Limiting to 512 characters for BERT model
    return result['label'], result['score']
```

Fig 7: Sentimental Analysis Function

7. Data Cleaning and Preprocessing:

Data from Reddit and news articles is cleaned and preprocessed. Sentiment scores are aggregated based on business days. Merging and normalization processes integrate sentiment and stock data for further modelling.



```python
def data_cleaning_and_preprocessing(data, reddit_df, news_df):
    # Convert 'Date' columns to datetime for each dataframe
    reddit_df['created_utc'] = pd.to_datetime(reddit_df['created_utc'])
    news_df['seendate'] = pd.to_datetime(news_df['publishedAt'])
    news_df.set_index('seendate', inplace=True)
    reddit_df.set_index('created_utc', inplace=True)

    reddit_data_aggregated = reddit_df.groupby(pd.Grouper(freq='B')).agg({
        'sentiment': 'first',
    }).rename(columns={'sentiment': 'reddit_sentiment'})

    news_data_aggregated = news_df.groupby(pd.Grouper(freq='B')).agg({
        'sentiment': 'first',
    }).rename(columns={'sentiment': 'news_sentiment'})

    # Convert 'seendate' to date only by removing the time component
    news_data_aggregated.index = news_data_aggregated.index.date
    news_data_aggregated.index = pd.to_datetime(news_data_aggregated.index).normalize()

    merged_data = data.merge(reddit_data_aggregated, how='left', left_index=True, right_index=True)
    merged_data = merged_data.merge(news_data_aggregated, how='left', left_index=True, right_index=True)

    merged_data.reset_index(inplace=True)
    return merged_data
```

Fig 8: Cleaning and preprocessing function

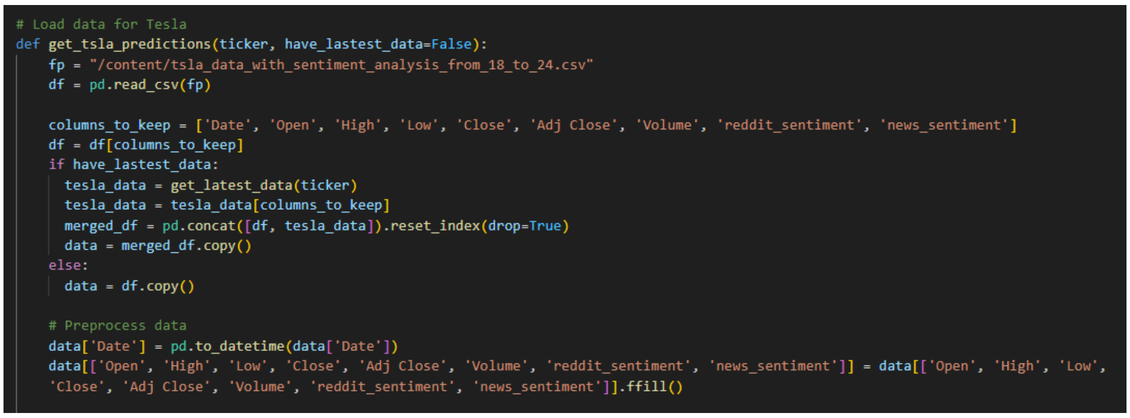
8. Loading Tesla Data:

The **get_tsla_predictions** function is responsible for loading Tesla stock data, enriched with sentiment fields derived from Reddit and news articles. Missing values in the dataset are systematically filled to ensure data integrity. The data is then preprocessed and prepared for the Temporal Fusion Transformer (TFT) model.

Additionally, the **get_latest_data** function dynamically fetches real-time data, including:

- **Stock Prices:** Collected from December 1st, 2024, to the most recent date using Yahoo Finance.
- **News Articles:** Retrieved through the NEWS API, providing up-to-date information on Tesla-related topics.
- **Reddit Data:** Gathered using the PRAW library, covering relevant posts from multiple subreddits.

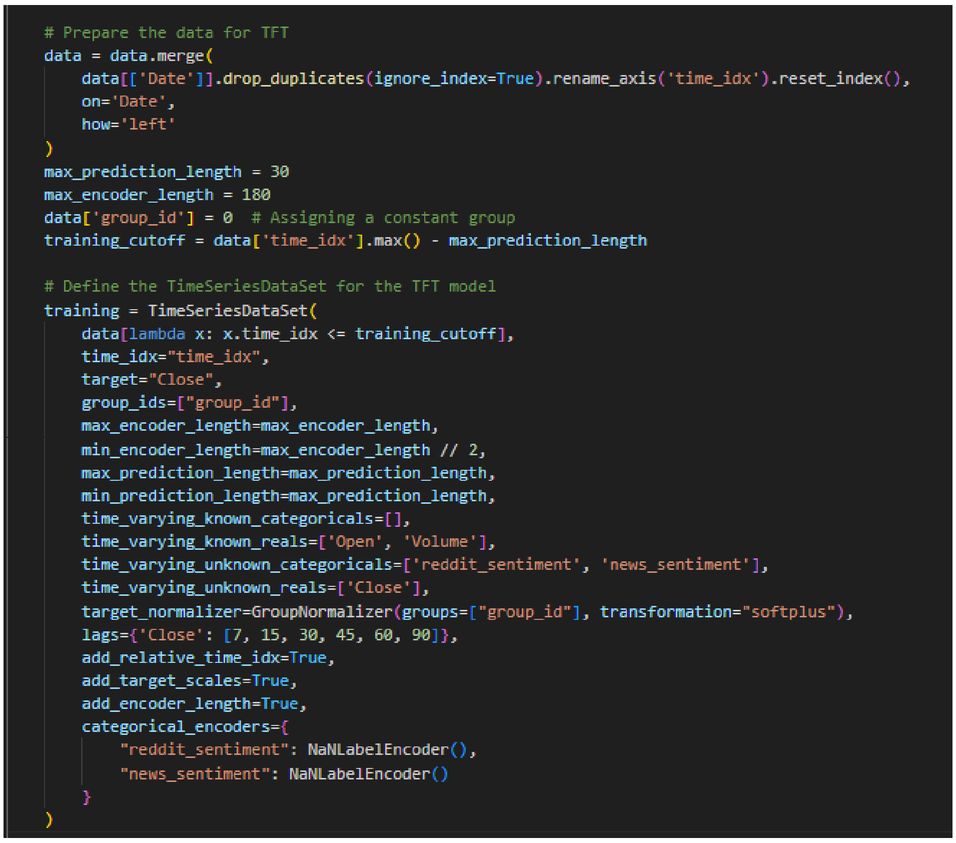
The function integrates these sources, performs sentiment analysis on the news and Reddit text using FinBERT, and prepares the sentiment-enriched data for seamless modelling.



