Configuration Manual

MSc Research Project
Msc Data Analytics

Kushagra Airen
Student ID: x22200878

School of Computing
National College of Ireland

Supervisor: Paul Stynes, Musfira Jilani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kushagra Airen
Student ID:	x22200878
Programme:	MSc Data Analytics
Year:	2023 – 2024
Module:	MSc Research Project
Supervisor:	Paul Stynes
Submission Due Date:	12-08-2024
Project Title:	Configuration Manual
Word Count:	969

Signature: KUSHAGRA AIREN

Date: 12-08-2024

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

Kushagra Airen
X22200878

1 Introduction

This document will provides a step-by-step guide to implement the models build to solve the research question ”To what extent can a hybrid ARIMA-LSTM model accurately predict unemployment rates specifically during economic downturns?“. It discusses the hardware/software requirements, environment and system configuration needed for the research. The execution process of the three models applied that is ARIMA, LSTM and ARIMA-LSTM are detailed in this manual. The code is presented in two different files one for standalone ARIMA and LSTM models and the other for ARIMA-LSTM hybrid model.

2 System Configuration

2.1 Hardware Requirements

Below are the hardware requirements needed for this research.

- Machine: MacBook with Apple M2 chip or Windows 11 and above
- Processor: Apple M2 (8-core CPU, 8-core GPU) or Intel Core i7 (11th or 12th Gen)
- RAM: 8 GB (Minimum)
- Storage: 50 GB of free disk space
- Operating System: macOS Ventura (or newer) or Windows 11 Home or Pro

2.2 Software Requirements

Below are the software requirements needed for this research.

- Operating System: macOS Ventura or Windows 11
- Python Version: Python 3.11
- Integrated Development Environment (IDE): Jupyter Notebook (available via Anaconda)
- Package Manager: Anaconda Navigator
- Microsoft Excel (for data inspection)

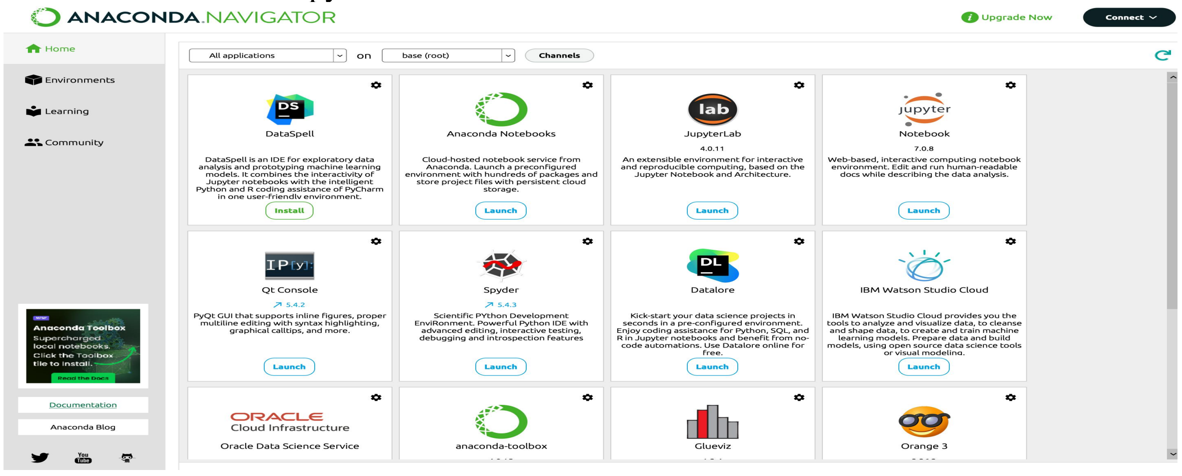
3 Environment Setup

This section will be explaining the set up of the environment for tools and libraries to be successfully installed

3.1 Installing Anaconda

- Download Anaconda from https://www.anaconda.com/download
- Open the downloaded “.pkg” file and follow the installation process and then add Anaconda to the system path

- Once the installation process in complete open ANACONDAS NAVIGATOR which will manage the enivornment.
- Launch Jupyter Notebook



DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendlv environment.

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

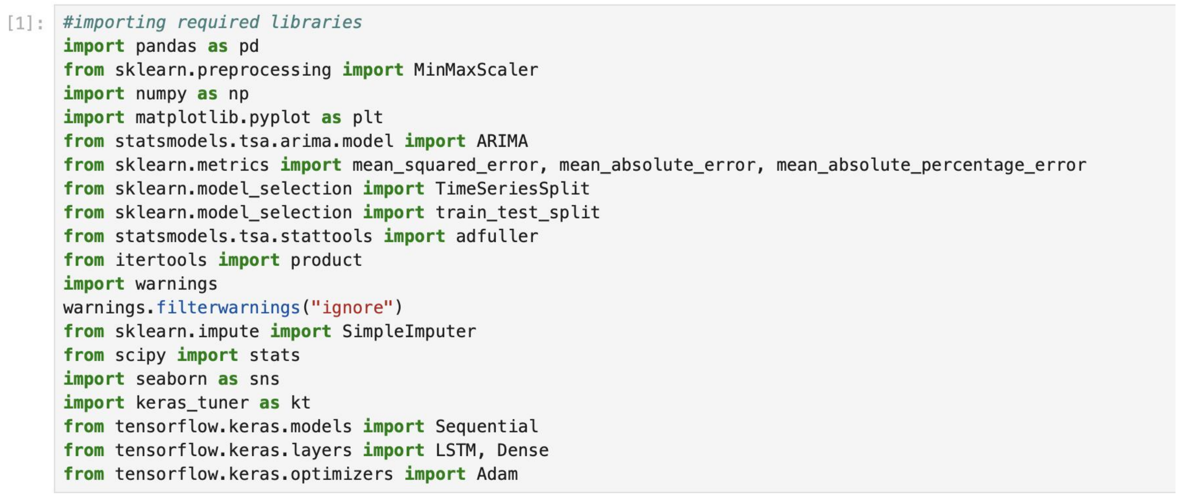
Figure 1: Anaconda Navigator

3.2 Open Jupyter Notebook

- Click on the launch button of jupyter notebook.
- Once the notebook is open, go where the file is saved
- After opening the research's python file with extention (.ipynb) run all the cells to execute the code

