Configuration Manual

MSc Research Project
MSc in Fintech

Chahine Firas
Student ID: 21233543

School of Computing
National College of Ireland

Supervisor: Pr. Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Firas Chahine
Student ID:	X21233543
Programme:	MSc in Fintech **Year:** 2022-2023
Module:	Research Project
Lecturer:	Pr. Brian Byrne
Submission Due Date:	14 August 2023
Project Title:	**"Comparative Analysis of ARIMA and LSTM Models for Predicting Electricity Consumption, Electricity Price and Stock Prices: A Case Study of Victoria, Australia"**
Word Count:	1504 **Page Count:** 17

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: Firas Chahine

Date: 14th August 2023

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):	

Contents

Configuration Manual

Chahine Firas
Student ID: x21233543

1 Introduction

This user configuration handbook provides a full, sequential description of required elements for both the product and the method. These are required to complete the research project titled "Comparative Analysis of ARIMA and LSTM Models for Predicting Electricity Consumption, Electricity Price and Stock Prices: A Case Study of Victoria, Australia". The procedures given include the hardware and software requirements as well. Furthermore, the handbook includes exemplary code snippets used in various models, as well as their associated results, all with the goal of providing practical instruction.

2 Data Gathering

This study proposal makes use of two separate datasets:

- The first dataset is on electricity price and consumption in Victoria Australia. This dataset consists of 14 columns and 216 rows. This dataset was obtained from Kaggle.com. Various price and demand parameters such as temperature, holidays, RRP, demand, solar exposure, negative RRP, positive RRP and more are recorded in this dataset, with RRP and demand chosen as the predictive variable. The data is load into the system in CSV format, and a preprocessing step is performed to properly format the date information.
- The second datasets come from Yahoo Finance and includes the NSX 200 Australian Index. This dataset is divided into 4 columns and has 1265 rows, consisting of the historical data of the NSX200 starting from 1/1/2015 ending in 31/12/2019. This dataset collection serves as the foundation for the full analysis and inquiry provided in this study. Various price and demand parameters such as Date, Open, High, Close were recorded in this dataset, with the Close chosen as the predictive variable. The data is also loaded into the system in CSV format, and a preprocessing step is performed to properly format the date and other information.

3 System Configuration

In this section, Hardware and Software specification used in the study will be discussed

3.1 Local machine Hardware Specification

The project was completed on the hardware configuration shown in Figure 1.

Device specifications

Inspiron 15-3567

Device name	DESKTOP-7C72RDU
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
Installed RAM	8.00 GB (7.87 GB usable)
Device ID	201CF833-B756-428E-8C11-1839B9EDD70E
Product ID	00325-81099-30785-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Touch support with 10 touch points

Figure 1: Device Configuration

3.2 Google Colab Hardware Specification

Hardware	*Specification*
RAM	12.7 GB
GPU (allocated based on runtime)	Tesla K80 / Tesla T4 (11.4GB)
Disk	107.7GB

Table 2: Google Colab Hardware Specifications

3.3 Software Specifications

Software	*Specification*
OS	Windows 10 Home (64-bit)
Programming Language	Python 3.8
IDE	Google Colab

4 Installation and package required.

This step is in the same Google Colab with ARIMA and LSTM. The imported packages for the data pre processing consists of the most basic packages such as numpy, pandas, in addition to importing packages that will be useful for the analysis of the machine learning models such as matplotlib.plot, seaborn, sklearn and statsmodels.
First, setting up python is a crucial step to do, then loading the datasets into the google colab by uploading the CSV file into the files in google Colab and reading it using the formulas shown in the picture below.

```
[2] !pip install rpy2==3.5.1
```

```
#reading the dataset in Python
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
energy_demand = pd.read_csv('complete_dataset.csv')
energy_demand.describe()
```

Figure 2: loading Python and reading the dataset (electricity Dataset).

```
[183] #reading the dataset in Python
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
ausindex = pd.read_csv('INDEX_AU_XASX_XJO.csv')
ausindex.describe()
```

Figure 3: reading the NSX200 Dataset in Colab

5 Data Preprocessing

5.1 Electricity Consumption Dataset

Once the dataset is loaded into python, first step in the process will be cleaning. The dataset is examined for missing values as shown in the figure 4

```
# To check the number of missing values
energy_demand.isna().sum()
```

```
date               0
demand             0
RRP                0
demand_pos_RRP     0
RRP_positive       0
demand_neg_RRP     0
RRP_negative       0
frac_at_neg_RRP    0
min_temperature    0
max_temperature    0
solar_exposure     1
rainfall           3
school_day         0
holiday            0
dtype: int64
```