```python
# Load data for Tesla
def get_tsla_predictions(ticker, have_lastest_data=False):
    fp = "/content/tsla_data_with_sentiment_analysis_from_18_to_24.csv"
    df = pd.read_csv(fp)

    columns_to_keep = ['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'reddit_sentiment', 'news_sentiment']
    df = df[columns_to_keep]
    if have_lastest_data:
      tesla_data = get_latest_data(ticker)
      tesla_data = tesla_data[columns_to_keep]
      merged_df = pd.concat([df, tesla_data]).reset_index(drop=True)
      data = merged_df.copy()
    else:
      data = df.copy()

    # Preprocess data
    data['Date'] = pd.to_datetime(data['Date'])
    data[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'reddit_sentiment', 'news_sentiment']] = data[['Open', 'High', 'Low',
    'Close', 'Adj Close', 'Volume', 'reddit_sentiment', 'news_sentiment']].ffill()
```

Fig 9: Loading TSLA data function for TFT Model

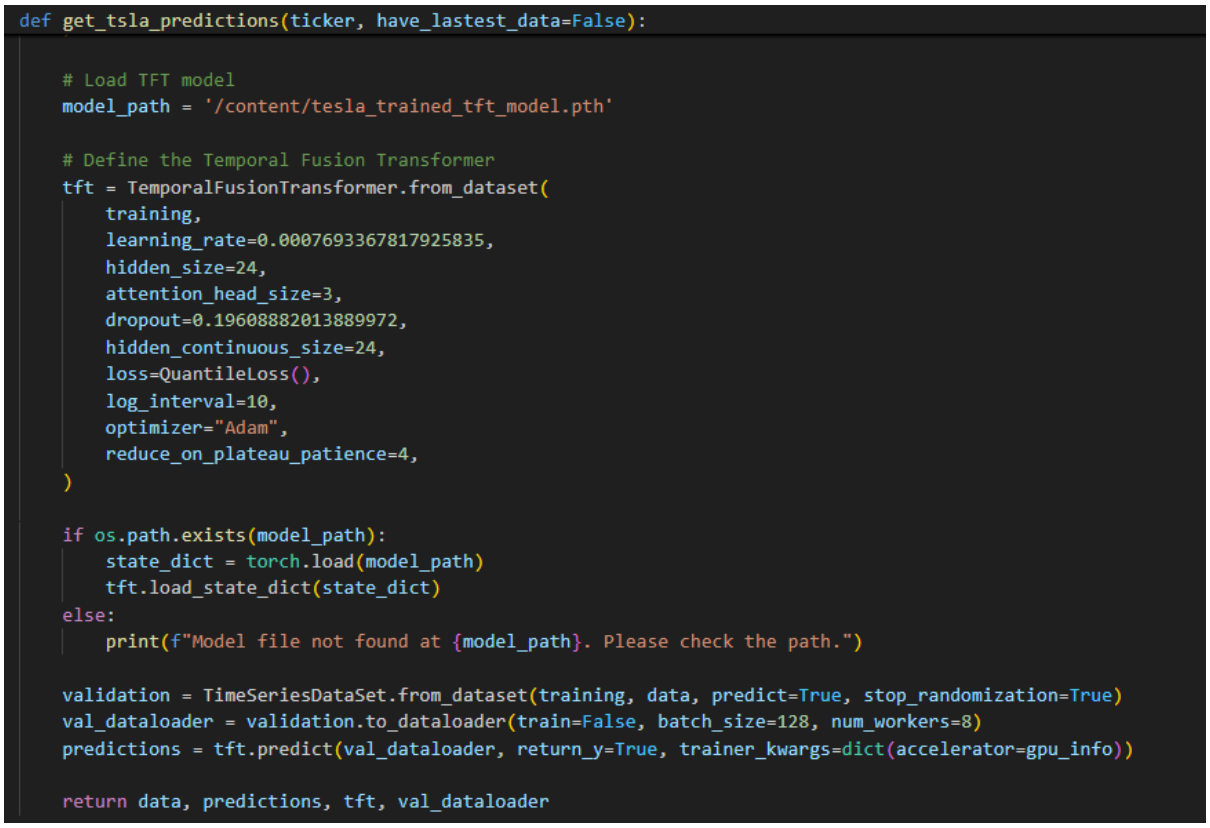
The data is restructured into a TimeSeriesDataSet format compatible with the TFT model. This includes configuring lags, encoder lengths, and categorical encoders for sentiment columns.



