Configuration Manual

MSc Research Project
Data Analytics

Shital Namdeo Raut
Student ID: x19243294

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes, Dr. Pramod Pathak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shital Namdeo Raut
Student ID:	X19243294
Programme:	Data Analytics **Year:** 2021
Module:	MSc Research Project
Lecturer: **Submission Due Date:**	31/01/2022
Project Title:	Configuration Manual
Word Count:	700 **Page Count:** 9

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: Shital Namdeo Raut

Date: 31st January 2022

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

Shital Raut
X19243294

1. Introduction

This handbook provides thorough instructions on how to set it up all or most of the essential software and hardware for developing the full system in the first place. The setup instructions will assist in the replication of the study in a much more concrete way. We'll assess the system's overall functionality as well as its interface. (In other words, how a person will engage with our platform through the system user interface.)

The Configuration Manual will be divided into three sections excluding this introduction part.

- Environmental Setup
- Libraries Required
- Steps carried out in each Experiment

2. Environmental Setup

2.1 Hardware Requirements

- RAM: 8GB RAM.
- System Memory: 500 GB HDD.
- Processor: 2.40 GHz Intel. Core i5

2.2 Software Requirements

- Windows Edition: Windows 10
- Scripting Language: Python 3.6.3

2.3 Integrated Development Environment: Google Colab Free.
The project was implemented using python language on Google Colab Free version.

1. Cloud Storage: Google Drive. (For storing the dataset on cloud drive which will be used by collab.

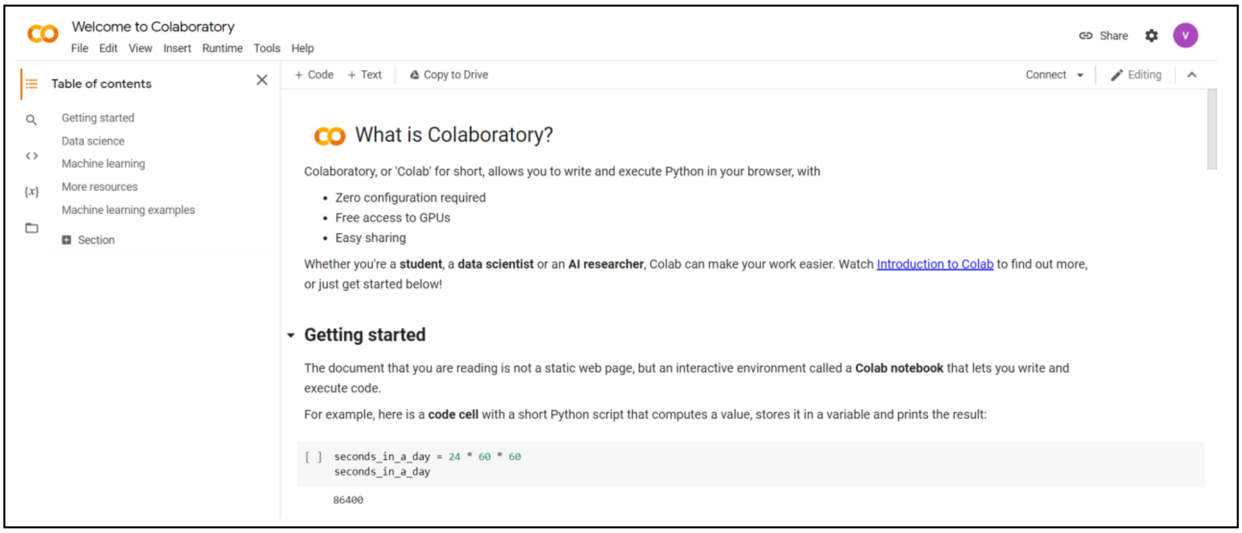

Figure 1: Google Colab

3. Libraries Required

Table 1 lists all of the libraries needed to complete this research study, as well as the procedures to load them. Before we can use it, we must first download some libraries.

pandas	import pandas as pd
numpy	import numpy as np
sklearn.metrics	mean_squared_error
matplotlib	import matplotlib.pyplot as plt
statsmodels.tsa.seasonal	import seasonal_decompose
statsmodels.tsa.arima_model	import ARIMA
statsmodels.tsa.api	ExponentialSmoothing, SimpleExpSmoothing, Holt
arch	Arch package

4. Implementation Details

1. Upload the dataset onto Google Drive.
2. Open Google Colab.
3. Follow the below steps on Google Colab:
    - Go to File and then Open Notebook- Train.ipynb.

4. Allocate runtime.
5. Execute all the steps till Mounting the Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Figure 2: Mounting the Google Drive.

