Configuration Manual

MSc Research Project
Data Analytics

RAMANAN LOGANATHAN
Student ID: 21202702

School of Computing
National College of Ireland

Supervisor: FURQAN RUSTAM

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	RAMANAN LOGANATHAN
Student ID:	21202702
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	FURQAN RUSTAM
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	1077
Page Count:	5

Signature:	
Date:	18th September 2023

Configuration Manual

RAMANAN LOGANATHAN
21202702

1 Introduction

The configuration manual outlines the sequential guide for executing the time series models associated with the research project. The project involves the implementation and evaluation of four different time series models, namely RNN (Recurrent Neural Network), ARIMA (AutoRegressive Integrated Moving Average), SARIMA (Seasonal AutoRegressive Integrated Moving Average), and LSTM (Long Short-Term Memory).

2 System Configuration

2.1 Software Requirements

The research work was executed utilising Google Colab, an open-source framework for AI/ML projects in the Google ecosystem. This setting is powered by a Python module. Installing each of these packages is necessary before the project can be built.

2.2 Hardware specifications

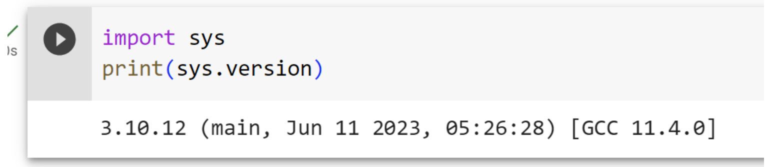
- System Name: suryas_zenbook
- Processor: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
- Installed RAM: 16.0 GB (15.7 GB usable)
- Storage Size: 1TB SSD (109,951,162,7776 bytes)
- OS type: 64-bit operating system, x64-based processor

3 Installation and Environment Setup

- **Python**

This project made use of a Python package. Since the majority of Deep Learning and Machine Learning Projects are supported by its numerous built-in libraries. With a variety of plots, it makes developing and analysing models easier. Since python's latest version is already installed in google colab, no need to install it again.

- **Google Colab**



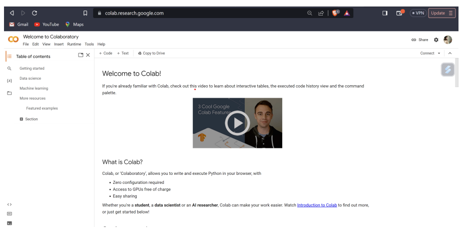
```
import sys
print(sys.version)
```

```
3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
```

Figure 1: Latest version in Google colab

Google offers a Jupyter notebook environment on the cloud called Colaboratory, also known as Google Colab. As It provides a practical platform for research, machine learning, data analysis, and coding tasks and Colab serves as a popular option for students, academics, and professionals because it avoids the need for complicated settings by allowing Python code to execute easily in a browser.

One of Colab's key benefits is its interface with Google Drive, which enables simple storing, sharing, and notebook collaboration. Additionally, it offers open access to GPUs and TPUs, facilitating the quicker completion of resource-intensive operations. Because Colab notebooks are interactive, data can be visualised and analysed in real-time, providing immediate insights. This can accessed from [1]as shown in the Figure 2 .



Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.

What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to find out more, or just get started below!

Figure 2: Google Colab's user interface

4 Data Collection

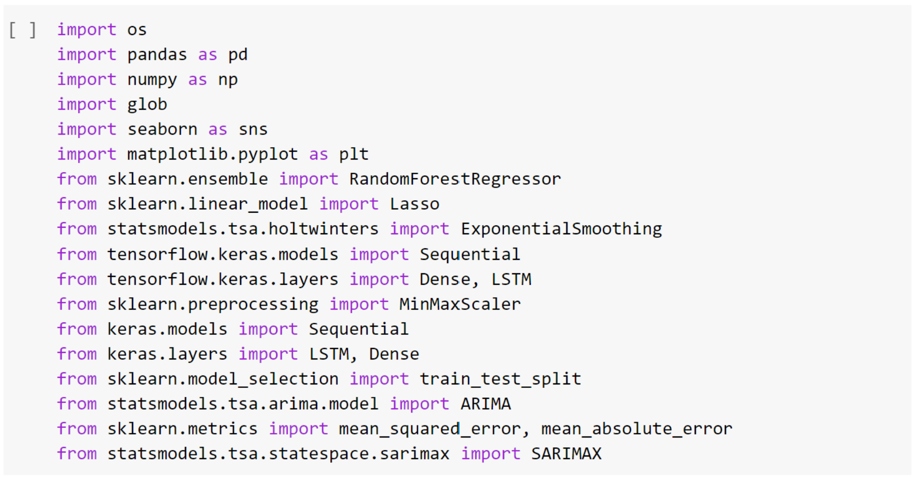
The quarterly data from the Ministry of Statistics and Implementation Programme [2] had the quarterly GDP data of different sectors. This dataset had Gross Domestic Product (GDP) figures for various industries across different quarters for a span of 15 years.