```python
# Prepare the data for TFT
data = data.merge(
    data[['Date']].drop_duplicates(ignore_index=True).rename_axis('time_idx').reset_index(),
    on='Date',
    how='left'
)
max_prediction_length = 30
max_encoder_length = 180
data['group_id'] = 0  # Assigning a constant group
training_cutoff = data['time_idx'].max() - max_prediction_length

# Define the TimeSeriesDataSet for the TFT model
training = TimeSeriesDataSet(
    data[lambda x: x.time_idx <= training_cutoff],
    time_idx="time_idx",
    target="Close",
    group_ids=["group_id"],
    max_encoder_length=max_encoder_length,
    min_encoder_length=max_encoder_length // 2,
    max_prediction_length=max_prediction_length,
    min_prediction_length=max_prediction_length,
    time_varying_known_categoricals=[],
    time_varying_known_reals=['Open', 'Volume'],
    time_varying_unknown_categoricals=['reddit_sentiment', 'news_sentiment'],
    time_varying_unknown_reals=['Close'],
    target_normalizer=GroupNormalizer(groups=["group_id"], transformation="softplus"),
    lags={'Close': [7, 15, 30, 45, 60, 90]},
    add_relative_time_idx=True,
    add_target_scales=True,
    add_encoder_length=True,
    categorical_encoders={
        "reddit_sentiment": NaNLabelEncoder(),
        "news_sentiment": NaNLabelEncoder()
    }
)
```

Fig 10: TSLA TFT model TimeSeriesDataSet

The pretrained TFT model for Tesla is loaded, and the validation dataset is prepared. The model predicts stock prices using the time-series structure of the data.

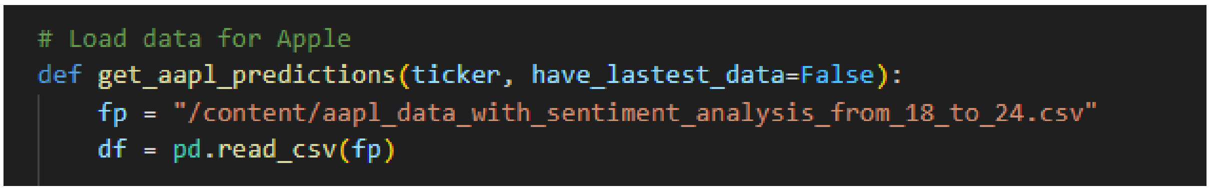


```python
def get_tsla_predictions(ticker, have_lastest_data=False):

    # Load TFT model
    model_path = '/content/tesla_trained_tft_model.pth'

    # Define the Temporal Fusion Transformer
    tft = TemporalFusionTransformer.from_dataset(
        training,
        learning_rate=0.0007693367817925835,
        hidden_size=24,
        attention_head_size=3,
        dropout=0.19608882013889972,
        hidden_continuous_size=24,
        loss=QuantileLoss(),
        log_interval=10,
        optimizer="Adam",
        reduce_on_plateau_patience=4,
    )

    if os.path.exists(model_path):
        state_dict = torch.load(model_path)
        tft.load_state_dict(state_dict)
    else:
        print(f"Model file not found at {model_path}. Please check the path.")

    validation = TimeSeriesDataSet.from_dataset(training, data, predict=True, stop_randomization=True)
    val_dataloader = validation.to_dataloader(train=False, batch_size=128, num_workers=8)
    predictions = tft.predict(val_dataloader, return_y=True, trainer_kwargs=dict(accelerator=gpu_info))

