FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

BMMS2074 Statistics for Data Science

Title : Forecasting for Electrical Demand at Victoria, Australia

Table of Contents

1.0 Introduction.. 1
2.0 Data Description.. 1
 2.1 Data Overview.. 1
 2.2 Data Quality.. 2
 2.3 Visualizations.. 2
3.0 Exploratory Data Analysis (EDA).. 3
4.0 Results and Discussion.. 4
 4.1 Model Selection.. 4
 4.2 Model Fitting.. 5
 4.3 Model Evaluation.. 9
5.0 Conclusion.. 10
6.0 References.. 13
7.0 Appendices.. 14

1.0 Introduction

The dataset under analysis captures the daily electricity price and demand in Victoria, Australia. With the population of 6.7 million people residing in the state as of the year 2020, the energy market in Victoria is influenced by huge populations, holidays, unique market dynamics such as negative pricing where producers pay consumers to use electricity and so on.

Sourced from Kaggle, Daily Electricity Price and Demand Data (kaggle.com) , the data comes from the records of electricity demand and prices in Victoria, Australia. The frequency of the data is daily which spans from 1 January 2015 to 6 October 2020 with a total of 2106 days of observations. Armed with this dataset, We conduct a comprehensive analysis where our objective is to develop a time series model to forecast future electricity demand and analyze key trends and patterns of electric usage in Victoria. Understanding and forecasting can help us to manage electricity supply to better optimize energy resource allocation. This is crucial as electricity supply must match with demand to avoid electricity shortages and surpluses. In order to optimize energy resource allocation, using a monthly dataset would be more practical as managers would plan the resource allocation monthly instead of daily.

2.0 Data Description

2.1 Data Overview

The dataset consists of 2106 observations across 14 variables, capturing daily electricity demand and related features in Victoria, Australia. But we will convert and treat it as monthly data which spans from the year 2015 to 2020. The two key variables which are the primary focus of this analysis are as shown below while the remaining 12 can be referred in Appendix Table A1:

Key Variable	Data Type	Description
Date	Datetime	The date of the recorded electricity demand. Captured in the format of YYYY-MM-DD
Demand	Float	The total amount of electricity used throughout the day. Measured in megawatt per hour (MWh).

Table 2.0: List of key variables with descriptions

2.2 Data Quality

Ensuring data quality is crucial for accurate analysis and forecasting. After checking the entire dataframe, One issue encountered is missing values. We acknowledged that the variables **solar_exposure** and **rainfall** have missing values.

Instead of dropping only these two columns, we address this issue by dropping all columns that are related to RRP and select only the key variables since our analysis is focused on understanding and forecasting the Demand. This step has not only cleaned our dataset from missing values but also prepared us with only relevant data, allowing us to focus on the time series analysis to see the trend in electricity consumption. We also ensure that the Date column is in Date format (YYYY-MM-DD), so the data stays consistent and being treated correctly when we perform operations based on the time as well as excluding the last month since the data is not complete. This entire process is cited in Appendix B.

2.3 Visualizations

Time series of monthly demand is plotted to provide insights into the data's structure and behavior enabling us to observe the distribution, existing patterns and stationarity of the dataset.