5 Implementation

5.1 Importing Libraries

The sections below demonstrate how to implement the project using Python. Run them as instructed, step by step. Preprocessing the provided data is the first step before we start the implementation. The libraries required for startup are shown in the picture below 3. ,

[1]https://colab.research.google.com/
[2]https://mospi.gov.in/data

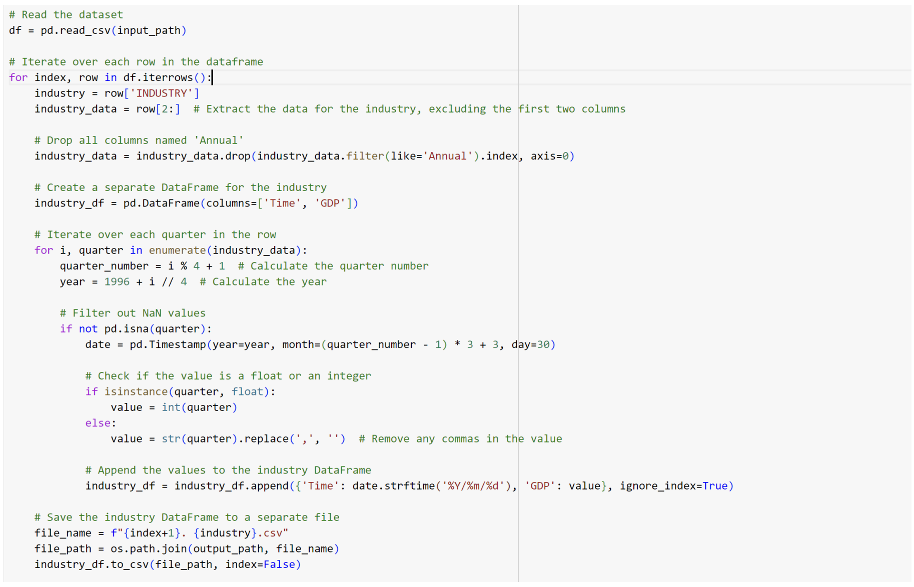


```
[ ] import os
    import pandas as pd
    import numpy as np
    import glob
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.linear_model import Lasso
    from statsmodels.tsa.holtwinters import ExponentialSmoothing
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, LSTM
    from sklearn.preprocessing import MinMaxScaler
    from keras.models import Sequential
    from keras.layers import LSTM, Dense
    from sklearn.model_selection import train_test_split
    from statsmodels.tsa.arima.model import ARIMA
    from sklearn.metrics import mean_squared_error, mean_absolute_error
    from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Figure 3: Importing Required Libraries

5.2 Data Preprocessing

The industry column specifies the particular industry or sector to which the GDP values pertain. This data is further splitted into 9 different CSV files as it represents 9 different GDP data.To do this operation the code shown in figure 4 need to be executed.

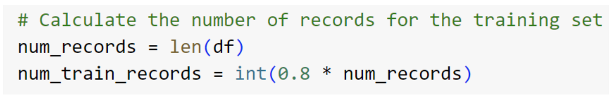


```
# Read the dataset
df = pd.read_csv(input_path)

# Iterate over each row in the dataframe
for index, row in df.iterrows():
    industry = row['INDUSTRY']
    industry_data = row[2:]  # Extract the data for the industry, excluding the first two columns

    # Drop all columns named 'Annual'
    industry_data = industry_data.drop(industry_data.filter(like='Annual').index, axis=0)

    # Create a separate DataFrame for the industry
    industry_df = pd.DataFrame(columns=['Time', 'GDP'])

    # Iterate over each quarter in the row
    for i, quarter in enumerate(industry_data):
        quarter_number = i % 4 + 1  # Calculate the quarter number
        year = 1996 + i // 4  # Calculate the year