Figure 4: missing values of electricity consumption dataset

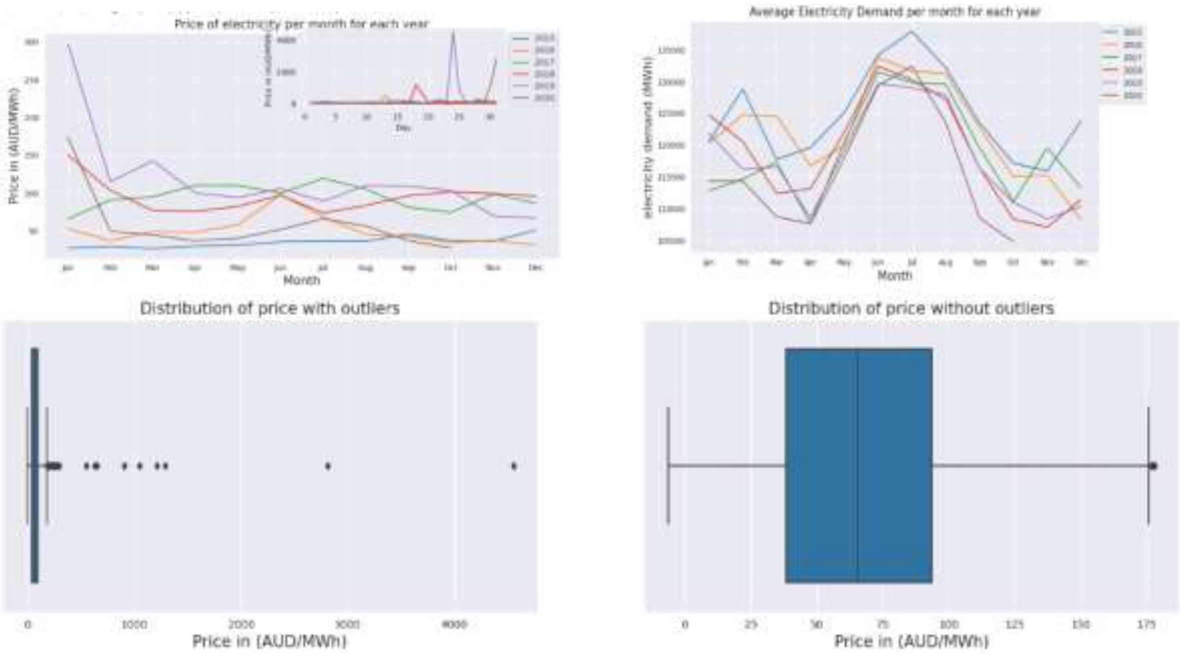
We can see 4 missing values, 1 from solar_exposure and 3 from rainfall. To deal with missing values, and since the number of missing values is low, we filled the missing values with median, then tested again the dataset for missing values and there was no missing values in the dataset as shown in the figure 5

```
[88] df.isna().sum()
```

```
date               0
demand             0
RRP                0
demand_pos_RRP     0
RRP_positive       0
demand_neg_RRP     0
RRP_negative       0
frac_at_neg_RRP    0
min_temperature    0
max_temperature    0
solar_exposure     0
rainfall           0
school_day         0
holiday            0
dtype: int64
```

Figure 5: missing values after filling them with the median

Second step in the data preprocessing is to explore the data. To do so, multiple plots were added to visualize the dataset and give a better understanding of it. Starting with plotting the variation of the price over a year, then the price of electricity per month for each of the 5 years on a monthly basis, followed by comparing the distribution of price with and without outliers, then plotting the variation of the positive and negative price over the year and for each year on a monthly basis. After plotting the price of the electricity with this different plots and parameters, next step would be visualizing the demand of the electricity with the same plots in order to understand how the demand of the electricity is changing over the year, and over the 5 years on a monthly basis. Here are some plot visualizing the demand and the price.



After visualizing the demand and the price of the electricity, next step will be to plot the hexogenous factors related to the electricity demand and price, in order to conclude the reason of the fluctuation of the price and the demand. One of these factors would be the temperature, which could give us an understanding of the seasons in Australia. Figure 6 shows the variation of the temperature over the month for the 5 years.