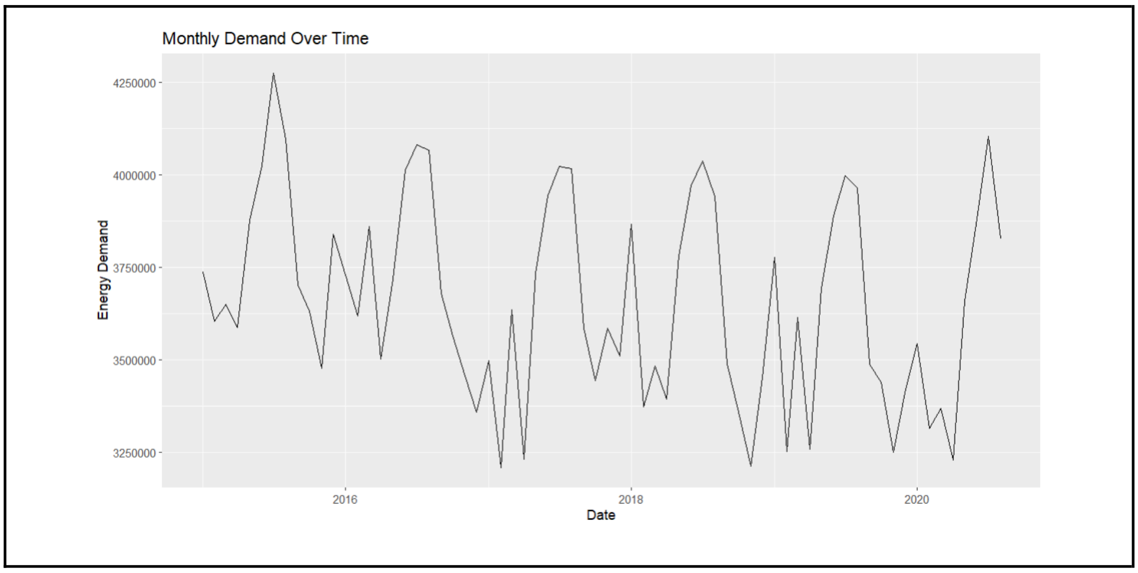


Figure 2.1: Time series plot of monthly demand

Based on Figure 2.1, the plot shows a slight downward trend and clear seasonal pattern, where energy demand appears to rise and fall consistently each year. Peaks in demand occur consistently every year showing a clear yearly pattern indicating the presence of seasonality . Noticing the seasonal pattern are a key feature of the data, the SARIMA model could be a good model for forecasting as it take seasonal component of data into account.

3.0 Exploratory Data Analysis (EDA)

The analysis begins by creating monthly demand data frames to visualize trends and seasonality, the key factors in predicting future demand. Line graphs for dataset reveal non-stationarity such as long-term trends and seasonal patterns. then convert monthly demand into a time series object which decomposes into its trend, seasonal, and random components. Clear seasonal patterns and trends are observed, with seasonal graphs emphasizing mid-year peaks in demand and dips at both ends. A seasonal subseries graph further highlights these patterns, showing June as the most consistent month with minimal fluctuations over the years. The process is shown in Appendix C

Trend Analysis

Time Series plots given in the initial view of trends shows it decreasing over time.

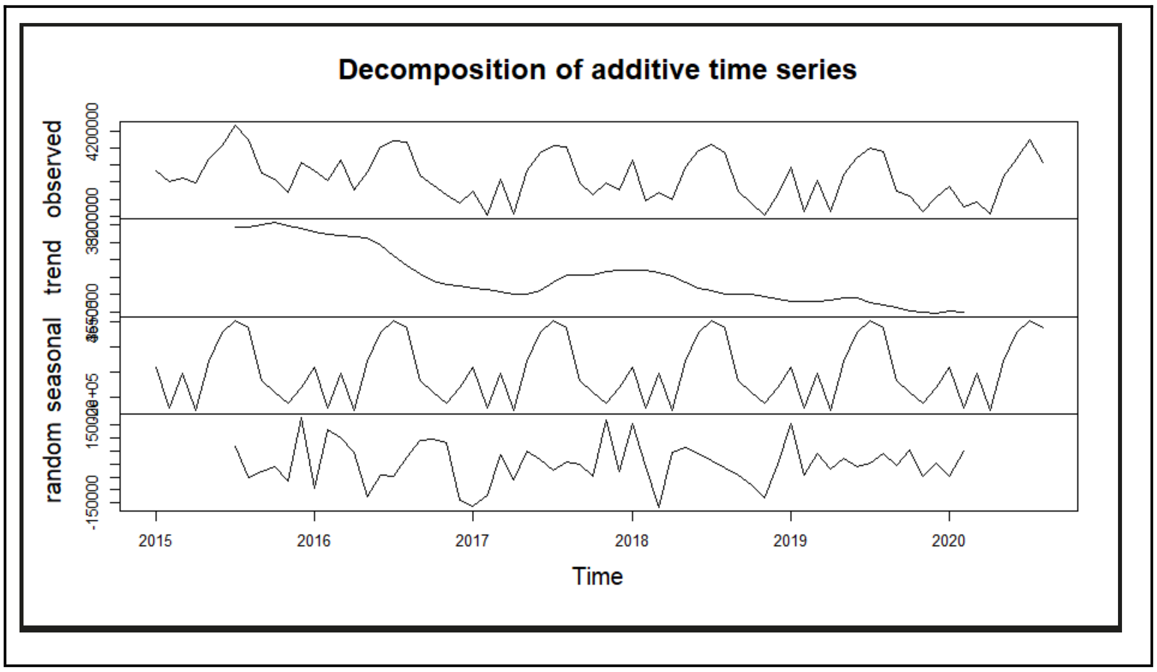
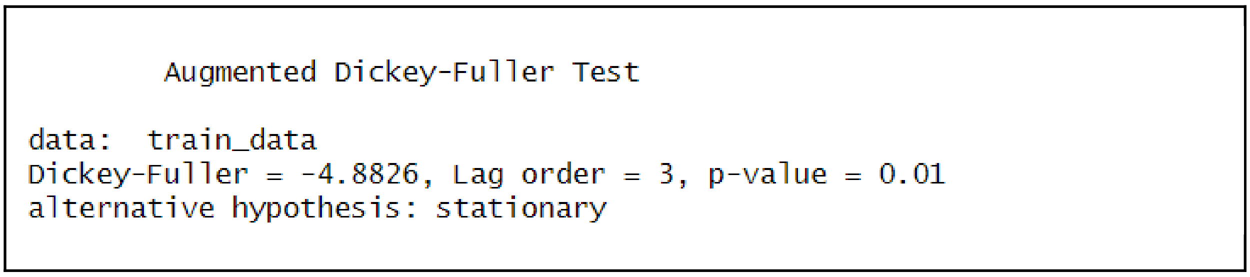