        # Filter out NaN values
        if not pd.isna(quarter):
            date = pd.Timestamp(year=year, month=(quarter_number - 1) * 3 + 3, day=30)

            # Check if the value is a float or an integer
            if isinstance(quarter, float):
                value = int(quarter)
            else:
                value = str(quarter).replace(',', '')  # Remove any commas in the value

            # Append the values to the industry DataFrame
            industry_df = industry_df.append({'Time': date.strftime('%Y/%m/%d'), 'GDP': value}, ignore_index=True)

    # Save the industry DataFrame to a separate file
    file_name = f"{index+1}. {industry}.csv"
    file_path = os.path.join(output_path, file_name)
    industry_df.to_csv(file_path, index=False)
```

Figure 4: Data Preprocessing

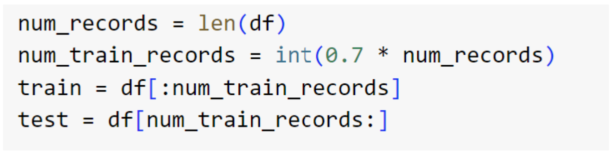
5.3 Splitting of Data

The data has been splitted into 80-20 ratio for train and test respectively for ARIMA model as shown in the figure 5 and for RNN, LSTM and SARIMA the split was 70-30 as shown in the below figure 6 7 8 respectively.



```
# Calculate the number of records for the training set
num_records = len(df)
num_train_records = int(0.8 * num_records)
```

Figure 5: Train and Test Split for ARIMA



```
num_records = len(df)
num_train_records = int(0.7 * num_records)
train = df[:num_train_records]
test = df[num_train_records:]
```

Figure 6: Train and Test Split for RNN

5.4 Implementing ARIMA Model

The ARIMA model was implemented by executing the code in the below picture 9. All the necessary steps for building the model has been included in the code. This code gives a result with actual GDP, predicted GDP, difference between actual and predicted GDP values and the difference percentage.

5.5 Implementing SARIMA Model

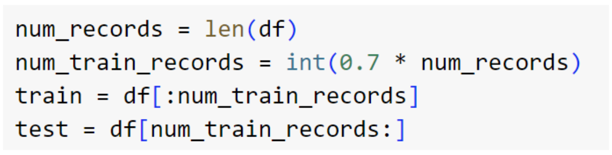
The SARIMA model was implemented by executing the code in the below picture 10. All the necessary steps for building the model has been included in the code. This code gives a result with actual GDP, predicted GDP, difference between actual and predicted GDP values and the difference percentage.

5.6 Implementing LSTM Model

The LSTM model was implemented by executing the code in the below picture 11. All the necessary steps for building the model has been included in the code. This code gives a result with actual GDP, predicted GDP, difference between actual and predicted GDP values and the difference percentage.

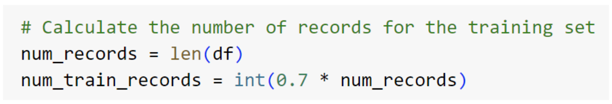
5.7 Implementing RNN Model

The RNN model was implemented by executing the code in the below picture 12. All the necessary steps for building the model has been included in the code. This code gives a result with actual GDP, predicted GDP, difference between actual and predicted GDP values and the difference percentage.



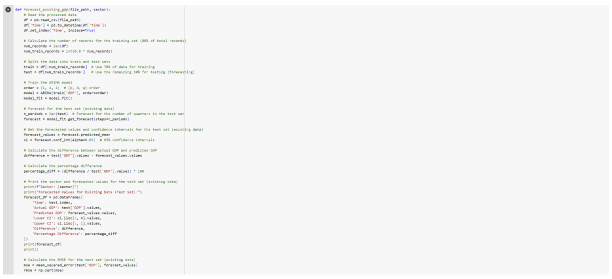
```python
num_records = len(df)
num_train_records = int(0.7 * num_records)
train = df[:num_train_records]
test = df[num_train_records:]
```

Figure 7: Train and Test Split for LSTM



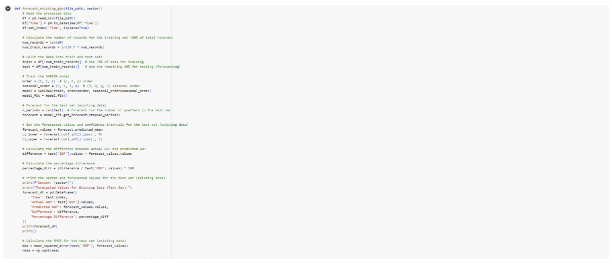
```python
# Calculate the number of records for the training set
num_records = len(df)
num_train_records = int(0.7 * num_records)
```

Figure 8: Train and Test Split for SARIMA



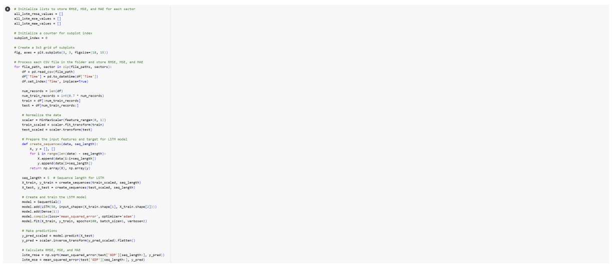
```python
def forecast_existing_gdp(file_path, sector):
    # Read the processed data
    df = pd.read_csv(file_path)
    df['Time'] = pd.to_datetime(df['Time'])
    df.set_index('Time', inplace=True)