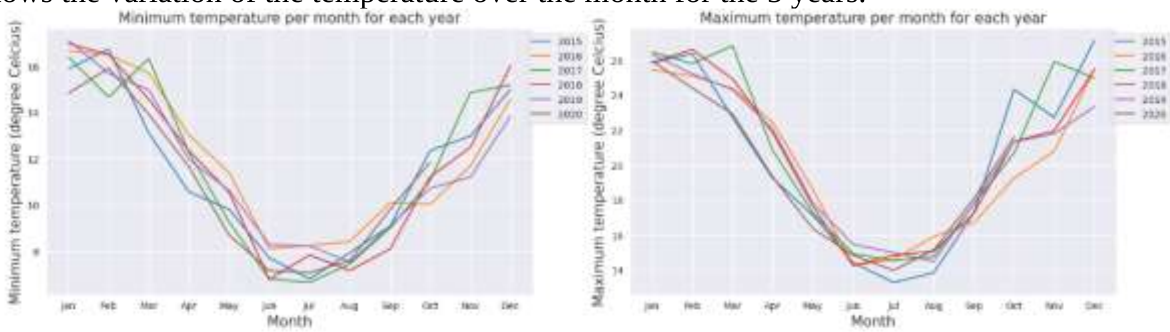


Figure 5: Min and Max temperature graph over the month for each of the 5 years

Next, drawing the correlation matrix between the multiple factors. Figure 6 shows the plot.

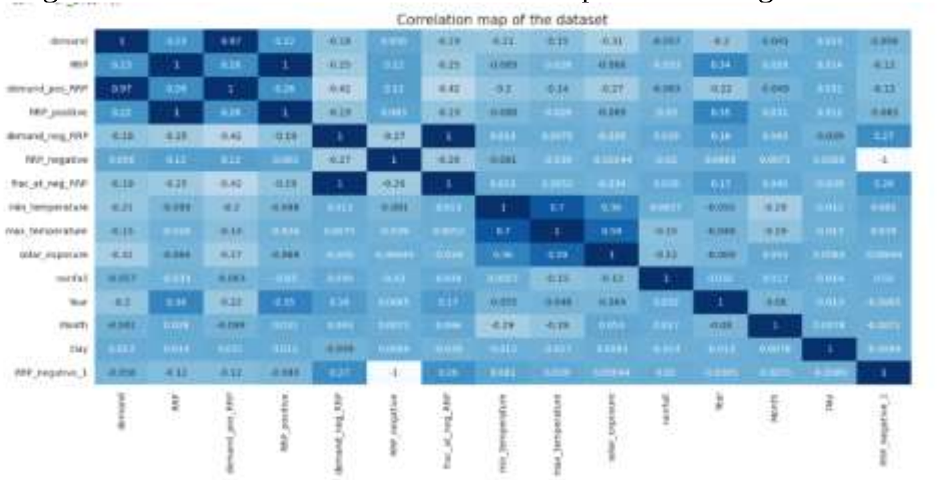


Figure 6: Correlation map of the dataset

5.2 NSX Index Dataset

After loading the NSX Dataset, plotting and visualization would be good in order to visualize how the data is distributed. Figure 7 shows the distribution of the dataset.

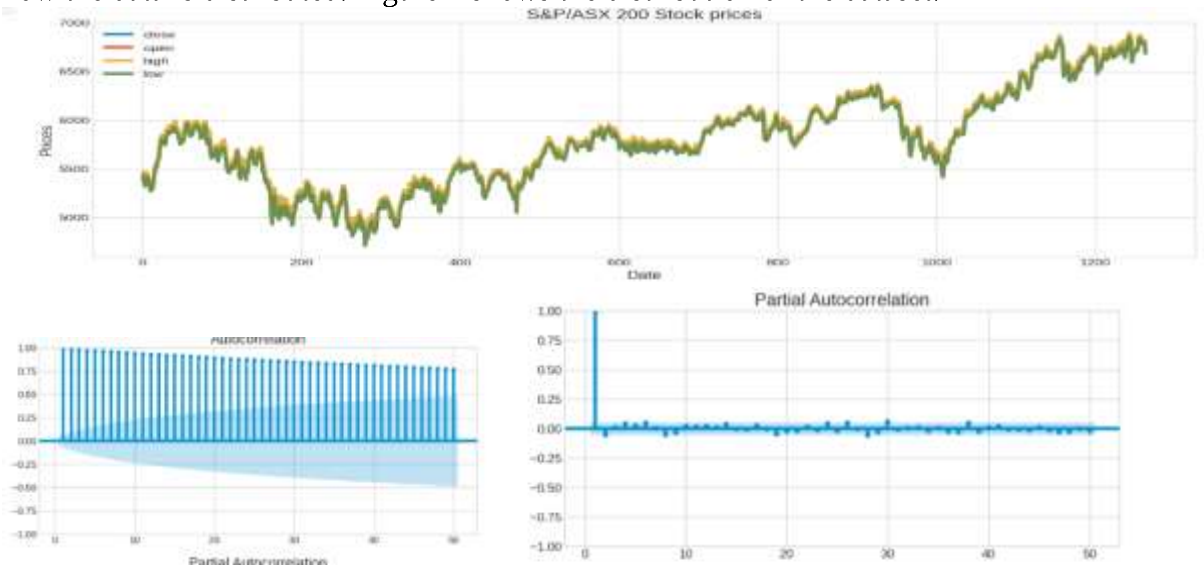


Figure 7: Closing Price vs Date and The autocorrelation and Partial Correlation Plot