Figure 3.1: Plot of decomposition of additive time series

Seasonality

There is a clear pattern with peaks in July and dips in February, April and November. As referred in Appendix C, Figure C4

Stationarity



```
        Augmented Dickey-Fuller Test

data:  train_data
Dickey-Fuller = -4.8826, Lag order = 3, p-value = 0.01
alternative hypothesis: stationary
```

Figure 3.2: Augmented Dickey-Fuller Test Results.

ADF (Augmented Dickey-Fuller) Test:

- H0 (Null Hypothesis): The time series is non-stationary (contains a unit root).
- H1 (Alternative Hypothesis): The time series is stationary.

The ADF Test indicates that the p-value is 0.01. So reject H0 of non-stationarity at alpha = 0.05.The time series is likely stationary and has constant mean and variance over time.

Autocorrelation

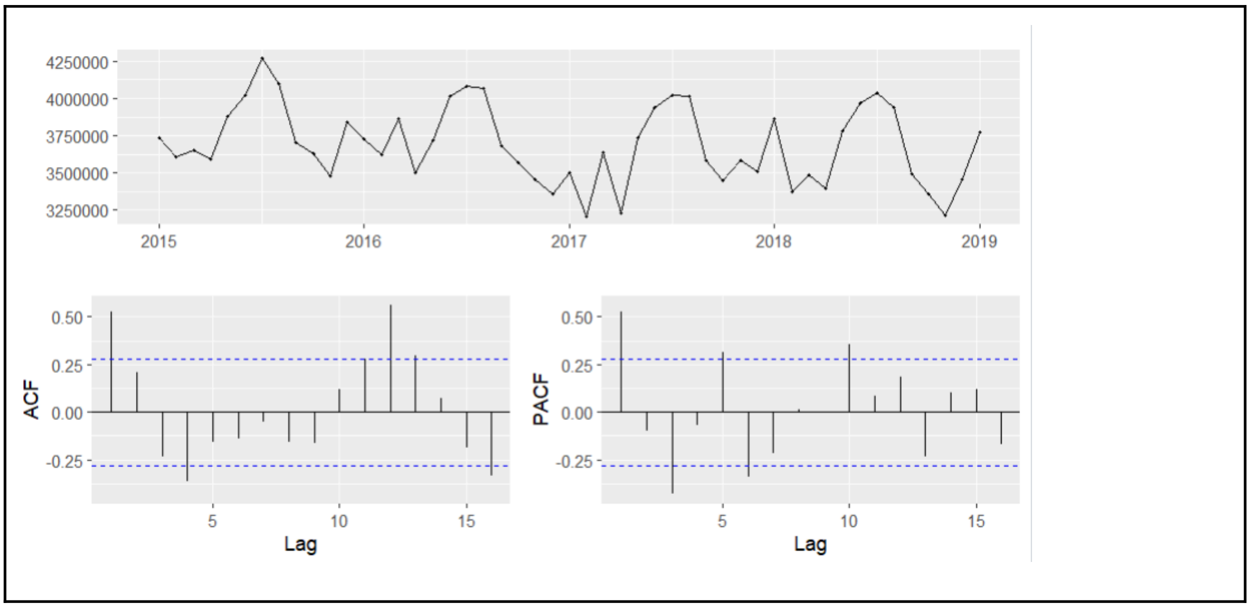


Figure 3.1

Then we observe the ACF and PACF plots to identify the parameters and possible candidates for our ARIMA model. The ACF shows autocorrelation at different lags and PACF measures relationships after removing effects of lags.

4.0 Results and Discussion

4.1 Model Selection

Model Candidates: Describe the different time series models considered, such as ARIMA, Exponential Smoothing, or advanced models like Prophet or LSTM.

1st Model Selection: Manual ARIMA(p,d,q)(P,D,Q)m

To manually fit an ARIMA model, we use the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots to understand the autocorrelation structure of the time series, in this case, the training data. Analyzing the Autocorrelation plots will allow us to determine the appropriate values for p,d,q and P,D,Q.

2nd Model Selection: Auto ARIMA

Auto ARIMA automatically performs a Test to determine if the time series dataset is stationary, then automatically searches for the optimal parameters for p,d,q and P,D,Q. It uses the Akaike Information Criterion (AICc) to select the best model.