3.3 Installing Important Libraries

The important Libraries used in this search is below

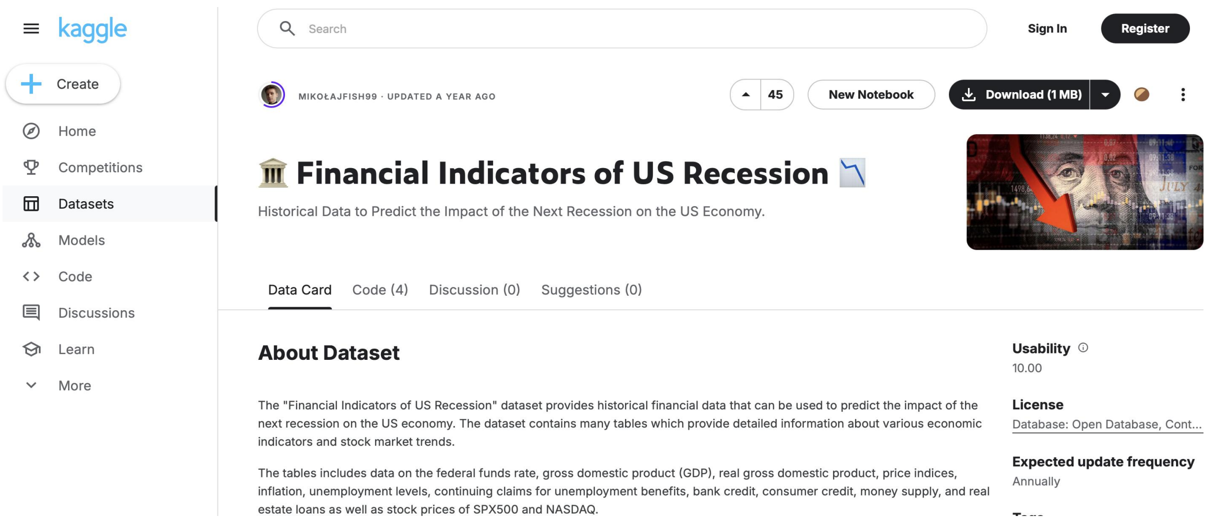


```
[1]: #importing required libraries
     import pandas as pd
     from sklearn.preprocessing import MinMaxScaler
     import numpy as np
     import matplotlib.pyplot as plt
     from statsmodels.tsa.arima.model import ARIMA
     from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
     from sklearn.model_selection import TimeSeriesSplit
     from sklearn.model_selection import train_test_split
     from statsmodels.tsa.stattools import adfuller
     from itertools import product
     import warnings
     warnings.filterwarnings("ignore")
     from sklearn.impute import SimpleImputer
     from scipy import stats
     import seaborn as sns
     import keras_tuner as kt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import LSTM, Dense
     from tensorflow.keras.optimizers import Adam
```

Figure 2: Importing required libraries

4 Data Gathering & Pre-processing

In this section the data collecting and the preprocessing's step is discussed for a smooth replication of the research project. The dataset is taken from Kaggle the link for the dataset is:



🏛 Financial Indicators of US Recession 📉

Historical Data to Predict the Impact of the Next Recession on the US Economy.

About Dataset

The "Financial Indicators of US Recession" dataset provides historical financial data that can be used to predict the impact of the next recession on the US economy. The dataset contains many tables which provide detailed information about various economic indicators and stock market trends.

The tables includes data on the federal funds rate, gross domestic product (GDP), real gross domestic product, price indices, inflation, unemployment levels, continuing claims for unemployment benefits, bank credit, consumer credit, money supply, and real estate loans as well as stock prices of SPX500 and NASDAQ.

Usability
10.00

License
Database: Open Database, Cont...

Expected update frequency
Annually

Figure 2: Financial Indicators Kaggle Dataset

4.1 Data Collection

A) Data Sources:

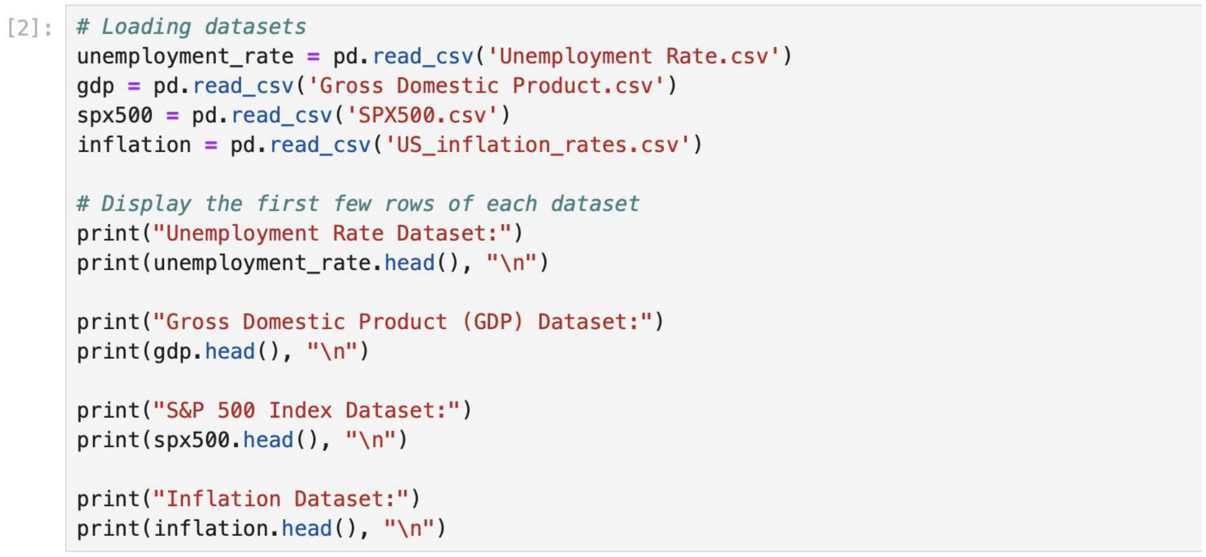
- **Unemployment Data:** Unemployment rates of US.
- **GDP Data:** GDP data of US.
- **Inflation Data:** Inflation rates of US.
- **Stock Market Prices Data:** S&P 500 stock dataset of US.

B) Downloading the datasets

- Download the dataset.
- Select the above economic indicators
- Save in the preferred directory on the local machine.