    return data, predictions, tft, val_dataloader
```

Fig 11: Loading Trained TSLA TFT Model

9. Loading Apple Data:

Similar to Tesla, Apple stock data is loaded using get_aapl_predictions.

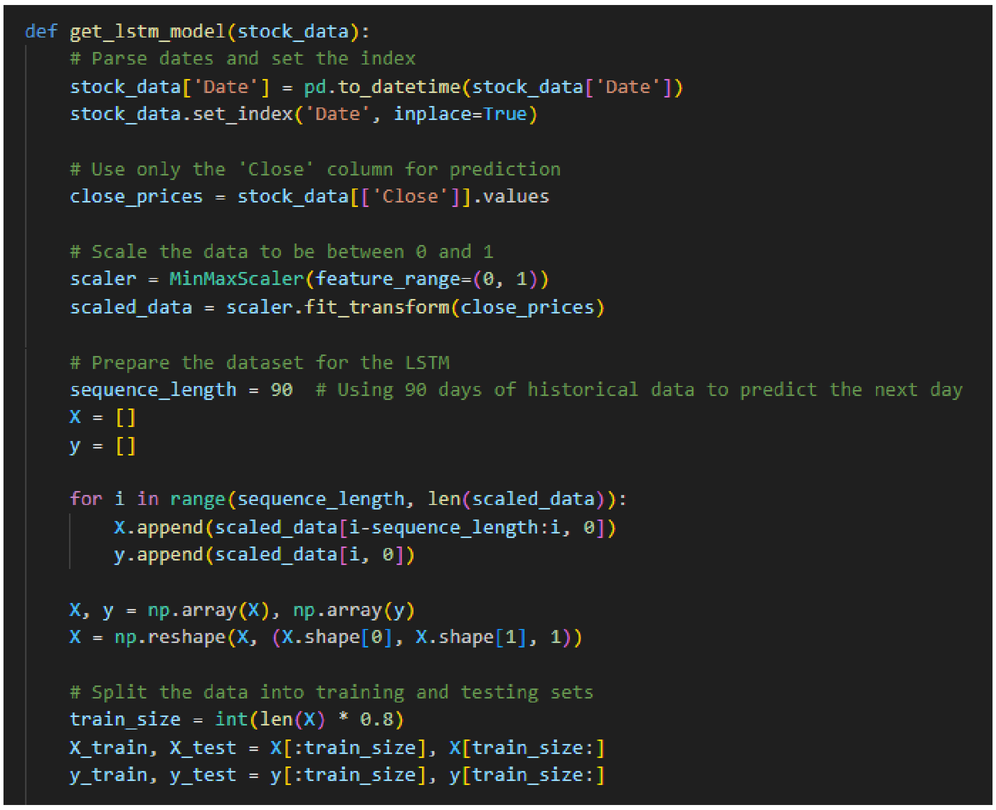


```python
# Load data for Apple
def get_aapl_predictions(ticker, have_lastest_data=False):
    fp = "/content/aapl_data_with_sentiment_analysis_from_18_to_24.csv"
    df = pd.read_csv(fp)
```

Fig 12: Loading AAPL data function for TFT Model

10. Building and Training the LSTM Model:

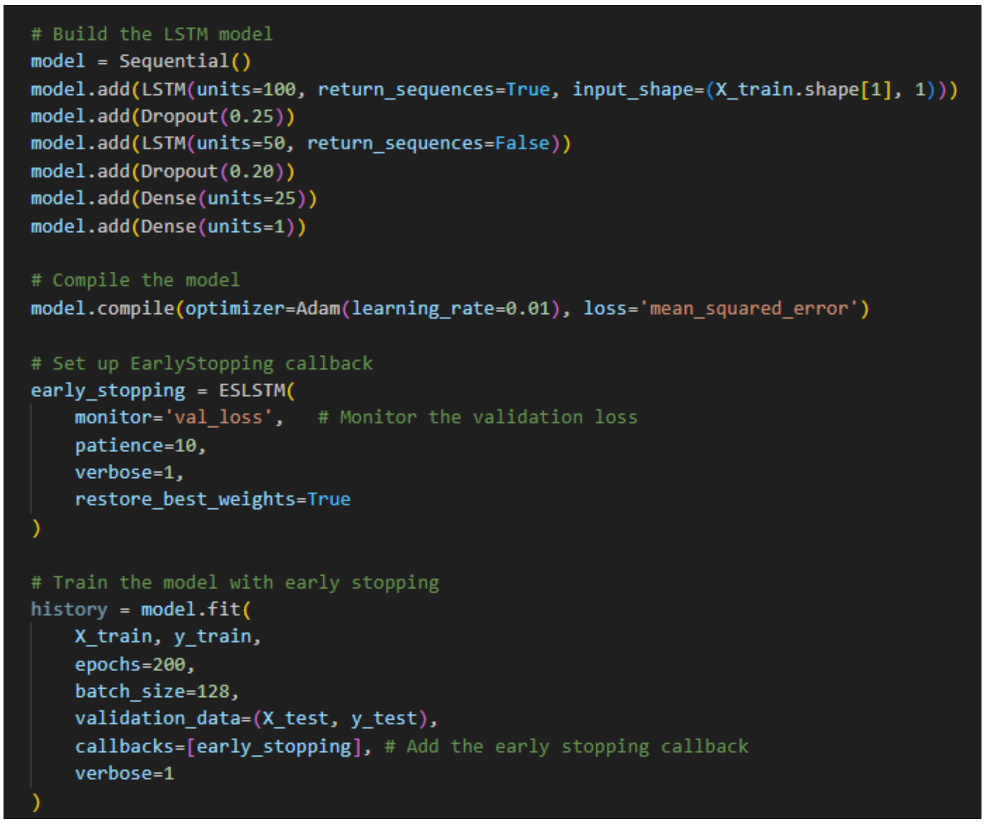
The LSTM model setup involves scaling data, preparing sequences of 90 days, and splitting it into training and testing sets. This is crucial for capturing sequential patterns in stock price movements.