6 Modelling

6.1 ARIMA

6.1.1 Electricity Consumption Dataset

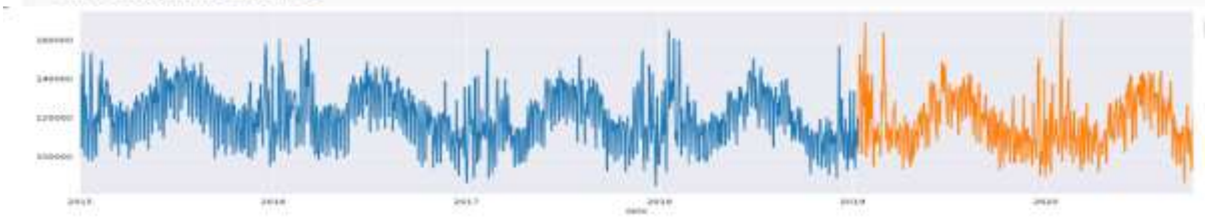
a) Check Stationarity of the Data

```
# create and summarize stationary version of time series
from statsmodels.tsa.stattools import adfuller

def adfuller_test(data):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    testing = adfuller(data, autolag='AIC')
    output = pd.Series(testing[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in testing[4].items():
        output['Critical Value (%s)'%key] = value
    print (output)
```

P value is less than 0.05, the null hypothesis is rejected

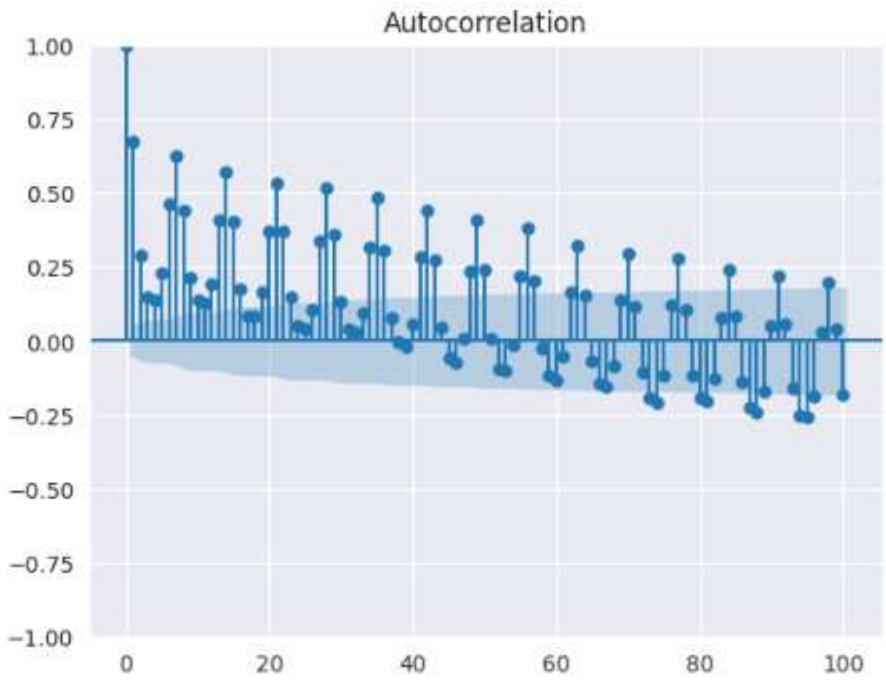
b) Split the data (70% train and 30% test)

```
[107] model_data = energy_demand[['date','demand','RRP','demand_pos_RRP','RRP_positive','demand_neg_RRP','RRP_negative','frac_at_neg_RRP','min_temperature','max_temperature

[108] model_data['date'] = pd.to_datetime(model_data['date'])
      model_data.set_index('date', inplace=True)
      train = model_data.iloc[0:round(len(model_data)*0.70)]
      test = model_data.iloc[len(train)-1:]
```



c) Testing Auto-Correlation

```
sm.graphics.tsa.plot_acf(train['demand'], lags=100)
plt.show()
```