6. Import the required libraries.

```
[ ] import numpy as np
    import pandas as pd
    from matplotlib import pyplot as plt
    from statsmodels.tsa.stattools import adfuller
    from statsmodels.tsa.seasonal import seasonal_decompose
    from statsmodels.tsa.arima_model import ARIMA
    from pandas.plotting import register_matplotlib_converters
    register_matplotlib_converters()
    from sklearn.metrics import mean_squared_error
    from statsmodels.tools.eval_measures import rmse
    from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
```

Figure 3: Importing the required libraries.

7. Read the data from csv file.

```
data = pd.read_csv('/content/drive/My Drive/Dairy_Time_Series/market-prices-dairy-products_en_7.csv')
```

Figure 4: Reading the data from csv file.

8. Convert "date" field to appropriate format and fetch only Ireland related data.

```
[ ] #Convert str column to DateTime
    data['Date'].astype('datetime64')

    0       2020-06-01
    1       2020-06-01
    2       2020-06-01
    3       2020-06-01
    4       2020-06-01
               ...
    30550   1991-01-01
    30551   1991-01-01
    30552   1991-01-01
    30553   1991-01-01
    30554   1991-01-01
    Name: Date, Length: 30555, dtype: datetime64[ns]
```

```
[ ] #Fetch only IE Data
    data = data.loc[data['Country'] == "IE"]
    data = data.loc[data['Product desc'] == "Butter"] #Change product accordingly
```

Figure 5: Converting the date field and fetching Ireland related data.

9. Clean the data for splitting it into training and testing data.

```
# Split data into train / test sets
train = data_final.iloc[:len(data_final)-12]
test = data_final.iloc[len(data_final)-12:] # set one year(12 months) for testing
```

Figure 6: Splitting the data into training and testing data set.

As the dataset is divided into training and testing dataset, we can now perform the experiments as follows:

5. Experiment 1: ARIMA Model

1) Fit ARIMA model to the dataset using following code:

```
# Fit ARIMA function to Butter dataset
model=ARIMA(data_final['Price'],order=(1,1,1))
model_fit=model.fit()
#model_fit.summary()
```

Figure 7: Applying ARIMA model for seasonal dairy price prediction.

2) Test ARIMA model using following code to find how well ARIMA model can predict and forecast the seasonal prices of Butter item:

```
#Testing Arima
start = len(train)
end = len(train) + len(test) - 1

# Predictions for one-year against the test set
predictions = model_fit.predict(start, end,
                                typ = 'levels').rename("Predictions")
# plot predictions and actual values
predictions.plot(legend = True)
test['Price'].plot(legend = True)

# Calculate root mean squared error
rmse_score = rmse(test["Price"], predictions)

# Calculate mean squared error
mse = mean_squared_error(test["Price"], predictions)

print("Arima Root Mean Squared Error : " + str(rmse_score))
print("Arima Mean Squared Error : " + str(mse))
```

Figure 8: Testing ARIMA model for seasonal dairy price prediction.

3) Forecast future Butter prices for the period of next three years using ARIMA model.

```
#Forcasting Arima
# Train the model on the full dataset
model = ARIMA(data_final['Price'],
                        order = (0, 1, 1)
                       )
result = model.fit()

# Forecast for the next 3 years
forecast = result.predict(start = len(data_final),
                          end = (len(data_final)-1) + 3 * 12,
                          typ = 'levels').rename('Forecast')

# Plot the forecast values
data_final['Price'].plot(figsize = (12, 5), legend = True)
forecast.plot(legend = True)
```

Figure 9: Forecasting future Butter prices using ARIMA model.

6. Experiment 2: ARIMA-GARCH Model

1) Install and import arch model using pip command as shown in below code snippet.

```
[ ]  !pip install arch
     !pip install pmdarima
     import arch
```

Figure 10: Importing arch module.

2) Fit Arima and Garch model on the given dataset.

```
[ ]  import statsmodels.tsa.arima_model as stm

     # fit ARIMA model
     model = stm.ARIMA(data_final['Price'], order=(3,1,2))
     model_fit = model.fit()

     # Fit garch model
     garch = arch.arch_model(data_final['Price'], vol='garch', p=1, o=0, q=1)
     garch_fitted = garch.fit()
     garch_forecast = garch_fitted.forecast(horizon=1)
     predicted_et = garch_forecast.mean['h.1'].iloc[-1]
```

Figure 11: Applying Arima and garch model on given dataset.

3) Test ARIMA-GARCH model using following code to find how well ARIMA-GARCH model can predict and forecast the seasonal prices of Butter item:

```
[ ]  #Testing
     predicted_mu = model_fit.predict(start, end,
                                      typ = 'levels')
     prediction = predicted_mu + predicted_et
     prediction.plot(legend = True)
     test['Price'].plot(legend = True)

     # Calculate root mean squared error
     rmse_score = rmse(test["Price"], prediction)

     # Calculate mean squared error
     mse = mean_squared_error(test["Price"], prediction)

     print("Arima-Garch Root Mean Squared Error : " + str(rmse_score))
     print("Arima-Garch Squared Error : " + str(mse))
```

Figure 12: Testing ARIMA-GARCH model for seasonal dairy price prediction.