C) Loading the dataset in the notebook

- Use the following python code



```
[2]: # Loading datasets
unemployment_rate = pd.read_csv('Unemployment Rate.csv')
gdp = pd.read_csv('Gross Domestic Product.csv')
spx500 = pd.read_csv('SPX500.csv')
inflation = pd.read_csv('US_inflation_rates.csv')

# Display the first few rows of each dataset
print("Unemployment Rate Dataset:")
print(unemployment_rate.head(), "\n")

print("Gross Domestic Product (GDP) Dataset:")
print(gdp.head(), "\n")

print("S&P 500 Index Dataset:")
print(spx500.head(), "\n")

print("Inflation Dataset:")
print(inflation.head(), "\n")
```

Figure 3: Loading datasets

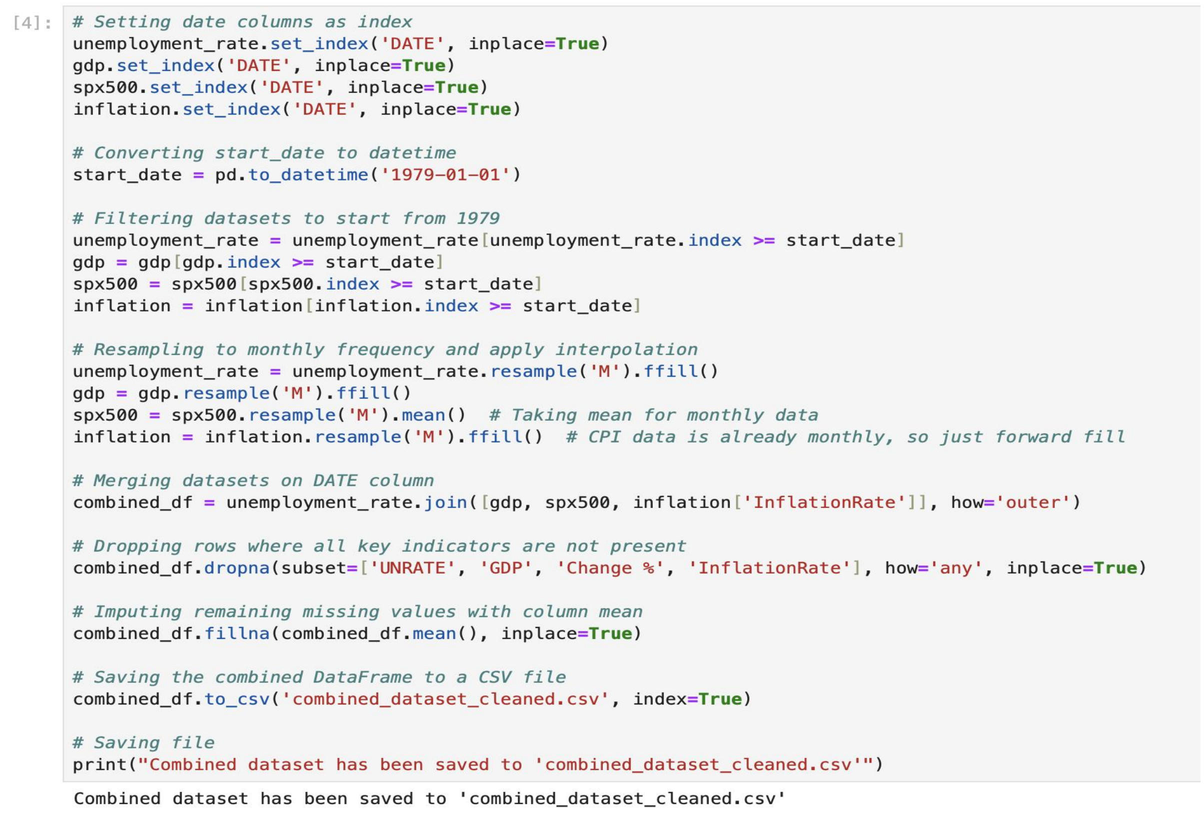
4.2 Data Preprocessing

The data preprocessing can be done by following steps

A) Data cleaning:

- Date column in all the datasets were converted into a standard datetime format
- Monthly inflation rate was calculated using the percentage change method in the Consumer Price Index (CPI) and then resampled into monthly frequency.
- In the S&P 500 Index dataset, columns Price, Open, High, Low, and Volume were dropped and only Change % is retained.
- Interpolation method was applied on the GDP data to convert quarterly data into monthly.
- The datasets were merged using date column and were checked for any null or missing values.

The final merged dataset contains 518 rows and was from 31st December 1979 to 1st January 2023. Below is the code used to implement the above steps:



```
[4]: # Setting date columns as index
unemployment_rate.set_index('DATE', inplace=True)
gdp.set_index('DATE', inplace=True)
spx500.set_index('DATE', inplace=True)
inflation.set_index('DATE', inplace=True)

# Converting start_date to datetime
start_date = pd.to_datetime('1979-01-01')

# Filtering datasets to start from 1979
unemployment_rate = unemployment_rate[unemployment_rate.index >= start_date]
gdp = gdp[gdp.index >= start_date]
spx500 = spx500[spx500.index >= start_date]
inflation = inflation[inflation.index >= start_date]

# Resampling to monthly frequency and apply interpolation
unemployment_rate = unemployment_rate.resample('M').ffill()
gdp = gdp.resample('M').ffill()
spx500 = spx500.resample('M').mean()  # Taking mean for monthly data
inflation = inflation.resample('M').ffill()  # CPI data is already monthly, so just forward fill

# Merging datasets on DATE column
combined_df = unemployment_rate.join([gdp, spx500, inflation['InflationRate']], how='outer')

# Dropping rows where all key indicators are not present
combined_df.dropna(subset=['UNRATE', 'GDP', 'Change %', 'InflationRate'], how='any', inplace=True)

# Imputing remaining missing values with column mean
combined_df.fillna(combined_df.mean(), inplace=True)

# Saving the combined DataFrame to a CSV file
combined_df.to_csv('combined_dataset_cleaned.csv', index=True)

# Saving file
print("Combined dataset has been saved to 'combined_dataset_cleaned.csv'")

Combined dataset has been saved to 'combined_dataset_cleaned.csv'
```