    # Calculate the number of records for the training set (80% of total records)
    num_records = len(df)
    num_train_records = int(0.8 * num_records)

    # Split the data into train and test sets
    train = df[:num_train_records]  # Use 70% of data for training
    test = df[num_train_records:]   # Use the remaining 30% for testing (forecasting)

    # Train the ARIMA model
    order = (1, 1, 1)  # (p, d, q) order
    model = ARIMA(train['GDP'], order=order)
    model_fit = model.fit()

    # Forecast for the test set (existing data)
    n_periods = len(test)  # Forecast for the number of quarters in the test set
    forecast = model_fit.get_forecast(steps=n_periods)

    # Get the forecasted values and confidence intervals for the test set (existing data)
    forecast_values = forecast.predicted_mean
    ci = forecast.conf_int(alpha=0.05)  # 95% confidence intervals

    # Calculate the difference between actual GDP and predicted GDP
    difference = test['GDP'].values - forecast_values.values

    # Calculate the percentage difference
    percentage_diff = (difference / test['GDP'].values) * 100

    # Print the sector and forecasted values for the test set (existing data)
    print(f"Sector: {sector}")
    print("Forecasted Values for Existing Data (Test Set):")
    forecast_df = pd.DataFrame({
        'Time': test.index,
        'Actual GDP': test['GDP'].values,
        'Predicted GDP': forecast_values.values,
        'Lower CI': ci.iloc[:, 0].values,
        'Upper CI': ci.iloc[:, 1].values,
        'Difference': difference,
        'Percentage Difference': percentage_diff
    })
    print(forecast_df)
    print()

    # Calculate the RMSE for the test set (existing data)
    mse = mean_squared_error(test['GDP'], forecast_values)
    rmse = np.sqrt(mse)
```

Figure 9: ARIMA Model



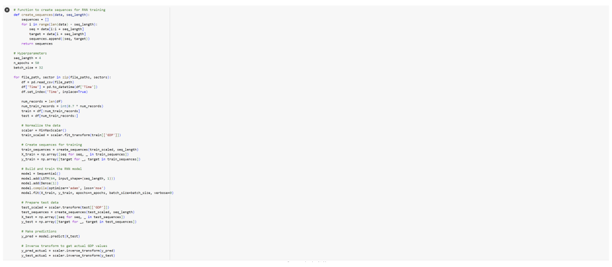
```python
def forecast_existing_gdp(file_path, sector):
    # Read the processed data
    df = pd.read_csv(file_path)
    df['Time'] = pd.to_datetime(df['Time'])
    df.set_index('Time', inplace=True)

    # Calculate the number of records for the training set (80% of total records)
    num_records = len(df)
    num_train_records = int(0.7 * num_records)

    # Split the data into train and test sets
    train = df[:num_train_records]  # Use 70% of data for training
    test = df[num_train_records:]   # Use the remaining 30% for testing (forecasting)

    # Train the SARIMA model
    order = (1, 1, 1)  # (p, d, q) order
    seasonal_order = (1, 1, 1, 4)  # (P, D, Q, S) seasonal order
    model = SARIMAX(train, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()

    # Forecast for the test set (existing data)
    n_periods = len(test)  # Forecast for the number of quarters in the test set
    forecast = model_fit.get_forecast(steps=n_periods)