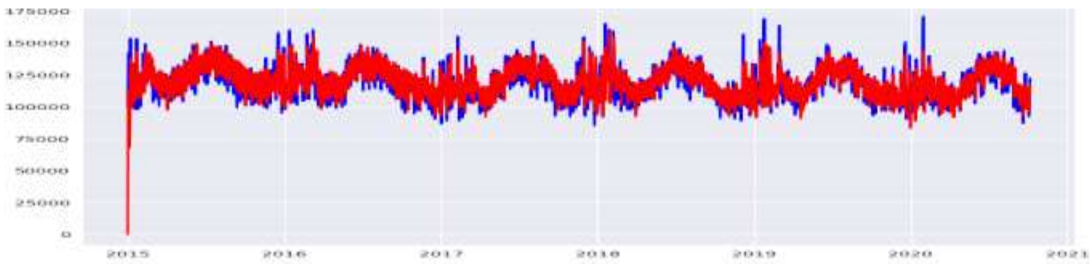
d) Fitting ARIMA using auto-ARIMA

```
[118] stepwise_fit = pm.auto_arima(df['demand'], trace=True, suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=44430.390, Time=1.58 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=45157.766, Time=0.09 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=45146.712, Time=0.12 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=45121.745, Time=0.29 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=45155.768, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=44493.571, Time=0.92 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=44435.078, Time=0.84 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=44424.102, Time=3.81 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=44437.023, Time=1.30 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=44222.293, Time=6.84 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=44351.102, Time=2.62 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=44188.029, Time=4.17 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=44214.451, Time=1.39 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=43919.694, Time=11.52 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=7.16 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=43931.447, Time=10.22 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=43950.043, Time=9.87 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=43900.858, Time=7.64 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=43979.564, Time=5.76 sec
 ARIMA(5,1,2)(0,0,0)[0]             : AIC=44185.911, Time=3.56 sec
 ARIMA(5,1,4)(0,0,0)[0]             : AIC=43931.403, Time=8.93 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=inf, Time=6.84 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=43946.615, Time=7.20 sec

Best model:  ARIMA(5,1,3)(0,0,0)[0]
Total fit time: 102.788 seconds
```

The best ARIMA model was ARIMA(5,1,3)

e) Plotting the forecasted with the Actual Model



f) Calculating the MAE, MSE, RMSE, Rsquared

```
# print the evaluation metrics and test results
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Akaike Information Criterion (AIC):", aic)
print("Bayesian Information Criterion (BIC):", bic)
print("R-squared:", r_squared)
print("Ljung-Box Test (p-values):")
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("Mean value of Electricity Demand:", mean_demand)
print(lb_test_result)
```

```
Mean Absolute Error (MAE): 6002.291113014625
Mean Squared Error (MSE): 71204206.00655644
Root Mean Squared Error (RMSE): 8440.515134406023
Akaike Information Criterion (AIC): 43900.85022915648
Bayesian Information Criterion (BIC): 43951.7268[illegible]
R squared: 0.[illegible]
Ljung Box Test (p values):
Mean Absolute Percentage Error (MAPE): 5.[illegible] %
Mean value of Electricity Demand: [illegible]
```

6.1.2 NSX 200 Index

a) Check Stationarity of the Data

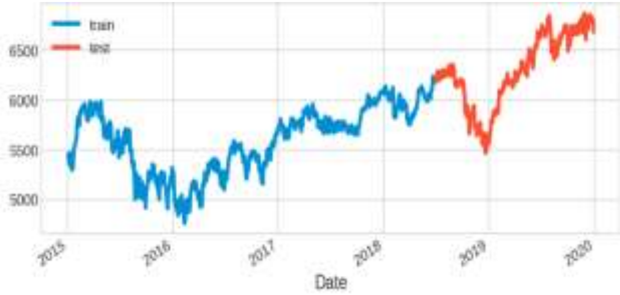
```
#Perform Augmented Dickey-Fuller test:
print('Results of Dickey Fuller Test:')
dftest = adfuller(ausindex['Close'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statis
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)
```

```
Results of Dickey Fuller Test:
Test Statistic                  -0.986497
p-value                          0.758157
#Lags Used                       7.000000
Number of Observations Used   1256.000000
Critical Value (1%)             -3.435567
Critical Value (5%)             -2.863844
Critical Value (10%)            -2.567997
dtype: float64
```

To reject the H0 (null Hypothesis) , the pvalue should less less than 0.05
Applying the difference data method, the result are shown below:

```
ADF Statistic: -14.347214
p-value: 0.000000
Critical Values:
        1%: -3.436
        5%: -2.864
        10%: -2.568
```

b) Split the data (70% training and 30% testing)