```python
def get_lstm_model(stock_data):
    # Parse dates and set the index
    stock_data['Date'] = pd.to_datetime(stock_data['Date'])
    stock_data.set_index('Date', inplace=True)

    # Use only the 'Close' column for prediction
    close_prices = stock_data[['Close']].values

    # Scale the data to be between 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(close_prices)

    # Prepare the dataset for the LSTM
    sequence_length = 90  # Using 90 days of historical data to predict the next day
    X = []
    y = []

    for i in range(sequence_length, len(scaled_data)):
        X.append(scaled_data[i-sequence_length:i, 0])
        y.append(scaled_data[i, 0])

    X, y = np.array(X), np.array(y)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    # Split the data into training and testing sets
    train_size = int(len(X) * 0.8)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]
```

Fig 13: LSTM Model Function

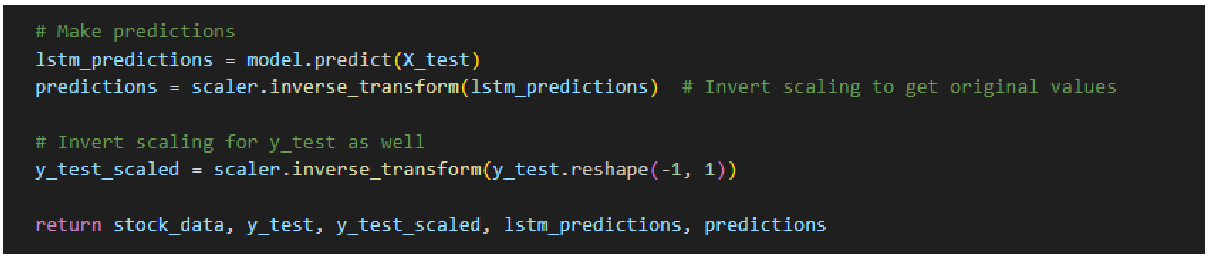
The LSTM model is constructed with layers for processing sequential data. Training is performed with early stopping to prevent overfitting. Model predictions are scaled back to their original range.



```python
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.25))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.20))
model.add(Dense(units=25))
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')

# Set up EarlyStopping callback
early_stopping = ESLSTM(
    monitor='val_loss',   # Monitor the validation loss
    patience=10,
    verbose=1,
    restore_best_weights=True
)

# Train the model with early stopping
history = model.fit(
    X_train, y_train,
    epochs=200,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping], # Add the early stopping callback
    verbose=1
)
```

Fig 14: LSTM Model Function

Predictions are generated for the test dataset. Both the predicted and actual stock prices are rescaled for evaluation and visualization.

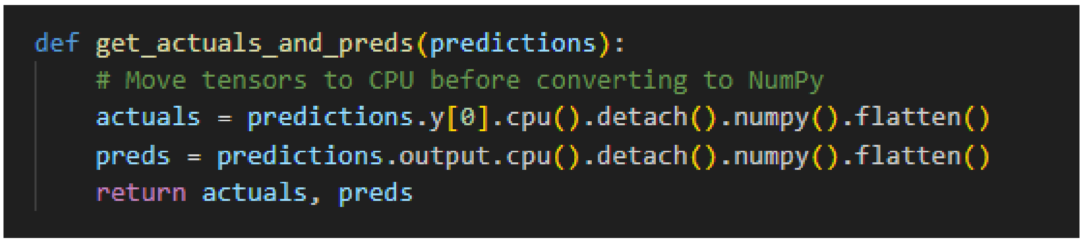


```
# Make predictions
lstm_predictions = model.predict(X_test)
predictions = scaler.inverse_transform(lstm_predictions)  # Invert scaling to get original values

# Invert scaling for y_test as well
y_test_scaled = scaler.inverse_transform(y_test.reshape(-1, 1))

return stock_data, y_test, y_test_scaled, lstm_predictions, predictions
```

Fig 15: LSTM Model Function

11. Extracting Actual and Predicted Values:

A utility function retrieves actual and predicted stock prices from model outputs, preparing them for comparison and analysis in visualizations.

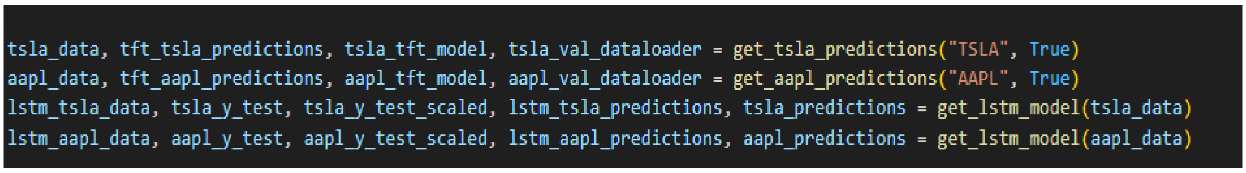