Figure 4: Merged dataset of economic inidicators

Plotting the unemployment rate of the merged dataset

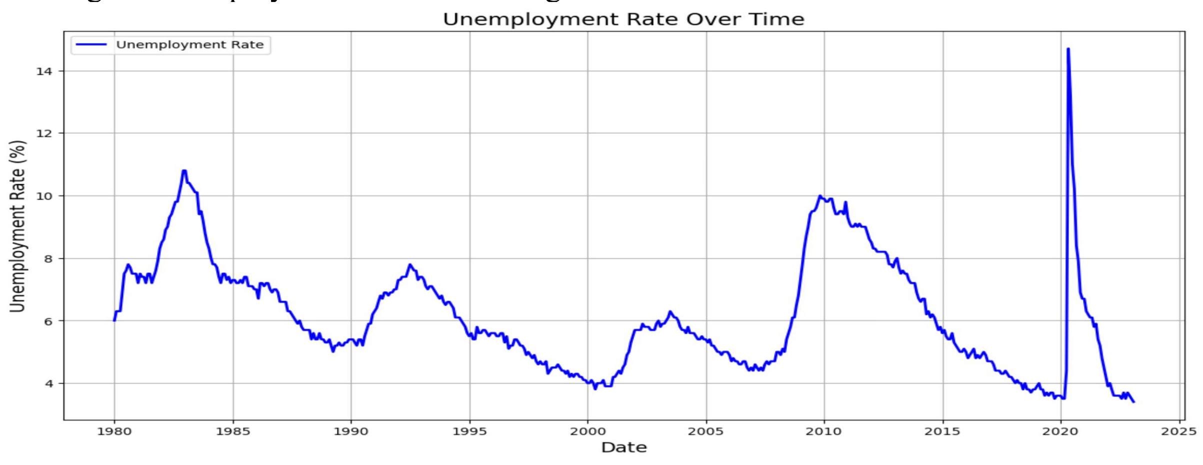
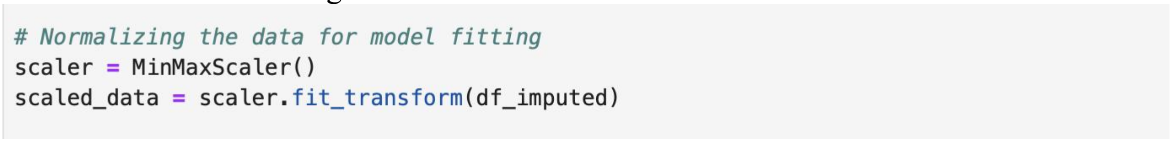


Figure 5: Unemployment Rate over Time

B) Data Normalizaion: Normalising the data is done by using the following steps:

- Use Min-Max scaling

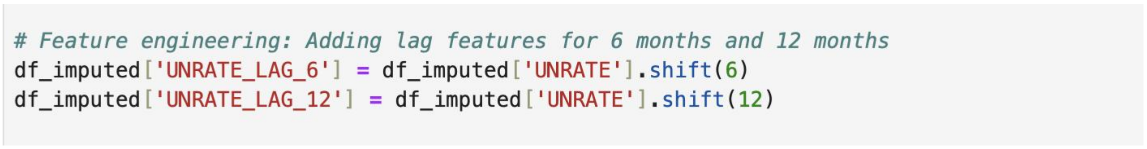


```
# Normalizing the data for model fitting
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df_imputed)
```

Figure 6: Normalizing the dataset

C) Feauture Engineering

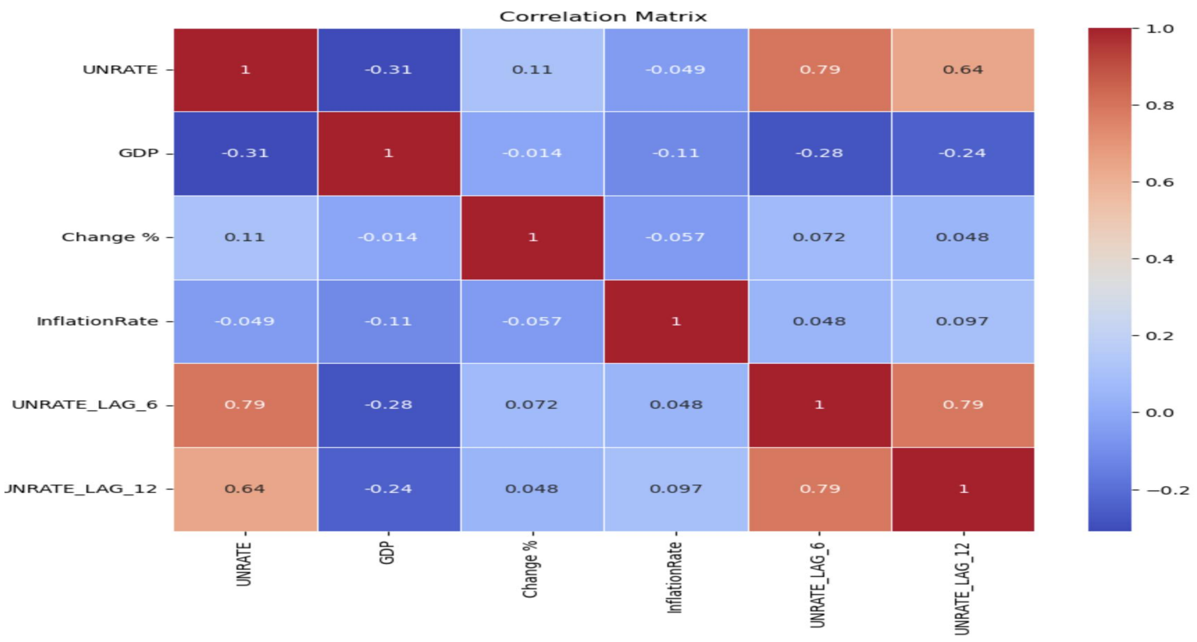
- Create Lag Features of 6 and 12 months of Unemployment Rate



```
# Feature engineering: Adding lag features for 6 months and 12 months
df_imputed['UNRATE_LAG_6'] = df_imputed['UNRATE'].shift(6)
df_imputed['UNRATE_LAG_12'] = df_imputed['UNRATE'].shift(12)
```

Figure 7: Creating Lag features

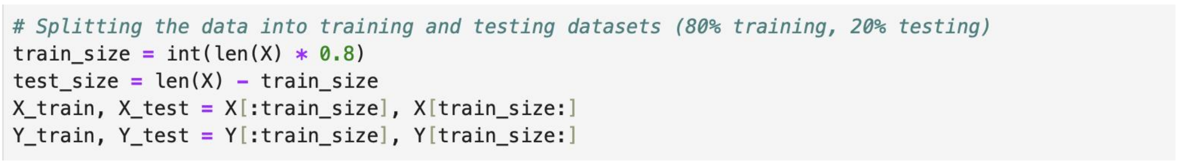
- Correlation analysis is done on economic indicators and created lag variables.