3rd Model Selection: Seasonal Naive

This simple but powerful model is an extremely simple benchmark model that uses the previously observed value from the last season as the forecast for the current season.

4th Model Selection: Holt-Winters

It's an exponential smoothing method that can handle trends and seasonal components for time series forecasting. It is a very useful model for strong seasonal time series datasets.

4.2 Model Fitting

First Model Fitting: MANUAL ARIMA

First we plot ACF, PACF of the data for possible candidates

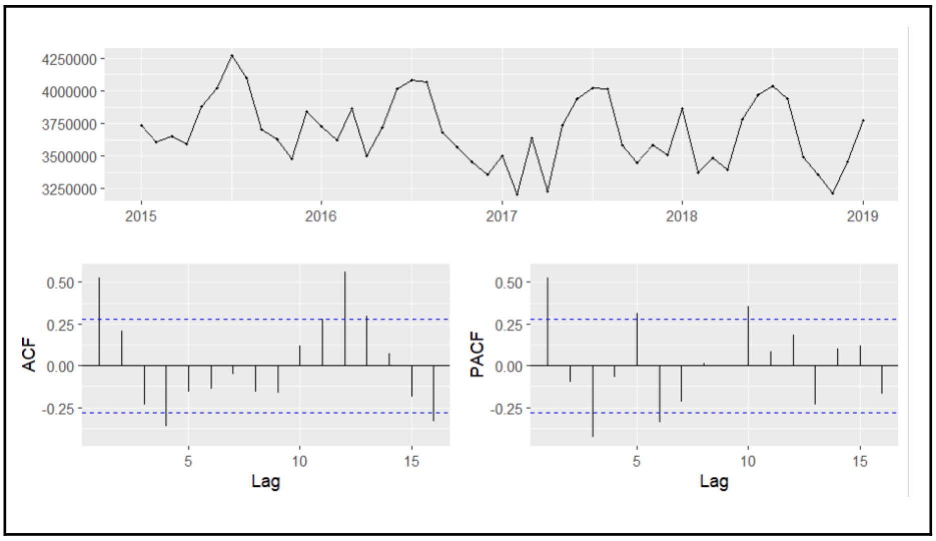


Figure 4.0

ARIMA MODEL: ARIMA(p,d,q)(P,D,Q)m

Based on the figure above, there is an obvious seasonal element to it. Therefore, we will perform 1 seasonal differencing to remove the seasonal element.

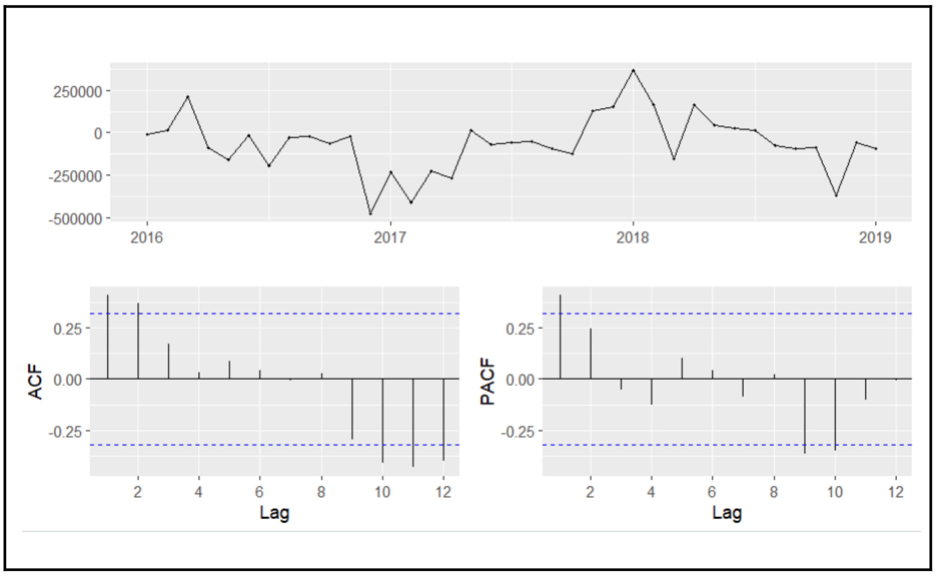
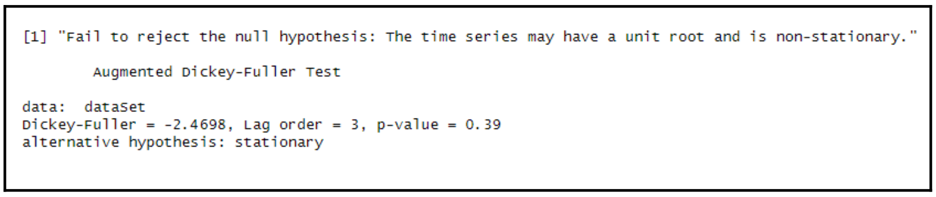


Figure 4.1 *After 1 seasonal differencing*

We can see there are no more seasonal elements from the above plot. So now we test the non-seasonal stationarity with the adf test.