```
def get_actuals_and_preds(predictions):
    # Move tensors to CPU before converting to NumPy
    actuals = predictions.y[0].cpu().detach().numpy().flatten()
    preds = predictions.output.cpu().detach().numpy().flatten()
    return actuals, preds
```

Fig 16: Extracting Actual and Predicted Values

12. Combining Tesla and Apple Data Processing:

Both Tesla and Apple data are processed using the respective TFT and LSTM models. Outputs are consolidated to ensure uniformity in predictions and evaluations.

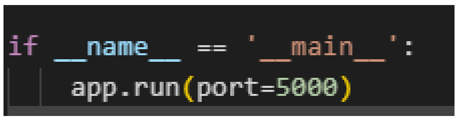


```
tsla_data, tft_tsla_predictions, tsla_tft_model, tsla_val_dataloader = get_tsla_predictions("TSLA", True)
aapl_data, tft_aapl_predictions, aapl_tft_model, aapl_val_dataloader = get_aapl_predictions("AAPL", True)
lstm_tsla_data, tsla_y_test, tsla_y_test_scaled, lstm_tsla_predictions, tsla_predictions = get_lstm_model(tsla_data)
lstm_aapl_data, aapl_y_test, aapl_y_test_scaled, lstm_aapl_predictions, aapl_predictions = get_lstm_model(aapl_data)
```

Fig 17: Combining Tesla and Apple Data for Both Models

13. Running the Flask Application:

This code snippet ensures that the Flask application runs on port 5000. The (if __name__ == "__main__") construct is a standard Python idiom used to execute the application only when the script is run directly, not when it is imported as a module. By invoking app.run(port=5000), the application is deployed and becomes accessible via the public Ngrok URL generated earlier in the notebook.