	UNRATE	GDP	Change %	InflationRate	UNRATE_LAG_6	UNRATE_LAG_12
UNRATE	1	-0.31	0.11	-0.049	0.79	0.64
GDP	-0.31	1	-0.014	-0.11	-0.28	-0.24
Change %	0.11	-0.014	1	-0.057	0.072	0.048
InflationRate	-0.049	-0.11	-0.057	1	0.048	0.097
UNRATE_LAG_6	0.79	-0.28	0.072	0.048	1	0.79
JNRATE_LAG_12	0.64	-0.24	0.048	0.097	0.79	1

Figure 8: Correlation Analysis

- Splitting dataset into training and testing set



```
# Splitting the data into training and testing datasets (80% training, 20% testing)
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]
```

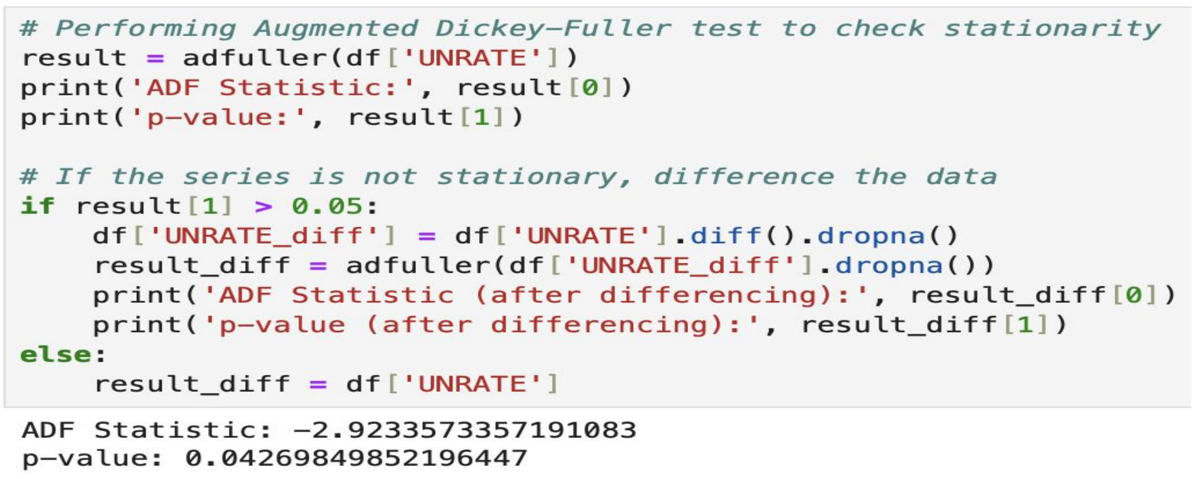
Figure 9: Splitting dataset

5 Model Configuration

This section will give steps for configuring the models used in this research. Set up of ARIMA, LSTM and hybrid ARIMA-LSTM model is discussed. The hyperparameter tuning is also discussed in this section.

5.1 ARIMA Model Setup

- Stationary test : Augmented Dickey-Fuller (ADF) test is used to check if the series is stationary

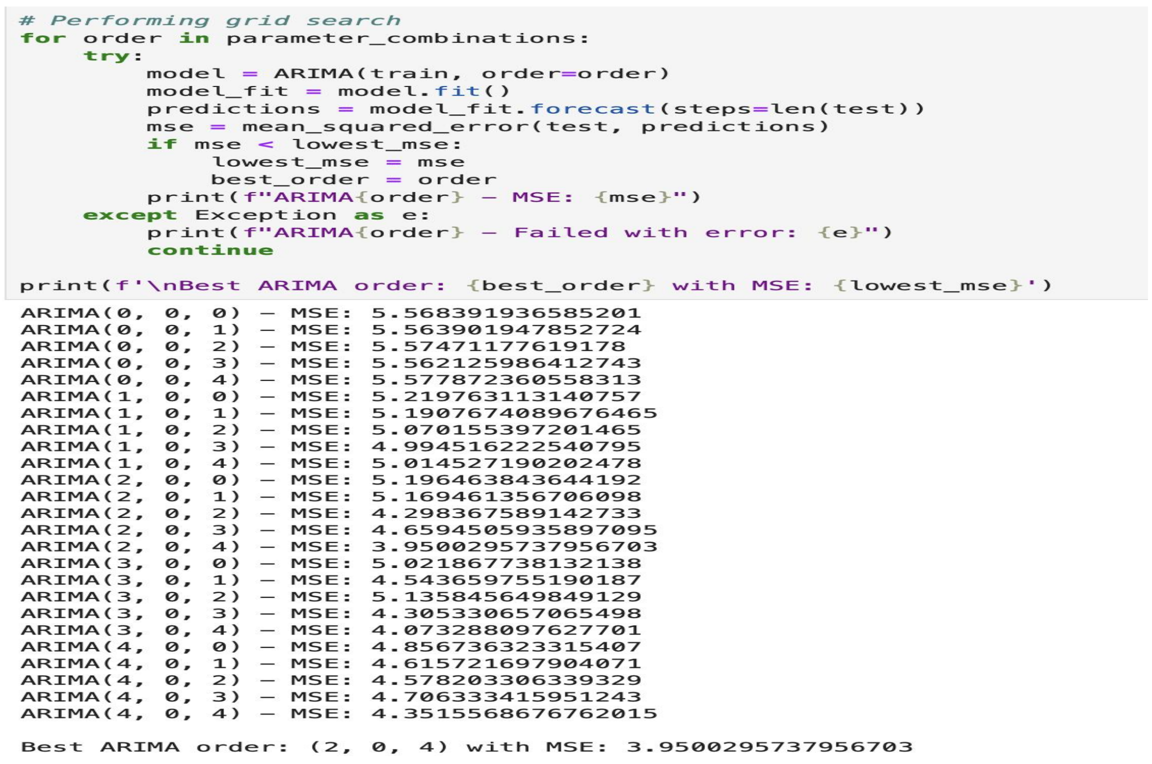


```
# Performing Augmented Dickey-Fuller test to check stationarity
result = adfuller(df['UNRATE'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])

# If the series is not stationary, difference the data
if result[1] > 0.05:
    df['UNRATE_diff'] = df['UNRATE'].diff().dropna()
    result_diff = adfuller(df['UNRATE_diff'].dropna())
    print('ADF Statistic (after differencing):', result_diff[0])
    print('p-value (after differencing):', result_diff[1])
else:
    result_diff = df['UNRATE']
```

```
ADF Statistic: -2.9233573357191083
p-value: 0.04269849852196447
```