4) Forecast future Butter prices for the period of next three years using ARIMA-GARCH model.

```
[ ] # Forecast for the next 3 years
    forecast = model_fit.predict(start = len(data_final),
                                 end = (len(data_final)-1) + 3 * 12,
                                 typ = 'levels').rename('Forecast')

    forcasting = forecast + predicted_et

    # Plot the forecast values
    data_final['Price'].plot(figsize = (12, 5), legend = True)
    forcasting.plot(legend = True)
```

Figure 13: Forecasting future Butter prices using ARIMA-GARCH model.

7. Experiment 3: Simple Exponential Model

1) Apply and test Simple Exponential Model using following code to find how well SEM model can predict and forecast the seasonal prices of Butter item:

```
[ ] #SimpleExpSmoothing
    y_hat_avg = test['Price']
    fit2 = SimpleExpSmoothing(np.asarray(data_final['Price'])).fit(smoothing_level=0.6,optimized=False)
    predictions = fit2.predict(start, end)
    predictions = pd.DataFrame(predictions,index=test.index,columns=['Predictions'])
    #predictions = pd.Series(predictions, index=test.index)

    # plot predictions and actual values
    predictions['Predictions'].plot(legend = True)
    test['Price'].plot(legend = True)
    # Calculate root mean squared error
    rmse_score = rmse(test["Price"], predictions['Predictions'])

    # Calculate mean squared error
    mse = mean_squared_error(test["Price"], predictions['Predictions'])

    print("SimpleExpSmoothing Root Mean Squared Error : " + str(rmse_score))
    print("SimpleExpSmoothing Mean Squared Error : " + str(mse))
```

Figure 14: Applying and Testing SEM model for seasonal dairy price prediction.

2) Forecast future Butter prices for the period of next three years using ARIMA-GARCH model.

```
[ ]  #Forcasting SimpleExpSmoothing
     SES_model = SimpleExpSmoothing(data_final['Price'])
     SES_result = SES_model.fit()

     # Forecast for the next 3 years
     forecast = SES_result.predict(start = len(data_final),
                                   end = (len(data_final)-1) + 3 * 12,
                                   )
     forecast = pd.DataFrame(forecast,columns=['Forecast'])

     # Plot the forecast values
     data_final['Price'].plot(figsize = (12, 5), legend = True)
     forecast['Forecast'].plot(legend = True)
```

Figure 15: Forecasting future Butter prices using SEM model.

8. Experiment 4: SARIMA Model

1) Fit SARIMA model to the dataset using following code:

```
[ ]  # Fit a SARIMAX
     from statsmodels.tsa.statespace.sarimax import SARIMAX

     model = SARIMAX(data_final['Price'],
                     order = (0, 1, 1),
                     seasonal_order =(2, 1, 1, 12))

     result = model.fit()
     #result.summary()
```

Figure 16: Applying SARIMA model for seasonal dairy price prediction.

2) Test SARIMA model using following code to find how well SARIMA model can predict and forecast the seasonal prices of Butter item:

```
[ ]  #Testing Sarima
     start = len(train)
     end = len(train) + len(test) - 1

     # Predictions for one-year against the test set
     predictions = result.predict(start, end,
                               typ = 'levels').rename("Predictions")

     # plot predictions and actual values
     predictions.plot(legend = True)
     test['Price'].plot(legend = True)

     # Calculate root mean squared error
     rmse_score = rmse(test["Price"], predictions)

     # Calculate mean squared error
     mse = mean_squared_error(test["Price"], predictions)

     print("Sarima Root Mean Squared Error : " + str(rmse_score))
     print("Sarima Mean Squared Error : " + str(mse))
```

Figure 17: Testing SARIMA model for seasonal dairy price prediction.

3) Forecast future Butter prices for the period of next three years using SARIMA model.

```
[ ]  #Forcasting Sarima
     # Train the model on the full dataset
     model = model = SARIMAX(data_final['Price'],
                             order = (0, 1, 1),
                             seasonal_order =(2, 1, 1, 12))
     result = model.fit()

     # Forecast for the next 3 years
     forecast = result.predict(start = len(data_final),
                               end = (len(data_final)-1) + 3 * 12,
                               typ = 'levels').rename('Forecast')

     # Plot the forecast values
     data_final['Price'].plot(figsize = (12, 5), legend = True)
     forecast.plot(legend = True)
```

Figure 18: Forecasting future Butter prices using SARIMA model.