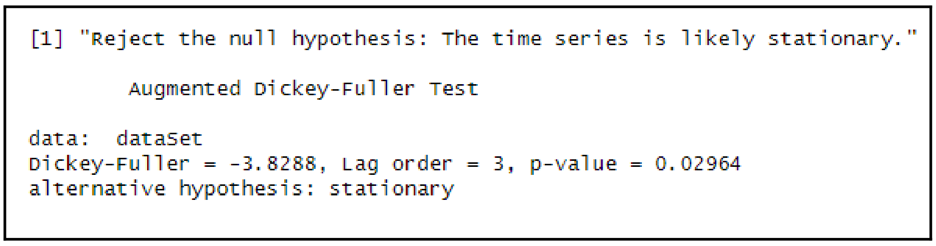
```
[1] "Fail to reject the null hypothesis: The time series may have a unit root and is non-stationary."

	Augmented Dickey-Fuller Test

data:  dataSet
Dickey-Fuller = -2.4698, Lag order = 3, p-value = 0.39
alternative hypothesis: stationary
```

Figure 4.2

We will be using ADF (Augmented Dickey-Fuller) to determine if the time series are stationary. For the ADF Test, the time series is likely stationary at p-value = 0.01

- H0 (Null Hypothesis): The time series is non-stationary (contains a unit root).
- H1 (Alternative Hypothesis): The time series is stationary.

The ADF Test indicates that the p-value is 0.39. So fail to reject H0 at alpha = 0.05.The time series is likely not stationary. Therefore,we will do 1 non-seasonal differencing and observe the adf.



```
[1] "Reject the null hypothesis: The time series is likely stationary."

	Augmented Dickey-Fuller Test

data:  dataSet
Dickey-Fuller = -3.8288, Lag order = 3, p-value = 0.02964
alternative hypothesis: stationary
```

Figure 4.3 ADF test

After the non-seasonal differencing, the ADF Test indicates that the p-value is 0.0296. So reject H0 at alpha = 0.05 indicating the time series is likely stationary.

Then we observe the ACF and PACF plots *after 1 seasonal and non-seasonal differencing* to identify the parameters and possible candidates for our ARIMA model. ACF shows autocorrelation at different lags. PACF measures relationships after removing effects of lag. Based on the ACF and PACF, we choose d=1 and D=1 because the non-seasonal and seasonal difference was done once. Then, P and Q = 0 because there are no spikes at lag 12 for ACF and PACF plots. Then, for p and q, because the ACF and PACF plot both look the same exponentially decreasing with a spike at lag 1. We will test possible combinations for p and q. Here are the results:

Model	*MAPE for test*	*RMSE for Train*	*RMSE for Test*	*RMSE Difference (%)*	*AICc*	*Residual P-Value*
ARIMA(1,1,0)(0,1,0)12	2.268923	138458.6	103964.9	24.91	969.9354	0.0701 (Pass)
ARIMA(0,1,1)(0,1,0)12	3.278616	136087.6	139812.3	-2.73	968.8229	0.1693 (Pass)
ARIMA(1,1,1)(0,1,0)12	3.283119	136087.3	140039.0	-2.90	970.993	0.1057 (Pass)

Refer Appendix G, Figure G2 for the formula of RMSE Difference

Using the residuals' p-value determines if the model has passed the test verifying whether the residuals (the differences between predicted and actual values) are random and uncorrelated. A low p-value indicates that the residuals contain patterns or trends that the model failed to capture, suggesting the model is not a good fit. Conversely, a high p-value (above a significance threshold, like 0.05) implies the residuals are random, meaning the model has adequately captured the data patterns and passed the test for good fit. Based on the residuals, the first model ARIMA(0,1,0)(0,1,0)12 will be rejected. Then, based on the AICc and RMSE difference, the third model ARIMA(0,1,1)(0,1,0)12 seems the most promising at 968.8229 and -2.73%. Therefore, currently, the best model parameters based on the plot is ARIMA(0,1,1)(0,1,0)[12].

Alternative ARIMA Model: ARIMA(1,1,0)(0,1,1)

After testing other various models, ARIMA(1,1,0)(0,1,1)12 showed the lowest AICc value at 968.47 and minimal difference between Train RMSE (121808.2) and Test RMSE (118236.3), indicating slight underfitting. This indicates that the model captures the underlying patterns, including seasonality, without overfitting. The combination of low MAPE (2.94), a minimal RMSE difference of at 3%, and robust performance on both training and test data supports the choice of ARIMA(1,1,0)(0,1,1) as a suitable model for forecasting with high predictive accuracy. Therefore, we will proceed with this model.