Figure 10: AD-Fuller test

- Implementing grid search to find the best parameters



```
# Performing grid search
for order in parameter_combinations:
    try:
        model = ARIMA(train, order=order)
        model_fit = model.fit()
        predictions = model_fit.forecast(steps=len(test))
        mse = mean_squared_error(test, predictions)
        if mse < lowest_mse:
            lowest_mse = mse
            best_order = order
        print(f"ARIMA{order} - MSE: {mse}")
    except Exception as e:
        print(f"ARIMA{order} - Failed with error: {e}")
        continue

print(f'\nBest ARIMA order: {best_order} with MSE: {lowest_mse}')
```

```
ARIMA(0, 0, 0) - MSE: 5.568391936585201
ARIMA(0, 0, 1) - MSE: 5.563901947852724
ARIMA(0, 0, 2) - MSE: 5.57471177619178
ARIMA(0, 0, 3) - MSE: 5.562125986412743
ARIMA(0, 0, 4) - MSE: 5.577872360558313
ARIMA(1, 0, 0) - MSE: 5.219763113140757
ARIMA(1, 0, 1) - MSE: 5.1907674089676465
ARIMA(1, 0, 2) - MSE: 5.070155397201465
ARIMA(1, 0, 3) - MSE: 4.994516222540795
ARIMA(1, 0, 4) - MSE: 5.014527190202478
ARIMA(2, 0, 0) - MSE: 5.196463843644192
ARIMA(2, 0, 1) - MSE: 5.169461356706098
ARIMA(2, 0, 2) - MSE: 4.298367589142733
ARIMA(2, 0, 3) - MSE: 4.6594505935897095
ARIMA(2, 0, 4) - MSE: 3.9500295737956703
ARIMA(3, 0, 0) - MSE: 5.021867738132138
ARIMA(3, 0, 1) - MSE: 4.543659755190187
ARIMA(3, 0, 2) - MSE: 5.135845649849129
ARIMA(3, 0, 3) - MSE: 4.305330657065498
ARIMA(3, 0, 4) - MSE: 4.073288097627701
ARIMA(4, 0, 0) - MSE: 4.856736323315407
ARIMA(4, 0, 1) - MSE: 4.615721697904071
ARIMA(4, 0, 2) - MSE: 4.578203306339329
ARIMA(4, 0, 3) - MSE: 4.706333415951243
ARIMA(4, 0, 4) - MSE: 4.3515568676762015

Best ARIMA order: (2, 0, 4) with MSE: 3.9500295737956703
```

Figure 11: Finding best parameters

- Model Fitting: ‘statsmodels’ is used to fit the data

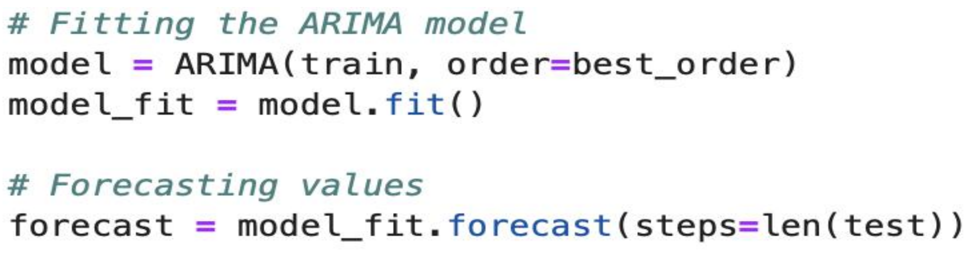

```
# Fitting the ARIMA model
model = ARIMA(train, order=best_order)
model_fit = model.fit()

# Forecasting values
forecast = model_fit.forecast(steps=len(test))
```

Figure 12: Fitting ARIMA model

5.2 LSTM Model Setup

- Dataset preparation is done for LSTM using a step size of 10.

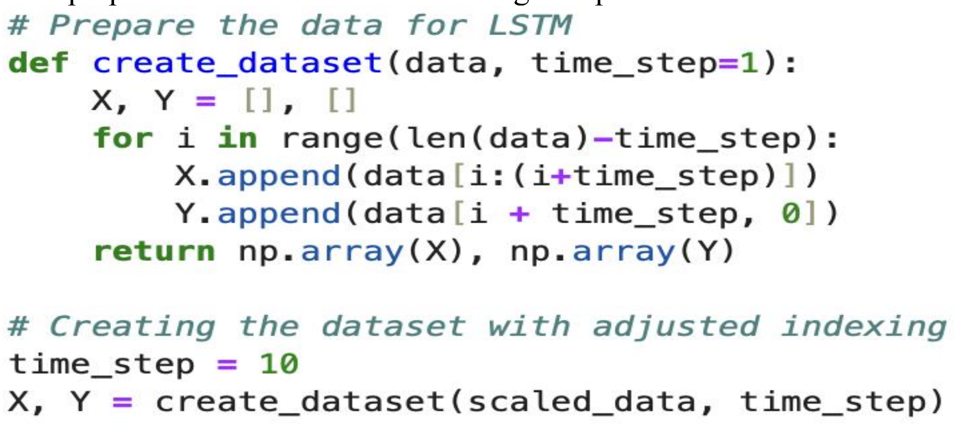

```
# Prepare the data for LSTM
def create_dataset(data, time_step=1):
    X, Y = [], []
    for i in range(len(data)-time_step):
        X.append(data[i:(i+time_step)])
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)

# Creating the dataset with adjusted indexing
time_step = 10
X, Y = create_dataset(scaled_data, time_step)
```

Figure 13: Dataset is created using time step

- Defining the LSTM model for Hyperparameter tuning

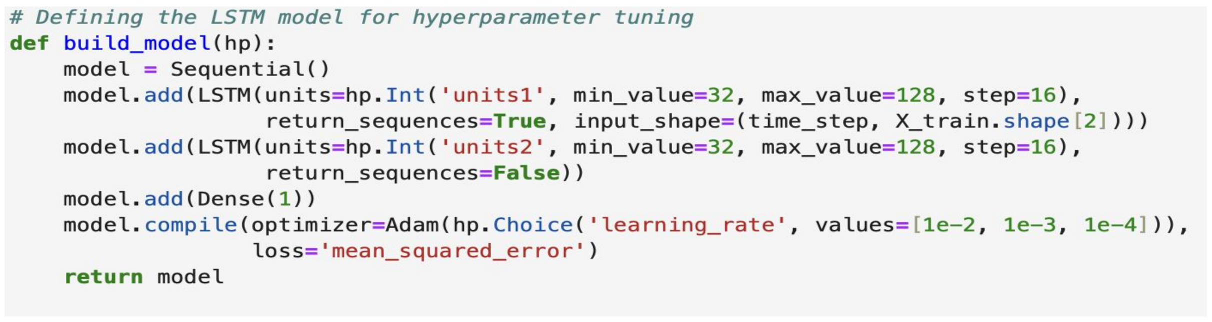

```
# Defining the LSTM model for hyperparameter tuning
def build_model(hp):
    model = Sequential()
    model.add(LSTM(units=hp.Int('units1', min_value=32, max_value=128, step=16),
                   return_sequences=True, input_shape=(time_step, X_train.shape[2])))
    model.add(LSTM(units=hp.Int('units2', min_value=32, max_value=128, step=16),
                   return_sequences=False))
    model.add(Dense(1))
    model.compile(optimizer=Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])),
                  loss='mean_squared_error')
    return model
```