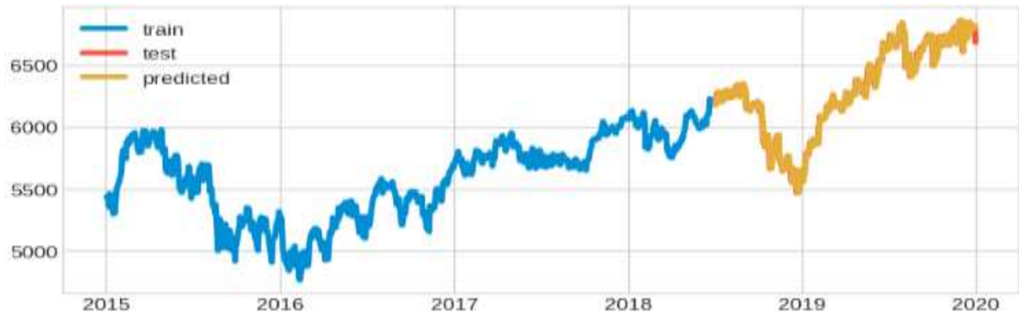


c) Fitting Auto Arima

```
] # Fit model with auto-arima
  import matplotlib.pyplot as plt
  from pmdarima import auto_arima
  arima_model = auto_arima(train, seasonal=False)
  arima_model.fit(train)
  results = arima_model.fit(train)
  print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  884
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -4626.682
Date:                Sat, 12 Aug 2023   AIC                           9255.363
Time:                        10:50:32   BIC                           9260.147
Sample:                             0   HQIC                          9257.192
                                - 884
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      2083.0411     74.056     28.128      0.000    1937.895    2228.187
===================================================================================
Ljung-Box (L1) (Q):                   0.06   Jarque-Bera (JB):               119.96
Prob(Q):                              0.81   Prob(JB):                         0.00
Heteroskedasticity (H):               0.36   Skew:                            -0.40
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.61
===================================================================================
```

d) Plotting the forecasted ARIMA Model



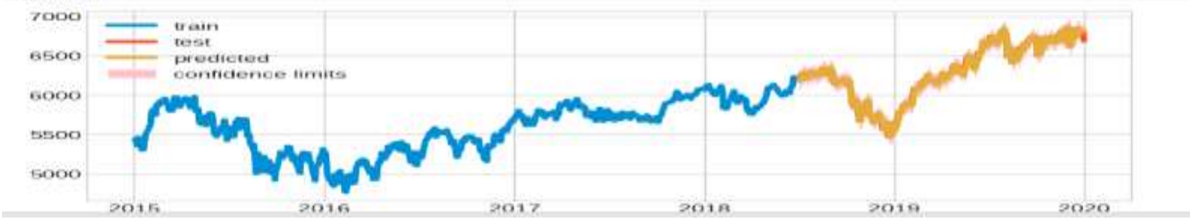
e) Calculating the MAE, MSE, RMSE, Rsquared

```
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test, predictions))
print("Root Mean Squared Error (RMSE):", rmse)
mae = mean_absolute_error(test, predictions)
print("Mean Absolute Error (MAE):", mae)
mse = mean_squared_error(test, predictions)
print("Mean Squared Error (MSE):", mse)
r_squared = r2_score(test, predictions)
print("R-squared:", r_squared)
aic = arima_model.aic()
bic = arima_model.bic()
print("Akaike Information Criterion (AIC):", aic)
print("Bayesian Information Criterion (BIC):", bic)
```

```
Root Mean Squared Error (RMSE): 45.66150832052377
Mean Absolute Error (MAE): 34.34131578947367
Mean Squared Error (MSE): 2084.973342105262
R-squared: 0.9834896453078981
Akaike Information Criterion (AIC): 13237.911197910547
Bayesian Information Criterion (BIC): 13243.052443032899
```

f) Forecasting with confidence interval

```
# Forecast with confidence intervals
lower_limits = [row[0] for row in confidence_intervals]
upper_limits = [row[1] for row in confidence_intervals]
# Plot predictions with conf_int
plt.figure(figsize=(10,4))
plt.plot(train, label='train')
plt.plot(test, label='test')
plt.plot(predicted, label='predicted')
plt.fill_between(test.index, lower_limits,
        upper_limits, color='pink', label='confidence limits')
plt.legend()
plt.show()
```