Formula for ARIMA(1,1,0)(0,1,1)[12] Model

$$(1-\phi_1 B)(1-B)(1-B^{12})Y_t = (1-\Theta_1 B^{12})\varepsilon_t$$

$$\text{AR1:} \ -0.5127 \quad \text{SMA1:} \ -0.6185$$

Step 1: Starting ARIMA Model Equation

$$(1+0.5127B)(1-B)(1-B^{12})Y_t = (1-0.6185B^{12})\varepsilon_t$$

Step 2: Expand the terms on both sides

$$(1+0.5127B)(1-B-B^{12}+B^{13})Y_t = \varepsilon_t - 0.6185B^{12}\varepsilon_t$$

Step 3: Multiply the terms

$$(1-B-B^{12}+B^{13}+0.5127B-0.5127B^2-0.5127B^{13}+0.5127B^{14})Y_t = \varepsilon_t - 0.6185B^{12}\varepsilon_t$$

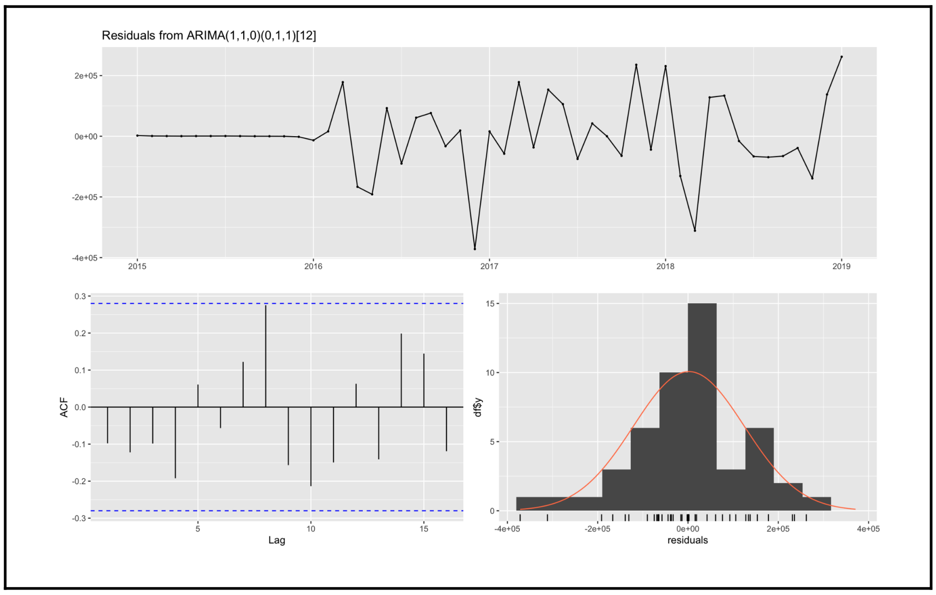
Step 4: Simplify

$$Y_t - Y_{t-1} - Y_{t-12} + Y_{t-13} + 0.5127Y_{t-1} - 0.5127Y_{t-2} - 0.5127Y_{t-13} + 0.5127Y_{t-14} = \varepsilon_t - 0.6185\varepsilon_{t-12}$$

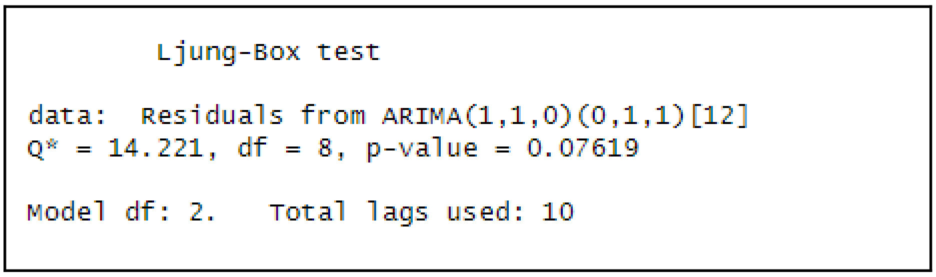
Step 5: Final simplified model

$$Y_t = 0.4873Y_{t-1} + 0.5127Y_{t-2} + Y_{t-12} - 0.4873Y_{t-13} - 0.5127Y_{t-14} + \varepsilon_t - 0.6185\varepsilon_{t-12}$$

Plot of Residual for ARIMA(1,1,0)(0,1,1)



Ljung-Box test results



```
        Ljung-Box test

data:  Residuals from ARIMA(1,1,0)(0,1,1)[12]
Q* = 14.221, df = 8, p-value = 0.07619

Model df: 2.   Total lags used: 10
```

- H0: Residuals are randomly distributed and uncorrelated
- H1:Residuals are not randomly distributed and/or correlated

The p-value is 0.07619, do not reject H0 at alpha = 0.05. It suggests the residuals of the model are likely randomly distributed and uncorrelated, indicating good model fit.