```
if __name__ == '__main__':
    app.run(port=5000)
```

Fig 18: Running the Flask Application

This structured workflow ensures efficient data handling, modelling, and deployment for real-time stock price predictions and analysis using state-of-the-art techniques.

8 How to Use the Website UI

The deployed web application offers a user-friendly interface for stock analysis and prediction. Below is a detailed guide on how the website's navigation and features work, based on the HTML pages and corresponding functionalities:

1. Homepage (home.html):

Overview: The homepage serves as the entry point for users. It allows them to select a stock ticker (e.g., TSLA or AAPL) and a model (Temporal Fusion Transformer or LSTM).

Inputs:

- **Ticker Symbol:** Enter the stock symbol (TSLA or AAPL).
- **Model Choice:** Select either "Temporal Fusion Transformer" or "LSTM".

Actions:

- **Predict Button:** Submits the inputs to fetch predictions and display results based on the selected model.
- **Compare Models Button:** Redirects the user to a comparison of model performances for the selected stock.

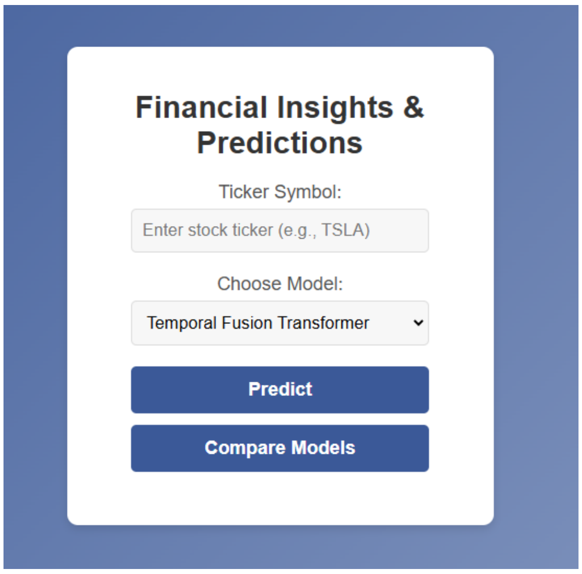
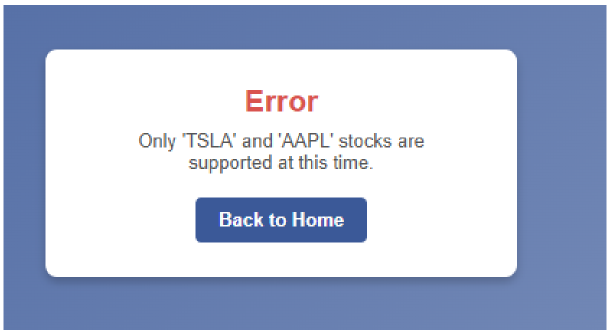

Financial Insights & Predictions
Ticker Symbol:
Enter stock ticker (e.g., TSLA)
Choose Model:
Temporal Fusion Transformer
Predict
Compare Models

Fig 19: Home Page

2. Error Page (error.html):

Functionality: Displays an error message if an unsupported stock ticker is entered.

Back to Home: Redirects the user to the homepage to re-enter valid inputs.


Error

Only 'TSLA' and 'AAPL' stocks are supported at this time.

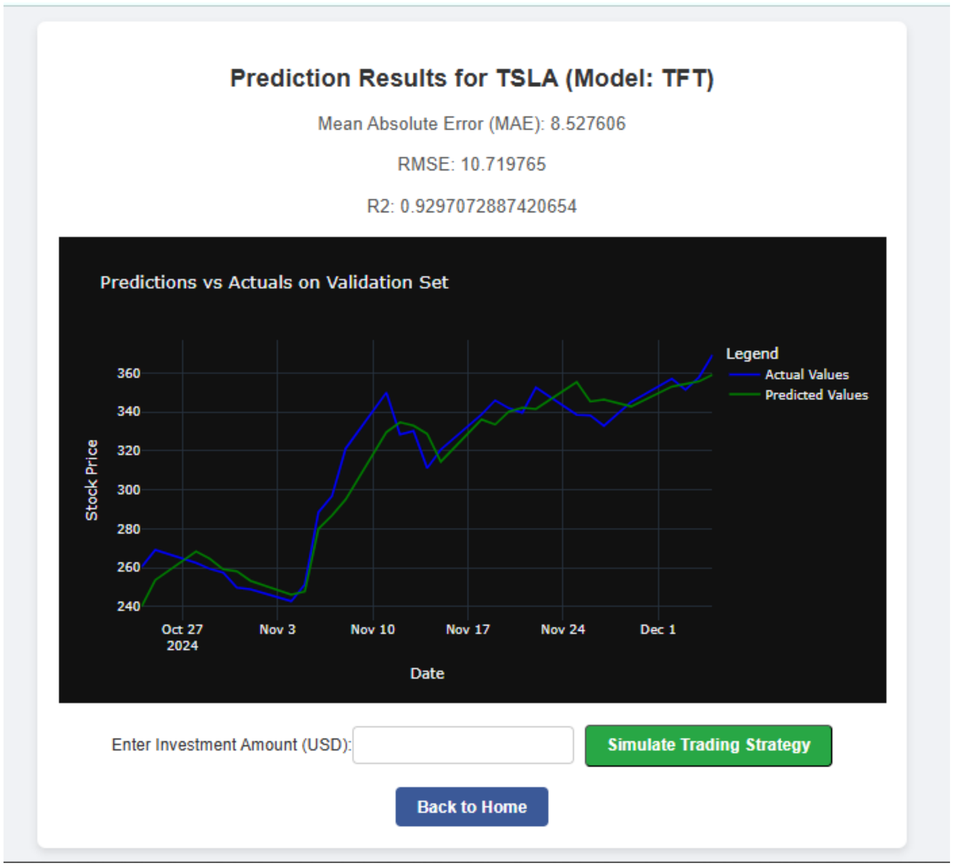
Fig 20: Error Page

3. Prediction Page for TFT Model (predict.html):

Overview: Displays prediction results for the Temporal Fusion Transformer model.

Actions:

- **Simulate Trading Strategy:** Generates a trading strategy based on the predicted prices.
- **Back to Home:** Returns to the homepage.



Prediction Results for TSLA (Model: TFT)

Mean Absolute Error (MAE): 8.527606

RMSE: 10.719765

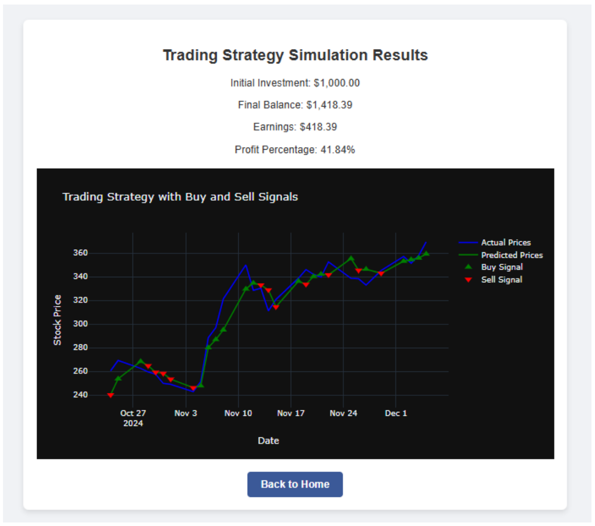
R2: 0.9297072887420654

Enter Investment Amount (USD):

Fig 21: Prediction Page for TFT Model

4. Trading Strategy Simulation Page (strategy.html):

Overview: Provides a simulated trading strategy based on predictions.



Trading Strategy Simulation Results

Initial Investment: $1,000.00

Final Balance: $1,418.39

Earnings: $418.39

Profit Percentage: 41.84%

Fig 22: Trading Strategy Simulation Page

Results Summary:

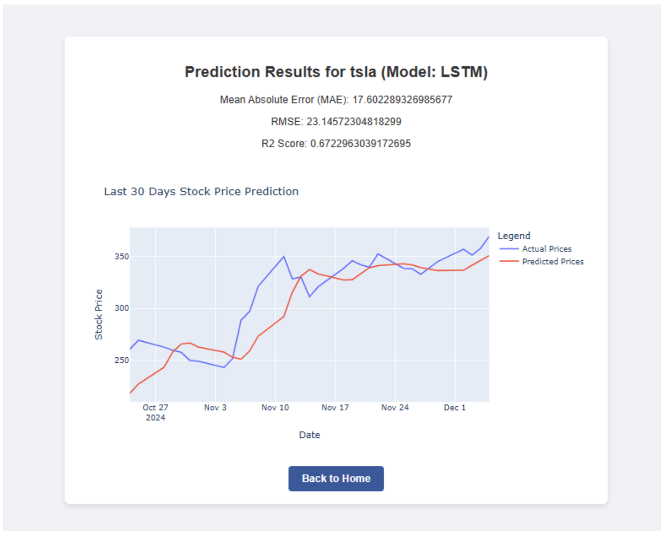
- Initial investment, final balance, earnings, and profit percentage.
- Stock price trends with buy and sell signals graph.

5. Prediction Page for LSTM Model (predict_lstm.html):

Overview: Similar to predict.html but tailored for LSTM predictions.

Key Elements:

- Displays metrics (MAE, RMSE, R²) specific to the LSTM model.
- Includes a graph comparing actual and predicted stock prices over the last 30 days.



Prediction Results for tsla (Model: LSTM)

Mean Absolute Error (MAE): 17.602289326985677

RMSE: 23.14572304818299

R2 Score: 0.6722963039172695

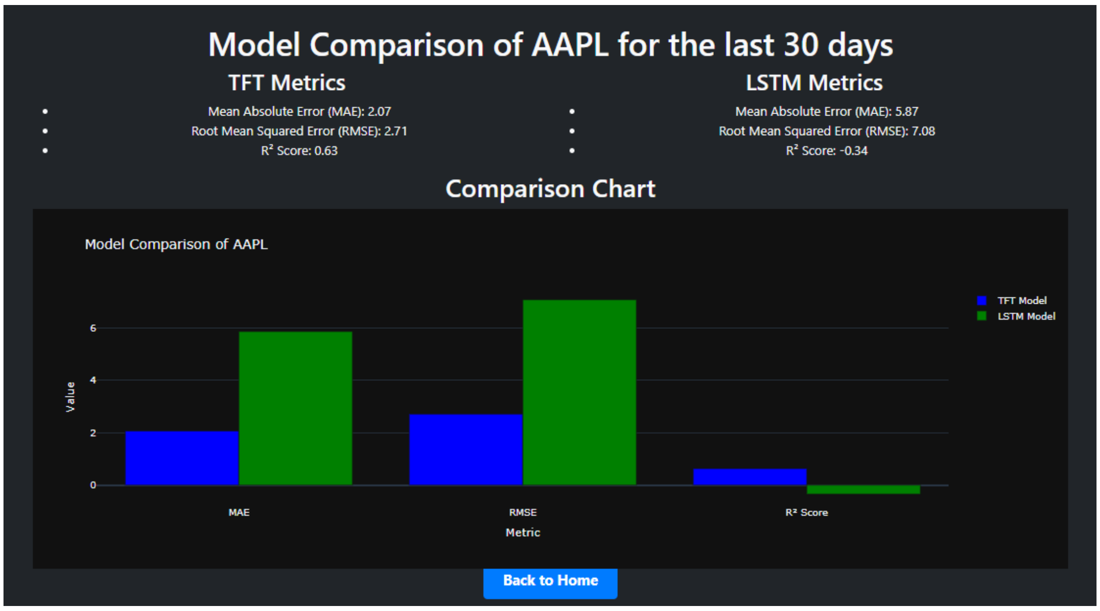
Fig 23: Prediction Page for LSTM Model

6. Model Comparison Page (compare_models.html):

Overview: Compares the performance of the TFT and LSTM models for a selected stock.

Key Elements:

- Displays MAE, RMSE, and R² scores for both models side-by-side.
- Visual representation of metric values for easier comparison.



Model Comparison of AAPL for the last 30 days

TFT Metrics

- Mean Absolute Error (MAE): 2.07
- Root Mean Squared Error (RMSE): 2.71
- R² Score: 0.63

LSTM Metrics

- Mean Absolute Error (MAE): 5.87
- Root Mean Squared Error (RMSE): 7.08
- R² Score: -0.34

Comparison Chart

Fig 23: Model Comparison Page

By following this guide, users can efficiently navigate the web application, leverage its analytical capabilities, and gain actionable insights for Tesla and Apple stocks.

Notes and Recommendations

- Ensure that API keys for Reddit, News API, and Ngrok are properly configured in the notebook.
- Verify that all input file paths are correctly specified and accessible.
- Always execute the project in **Google Colab Pro** to leverage GPU acceleration for faster processing.

References

[1] Yahoo! Finance. (n.d.). Stock Market Data API. Retrieved from [Yahoo Finance] (https://finance.yahoo.com/)

[2] Leetaru, K. (2013). The Global Database of Events, Language, and Tone (GDELT). Big Data, 1(1), 32–34.

[3] Gpreda. (n.d.). Collect WallStreetBets Data. Kaggle. Retrieved from https://www.kaggle.com/code/gpreda/collect-wallstreetbets-data/notebook.

[4] Injek0626. (n.d.). Reddit Stock-Related Posts. Kaggle Retrieved from https://www.kaggle.com/datasets/injek0626/reddit-stock-related-posts

[5] News API. (n.d.). Retrieve Live and Historical News Data. Retrieved from [https://newsapi.org] (https://newsapi.org)

[6] FinBERT: Sentiment Analysis for Financial Texts. (n.d.). Retrieved from https://huggingface.co/yiyanghkust/finbert-tone