6.1.3 Electricity Price

a) Check Stationarity of the Data

```
[ ] #apply adf_price test on the series
    result1 = adfuller_test(df_price['demand'])
    print('')
    result2 = adfuller_test(df_price['RRP'])
    print('')
    result3 = adfuller_test(df_price['demand_pos_RRP'
    print('')
    result4 = adfuller_test(df_price['RRP_positive'])
    print('')
    result5 = adfuller_test(df_price['demand_neg_RRP'
    print('')
    result6 = adfuller_test(df_price['RRP_negative'])
    print('')
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.953447
p-value                          0.001675
#Lags Used                      26.000000
Number of Observations Used   2079.000000
Critical Value (1%)             -3.433499
Critical Value (5%)             -2.862931
Critical Value (10%)            -2.567511
dtype: float64
```

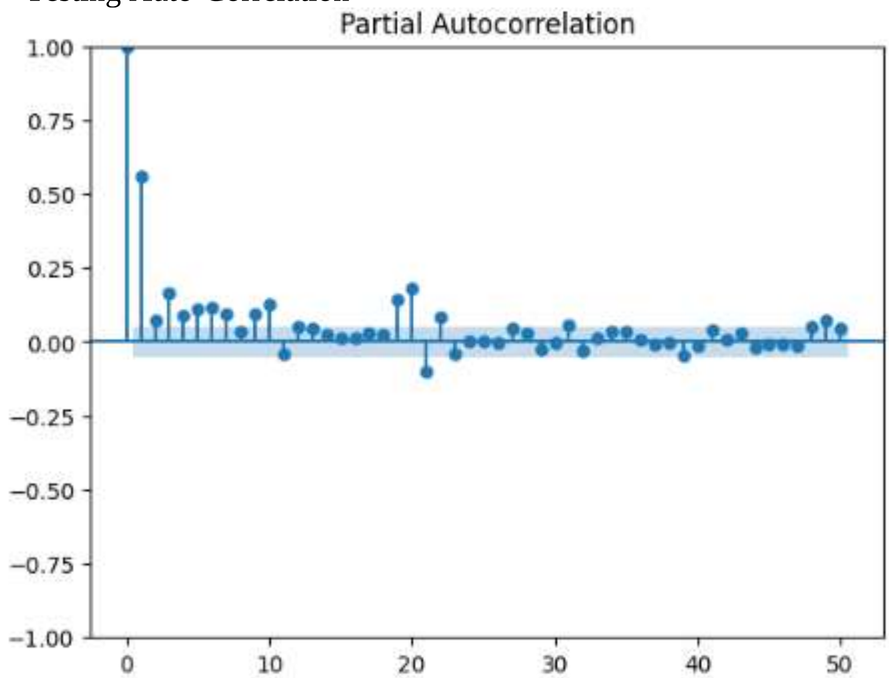
p-value is less than 0.05, the null hypothesis is rejected

b) Split the data (70% training and 30% testing)

```
[ ] model_data_price = df_price[['date','demand','RRP','demand_pos_RRP','RRP_positive','demand_neg_RRP','RRP_negative','frac_at_neg_RRP','min_temperature','max_temperatur
    train_price = model_data_price.iloc[0:round(len(model_data_price)*0.70)]
    test_price = model_data_price.iloc[len(train_price)-1:]
```

```
train_price['RRP'].plot(figsize=(20,5))
test_price['RRP'].plot(figsize=(20,5))
```

c) Testing Auto-Correlation



d) Fitting Auto Arima

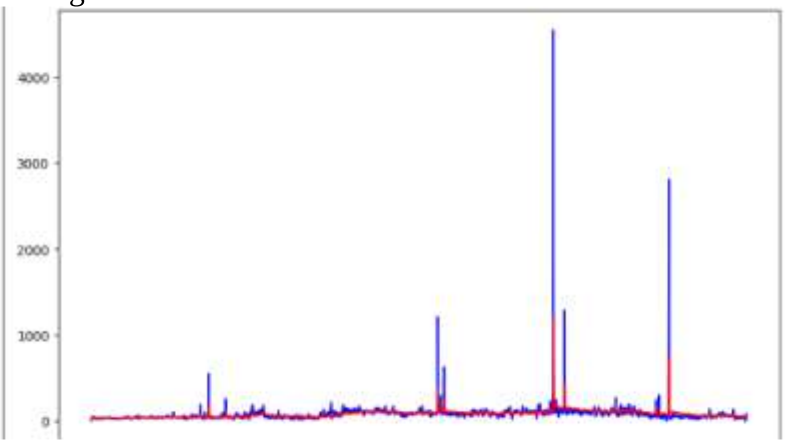
- iii) Fit ARIMA

```
[ ]  stepwise_fit_price = pm.auto_arima(df_price['RRP'], trace=True, suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=26266.100, Time=9.61 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=27240.733, Time=0.23 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=26930.023, Time=0.42 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=26372.524, Time=2.58 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=27238.733, Time=0.09 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=26264.158, Time=3.38 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=26264.890, Time=2.24 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=26262.796, Time=3.03 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=26264.196, Time=5.54 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=26748.727, Time=0.34 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=26260.837, Time=0.67 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=26370.529, Time=0.28 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=26928.023, Time=0.09 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=26262.234, Time=1.23 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=26262.195, Time=0.98 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=26262.919, Time=0.61 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=26746.727, Time=0.14 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=26264.151, Time=2.65 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 34.210 seconds
```