4.3 Model Evaluation

In this comparison table shown in Table I below, two key metrics are used to evaluate the models: **RMSE (Root Mean Squared Error)** and **MAPE (Mean Absolute Percentage Error)**.

RMSE measures the average magnitude of the prediction errors in the same units as the target variable (in this case, electric demand). It heavily penalizes larger errors, meaning that a lower RMSE indicates more accurate model predictions. For instance, **Auto ARIMA** has the lowest RMSE (108738.1 for training and 72324.83 for testing), suggesting it makes smaller errors in its forecasts compared to the other models.

MAPE represents the average absolute percentage difference between the predicted and actual values. It provides a percentage-based accuracy measure, making it easier to interpret across different datasets. A lower MAPE indicates better predictive accuracy. Here, **Auto ARIMA** again outperforms the others with the lowest MAPE (1.959689 for training and 1.607043 for testing), showing that its predictions are closest to the actual values in relative terms.

While **Manual ARIMA** also shows decent RMSE and MAPE, indicating it is accurate and generalizable, **Holts-Winters** and **Seasonal Naive** show higher RMSE and MAPE values, which suggest their forecasts are less precise and tend to deviate more from actual demand, especially in the training phase. Thus, RMSE and MAPE are essential in selecting the model with the best trade-off between precision and error minimization.

	Comparison table							
	Manual ARIMA		**Auto ARIMA**		**Holts-Winters**		**Seasonal Naive**	
SET	Train	Test	Train	Test	Train	Test	Train	Test
RMSE	121808.2	118236.3	108738.1	72324.83	139757.35	75476.49	176927.9	105943.8
MAPE	2.286996	2.940451	1.959689	1.607043	2.643607	1.618081	3.713575	2.581053

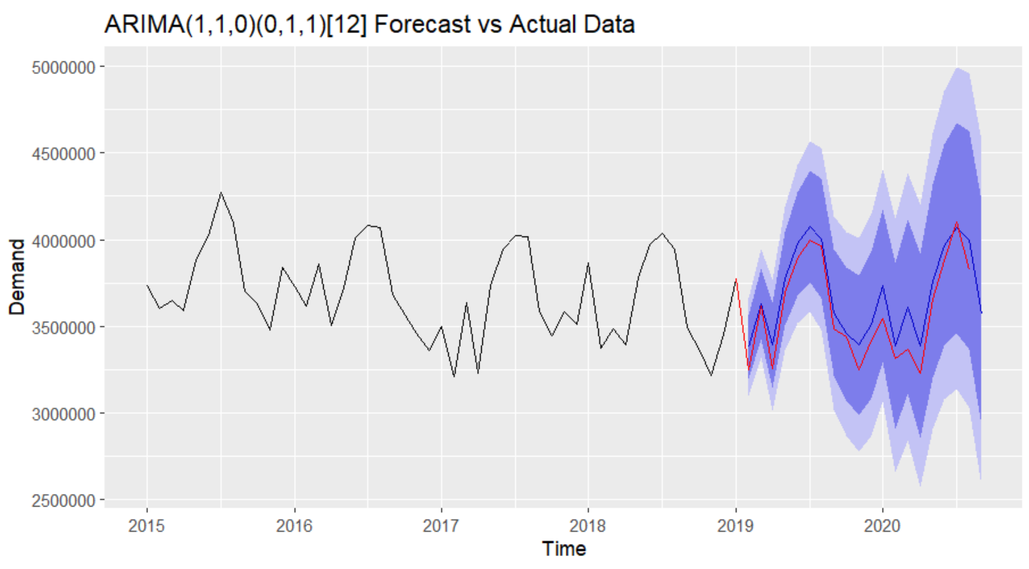
Table I Comparison Table

The **Manual ARIMA ARIMA(1,1,0)(0,1,1)[12]** model is chosen for forecasting electric demand due to its balanced performance between the training and test sets. With relatively low RMSE values (121808.2 for training and 118236.3 for testing), the model avoids significant overfitting or underfitting, ensuring consistent accuracy across datasets. Its MAPE values, 2.286996 for training and 2.940451 for testing, also reflect strong forecast accuracy with minimal error percentage. While some models like Auto ARIMA show slightly better performance, Manual ARIMA's small differences in both RMSE and MAPE between sets indicate a well-generalized model suitable for real-world applications.