Figure 14: Building the LSTM model

- Performing the hyperparameter search and printing the optimal parameters

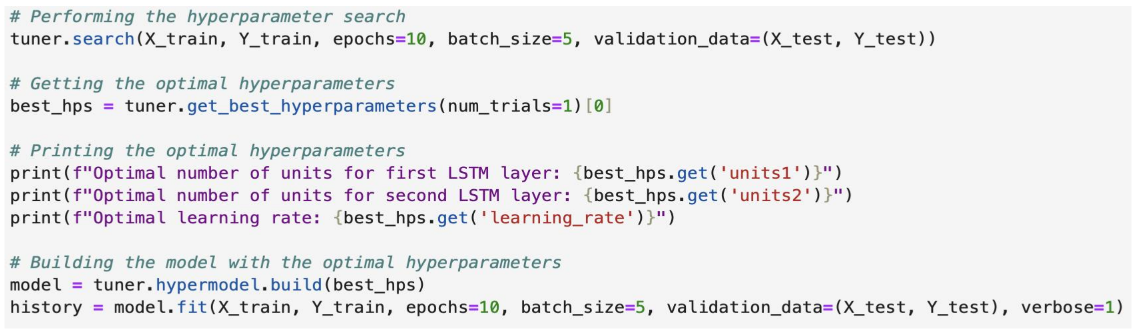
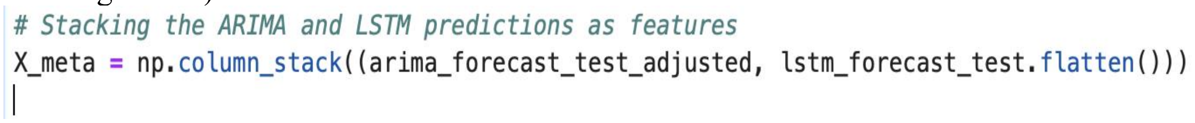

```
# Performing the hyperparameter search
tuner.search(X_train, Y_train, epochs=10, batch_size=5, validation_data=(X_test, Y_test))

# Getting the optimal hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Printing the optimal hyperparameters
print(f"Optimal number of units for first LSTM layer: {best_hps.get('units1')}")
print(f"Optimal number of units for second LSTM layer: {best_hps.get('units2')}")
print(f"Optimal learning rate: {best_hps.get('learning_rate')}")

# Building the model with the optimal hyperparameters
model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, Y_train, epochs=10, batch_size=5, validation_data=(X_test, Y_test), verbose=1)
```

Figure 15: Hyperparameter search

5.3 Hybrid ARIMA-LSTM Model Setup

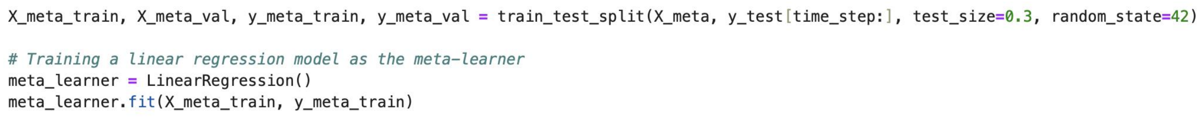
- Applying ARIMA model on the linear components of the time series data
- Applying LSTM model on the time series data to capture non-linear patterns.
- Combining the ARIMA and LSTM predictions as features for the meta-learner (Linear Regression).



```
# Stacking the ARIMA and LSTM predictions as features
X_meta = np.column_stack((arima_forecast_test_adjusted, lstm_forecast_test.flatten()))
```

Figure 16: Stacking the ARIMA & LSTM Forecast

- Use the Linear Regression model to combine ARIMA and LSTM predictions for the final forecast.



```
X_meta_train, X_meta_val, y_meta_train, y_meta_val = train_test_split(X_meta, y_test[time_step:], test_size=0.3, random_state=42)

# Training a linear regression model as the meta-learner
meta_learner = LinearRegression()
meta_learner.fit(X_meta_train, y_meta_train)
```

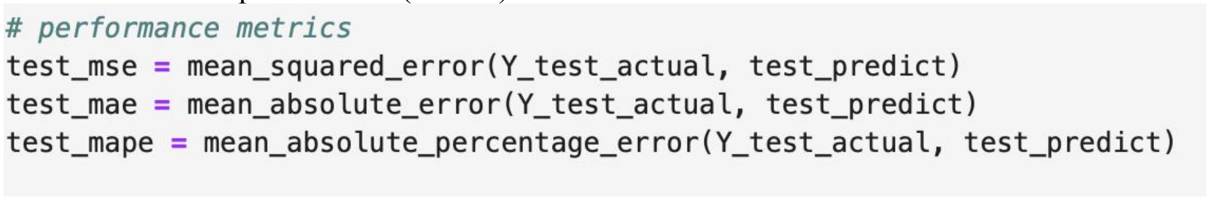
Figure 17: Linear regression applied on the stacked dataset

6 Model Evaluation

6.1 Performance Metrics

The performance metrics are carried out using below measures.

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)



```
# performance metrics
test_mse = mean_squared_error(Y_test_actual, test_predict)
test_mae = mean_absolute_error(Y_test_actual, test_predict)
test_mape = mean_absolute_percentage_error(Y_test_actual, test_predict)
```

Figure 18: Calculating performance metrics on predictions

6.2 Forecasting and saving predictions

The Actual vs Predicted Values were plotted along with a CSV file containing the actual and predicted values

- ARIMA forecast

```python
# Plotting the forecasted values and the actual values
plt.figure(figsize=(10, 6))
plt.plot(train.index, train, label='Training Data')
plt.plot(test.index, test, label='Actual Test Data', color='blue')
plt.plot(forecast_series, label='ARIMA Test Predictions', color='red')
plt.title('ARIMA Model Forecast')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.legend()
plt.grid(True)
plt.show()

# Saving the results to a CSV file
results_df = pd.DataFrame({
    'Date': forecast_index,
    'Actual': test,
    'Forecasted': forecast_series
})

results_df.to_csv('arima_forecast_results.csv', index=False)
print('ARIMA forecast results saved to arima_forecast_results.csv')
```