Best ARIMA model is (1,1,1)

e) Plotting the forecasted ARIMA Model



f) Calculating the MAE, MSE, RMSE, Rsquared

```
# Print the evaluation metrics and test results
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Akaike Information Criterion (AIC):", aic)
print("Bayesian Information Criterion (BIC):", bic)
print("R-squared:", r_squared)
print("Ljung-Box Test (p-values):")
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("Mean Value of Electricity Demand:", mean_RRP)
print(lb_test_result_price)
```

```
Mean Absolute Error (MAE): 24.6427743361466
Mean Squared Error (MSE): 15257.081959322215
Root Mean Squared Error (RMSE): 123.51956103922251
Akaike Information Criterion (AIC): 26260.83735863545
Bayesian Information Criterion (BIC): 26277.7935708738
R-squared: 0.10020496690701963
Ljung-Box Test (p-values):
Mean Absolute Percentage Error (MAPE): 37.61007189480868 %
Mean Value of Electricity Demand: 76.07955385072697
```

6.2 LSTM

6.2.1 Electricity Consumption Dataset

6.2.2 LSTM for electricity Price

6.2.3 LSTM for Index

a) Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

b) Splitting the data into train and test

```
##splitting dataset into train and test split
train_size = int(len(df1)*0.65)
train_data,test_data = df1[0:train_size,:],df1[train_size:len(df1),:1]
```

```
[ ] train_size
```

```
821
```

c) Creating LSTM

e.5) Creating LSTM

```
[ ] from keras.models import Sequential
    from keras.layers import Dense, Dropout, LSTM, Bidirectional
```

```
[ ] # Define the LSTM model
    def create_lstm_model(window_size):
        model = Sequential()
        model.add(LSTM(50, input_shape=(window_size, 1)))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        return model
```

d) Prepare LSTM

```
] # Prepare data for LSTM
  def prepare_lstm_data(data1, window_size):
      X, y = [], []
      for i in range(len(data) - window_size):
          X.append(data1[i:i + window_size])
          y.append(data1[i + window_size])
      return np.array(X), np.array(y)
```

e) Train LSTM

```
# Train the LSTM model
lstm_model1.fit(X_train1, y_train1, epochs=50, batch_size=64, verbose=1)
```

```
Epoch 8/50
13/13 [==============================] - 0s 24ms/step - loss: 0.0018
Epoch 9/50
13/13 [==============================] - 0s 19ms/step - loss: 0.0017
Epoch 10/50
13/13 [==============================] - 0s 20ms/step - loss: 0.0016
Epoch 11/50
13/13 [==============================] - 0s 27ms/step - loss: 0.0016
Epoch 12/50
13/13 [==============================] - 0s 27ms/step - loss: 0.0016
Epoch 13/50
13/13 [==============================] - 0s 18ms/step - loss: 0.0016
Epoch 14/50
13/13 [==============================] - 0s 23ms/step - loss: 0.0016
Epoch 15/50
13/13 [==============================] - 0s 22ms/step - loss: 0.0016
Epoch 16/50
13/13 [==============================] - 0s 26ms/step - loss: 0.0016
Epoch 17/50
```

f) Evaluation

```
# Calculate RMSE,MAE,MSE,R sqaured,MAPE
rmse = np.sqrt(mean_squared_error(y_test1, y_pred1))
mae = mean_absolute_error(y_test1, y_pred1)
mse = mean_squared_error(y_test1, y_pred1)
r_squared = r2_score(y_test1, y_pred1)
mape = np.mean(np.abs((y_test1 - y_pred1) / y_test1)) * 100
```

```
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R-squared:", r_squared)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
```

```
Root Mean Squared Error (RMSE): 66.28551079538008
Mean Absolute Error (MAE): 49.878688052856724
Mean Squared Error (MSE): 4393.768941404448
R-squared: 0.9628391709775476
Mean Absolute Percentage Error (MAPE): 0.8006405787352489 %
```