    # Get the forecasted values and confidence intervals for the test set (existing data)
    forecast_values = forecast.predicted_mean
    ci_lower = forecast.conf_int().iloc[:, 0]
    ci_upper = forecast.conf_int().iloc[:, 1]

    # Calculate the difference between actual GDP and predicted GDP
    difference = test['GDP'].values - forecast_values.values

    # Calculate the percentage difference
    percentage_diff = (difference / test['GDP'].values) * 100

    # Print the sector and forecasted values for the test set (existing data)
    print(f"Sector: {sector}")
    print("Forecasted Values for Existing Data (Test Set):")
    forecast_df = pd.DataFrame({
        'Time': test.index,
        'Actual GDP': test['GDP'].values,
        'Predicted GDP': forecast_values.values,
        'Difference': difference,
        'Percentage Difference': percentage_diff
    })
    print(forecast_df)
    print()

    # Calculate the RMSE for the test set (existing data)
    mse = mean_squared_error(test['GDP'], forecast_values)
    rmse = np.sqrt(mse)
```

Figure 10: SARIMA Model



```python
# Initialize lists to store RMSE, MSE, and MAE for each sector
all_lstm_rmse_values = []
all_lstm_mse_values = []
all_lstm_mae_values = []

# Initialize a counter for subplot index
subplot_index = 0

# Create a 3x3 grid of subplots
fig, axes = plt.subplots(3, 3, figsize=(18, 15))

# Process each CSV file in the folder and store RMSE, MSE, and MAE
for file_path, sector in zip(file_paths, sectors):
    df = pd.read_csv(file_path)
    df['Time'] = pd.to_datetime(df['Time'])
    df.set_index('Time', inplace=True)

    num_records = len(df)
    num_train_records = int(0.7 * num_records)
    train = df[:num_train_records]
    test = df[num_train_records:]

    # Normalize the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    train_scaled = scaler.fit_transform(train)
    test_scaled = scaler.transform(test)

    # Prepare the input features and target for LSTM model
    def create_sequences(data, seq_length):
        X, y = [], []
        for i in range(len(data) - seq_length):
            X.append(data[i:i+seq_length])
            y.append(data[i+seq_length])
        return np.array(X), np.array(y)

    seq_length = 5  # Sequence length for LSTM
    X_train, y_train = create_sequences(train_scaled, seq_length)
    X_test, y_test = create_sequences(test_scaled, seq_length)

    # Create and train the LSTM model
    model = Sequential()
    model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=2)

    # Make predictions
    y_pred_scaled = model.predict(X_test)
    y_pred = scaler.inverse_transform(y_pred_scaled).flatten()

    # Calculate RMSE, MSE, and MAE
    lstm_rmse = np.sqrt(mean_squared_error(test['GDP'][seq_length:], y_pred))
    lstm_mse = mean_squared_error(test['GDP'][seq_length:], y_pred)
```

Figure 11: LSTM Model



```python
# Function to create sequences for RNN training
def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        seq = data[i:i + seq_length]
        target = data[i + seq_length]
        sequences.append((seq, target))
    return sequences

# Hyperparameters
seq_length = 4
n_epochs = 50
batch_size = 32

for file_path, sector in zip(file_paths, sectors):
    df = pd.read_csv(file_path)
    df['Time'] = pd.to_datetime(df['Time'])
    df.set_index('Time', inplace=True)

    num_records = len(df)
    num_train_records = int(0.7 * num_records)
    train = df[:num_train_records]
    test = df[num_train_records:]

    # Normalize the data
    scaler = MinMaxScaler()
    train_scaled = scaler.fit_transform(train[['GDP']])

    # Create sequences for training
    train_sequences = create_sequences(train_scaled, seq_length)
    X_train = np.array([seq for seq, _ in train_sequences])
    y_train = np.array([target for _, target in train_sequences])

    # Build and train the RNN model
    model = Sequential()
    model.add(LSTM(64, input_shape=(seq_length, 1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size, verbose=0)

    # Prepare test data
    test_scaled = scaler.transform(test[['GDP']])
    test_sequences = create_sequences(test_scaled, seq_length)
    X_test = np.array([seq for seq, _ in test_sequences])
    y_test = np.array([target for _, target in test_sequences])

    # Make predictions
    y_pred = model.predict(X_test)

    # Inverse transform to get actual GDP values
    y_pred_actual = scaler.inverse_transform(y_pred)
    y_test_actual = scaler.inverse_transform(y_test)
```

Figure 12: RNN Model