Figure 19: Plotting and saving results of ARIMA

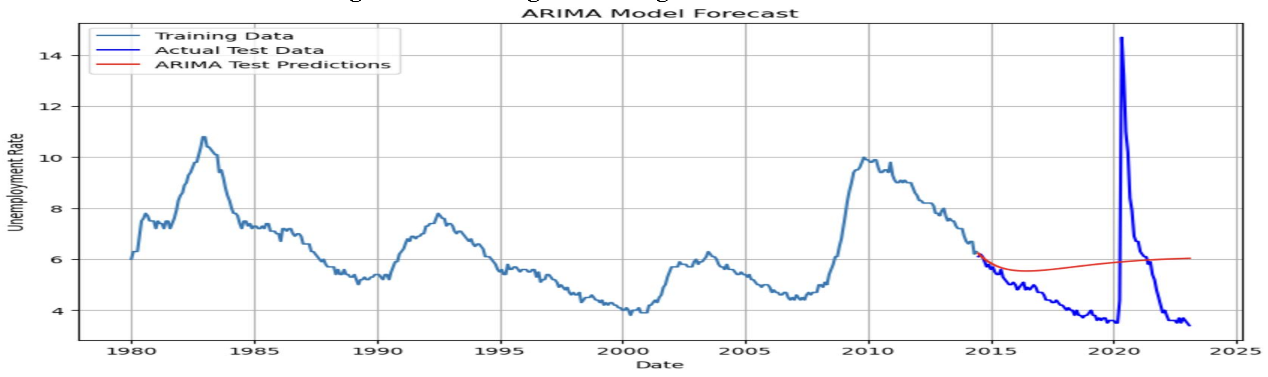


Figure 20: ARIMA model forecast

- LSTM forecast

```python
# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(df_imputed.index[time_step:train_size+time_step], Y_train_actual, label='Actual Train Data')
plt.plot(df_imputed.index[train_size+time_step:train_size+test_size+time_step], Y_test_actual, label='Actual Test Data')
plt.plot(df_imputed.index[train_size+time_step:train_size+test_size+time_step], test_predict, label='LSTM Test Predictions', color='red')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.title('LSTM Model Forecast')
plt.legend()
plt.grid(True)
plt.show()

# Saving the results to a CSV file
results_lstm_df = pd.DataFrame({
    'Date': df_imputed.index[train_size+time_step:train_size+test_size+time_step],
    'Actual': Y_test_actual.flatten(),
    'Predicted': test_predict.flatten()
})

results_lstm_df.to_csv('actual_vs_predicted_lstm.csv', index=False)
```

Figure 21: Plotting and saving results of LSTM

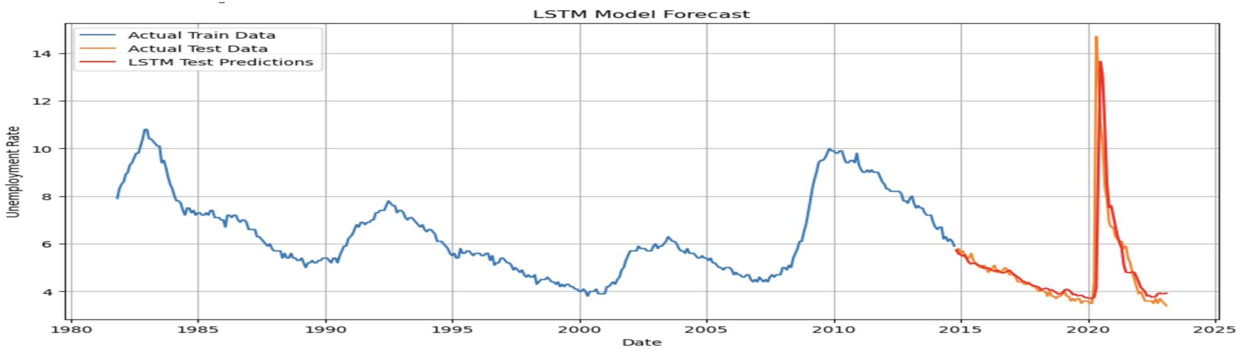


Figure 22: LSTM model forecast

- Hybrid ARIMA-LSTM forecast

```
# Plotting the hybrid model results for the test set
plt.figure(figsize=(12, 6))
plt.plot(test_data.index[time_step:], y_test[time_step:], label='Actual Test Data', color='orange')
plt.plot(test_data.index[time_step:], hybrid_test_predictions, label='Hybrid Test Predictions', color='red')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.title('Hybrid ARIMA-LSTM Model Forecast')
plt.legend()
plt.grid(True)
plt.show()

# Saving the results to a CSV file
results_hybrid_df = pd.DataFrame({
    'Date': test_data.index[time_step:],
    'Actual': y_test[time_step:],
    'ARIMA_Predicted': arima_forecast_test_adjusted,
    'LSTM_Predicted': lstm_forecast_test.flatten(),
    'Hybrid_Predicted': hybrid_test_predictions
})

results_hybrid_df.to_csv('actual_vs_predicted_hybrid.csv', index=False)
```

Figure 21: Plotting and saving results of ARIMA-LSTM

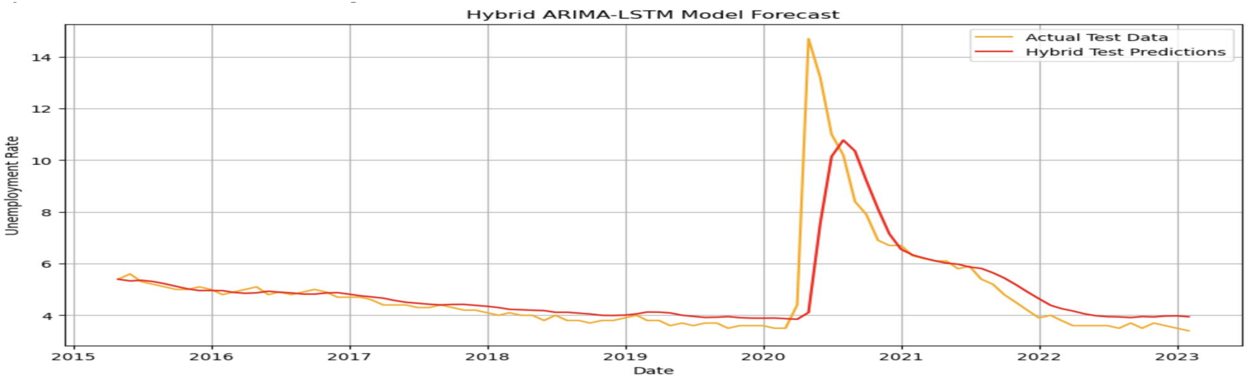


Figure 22: ARIMA-LSTM model forecast