Based on Table I, we have selected ManualArima ARIMA(1,1,0)(0,1,1)[12] with the lowest RMSE difference and MAPE difference to do the forecasting for the electric demand.

Forecasting

1st Model Forecasting: Manual ARIMA(1,1,0)(0,1,1)



5.0 Conclusion

5.1 Summary

In summary, there is a clear seasonal pattern in electric demand. The models selected aim to capture both the non-seasonal and seasonal components of the data. Hence, **model selection** is narrowed to **manual ARIMA**, **Auto ARIMA**, **Seasonal Naive**, and **Holt-Winters**. The manual ARIMA model is selected based on analyzing ACF and PACF plots, which help determine the model parameters (p, d, q, P, D, Q). Auto ARIMA automates the process, searching for the optimal parameters based on information criteria. Seasonal Naive uses last season's values as predictions, while Holt-Winters applies exponential smoothing.

For **model fitting**, the manual ARIMA model undergoes seasonal and non-seasonal differencing to achieve stationarity, confirmed by ADF tests. Different combinations of ARIMA parameters are tested, with results evaluated using RMSE and AICc values. The manual ARIMA(110)(011)[12] model performs well, balancing between training and test sets, and it is chosen as the best-fitting model. The residuals are checked for randomness using the Ljung-Box test, further confirming the model's validity.

In the **model evaluation** phase, performance metrics such as **RMSE** and **MAPE** are used to compare models. The **Auto ARIMA** shows the best performance with the lowest RMSE and MAPE, indicating strong forecasting ability. However, the manual ARIMA model, with slightly higher RMSE and MAPE, shows a good balance between training and test data, making it a reliable choice for forecasting electricity demand.

5.2 Implications

Based on our analysis in electricity price and demand, the model can provide insights for optimizing electricity grid management, pricing, and resource planning. The model can help grid operators manage electricity supply effectively, by anticipating periods of high demand, operators can reduce the risks of outages or surpluses, making the grid more reliable and efficient. This improves planning and reduces energy waste during low-demand periods. Besides, understanding demand fluctuations allows energy providers to implement dynamic pricing models, adjusting rates based on real-time demand. This benefits consumers through lower prices during off-peak hours and encourages energy conservation during peak times. This can stabilize demand fluctuations, smoothing out demand spikes resulting in predictable energy usage. Providers can also create a better balance of renewable and non-renewable energy sources, reducing reliance on fossil fuels and encouraging long term sustainability.

5.3 Limitations

We encountered some limitations in this assignment. First is that we couldn't test on all possible time series models due to time constraints, our chosen ARIMA model might not be the best fitting model, but at the very least it still gave us a promising result for this analysis.

Next one is that we did not test the model in a real-world setting. Since the dataset used is historical, the forecast might perform well on past data, but when practically deployed in a real-world environment where multiple external factors could also influence demand patterns. These unforeseen events were not factored into the model compromising forecast accuracy when in practice.

Lastly, not using advanced neural networks also falls under our limitations. Advanced neural networks are able to recognise complex patterns in data, like how holidays affect electricity demand over time while simpler models do not take this into account, which the analysis misses out on complex interactions between features leading to less accurate forecasting of electricity demand.

5.4 Recommendations

1. **Adopt the SARIMA(1,1,0)(0,1,1)[12] Model**: This model showed the lowest RMSE difference between training and test sets, indicating a balanced fit without overfitting or underfitting. Since the RMSE for training was slightly higher than the test set, it may be slightly underfitting, but it captures the seasonal patterns well, making it suitable for future forecasts.
2. **Caution with Auto ARIMA**: While the Auto ARIMA model achieved the lowest MAPE (1.607), its high RMSE difference suggests significant underfitting. It may not effectively capture patterns in the training data, which could affect its ability to generalize to unseen data.
3. **Preference for Manual ARIMA**: The manual ARIMA model, with the lowest RMSE difference, demonstrated a strong balance between the training and test sets, making it less likely to overfit or underfit. It's MAPE of 2.94 is still well below 5%, meaning it provides accurate predictions and is reliable across different datasets, making it better suited for handling new or unseen data.
4. **Explore Other Models**: Consider experimenting with advanced models like Long Short-Term Memory (LSTM) and Prophet for time series forecasting. These models might provide improved accuracy and better handling of complex patterns, especially for long-term predictions.

6.0 References

1. Hyndman, R. J., & Athanasopoulos, G. (2018). *8.7 ARIMA modelling in R | Forecasting: Principles and Practice (2nd ed)*. OTexts. Retrieved September 13, 2024, from https://otexts.com/fpp2/arima-r.html

2. Noble, J. (2024, May 24). *Introducing ARIMA models*. What are ARIMA models? Retrieved September 13, 2024, from https://www.ibm.com/topics/arima-model

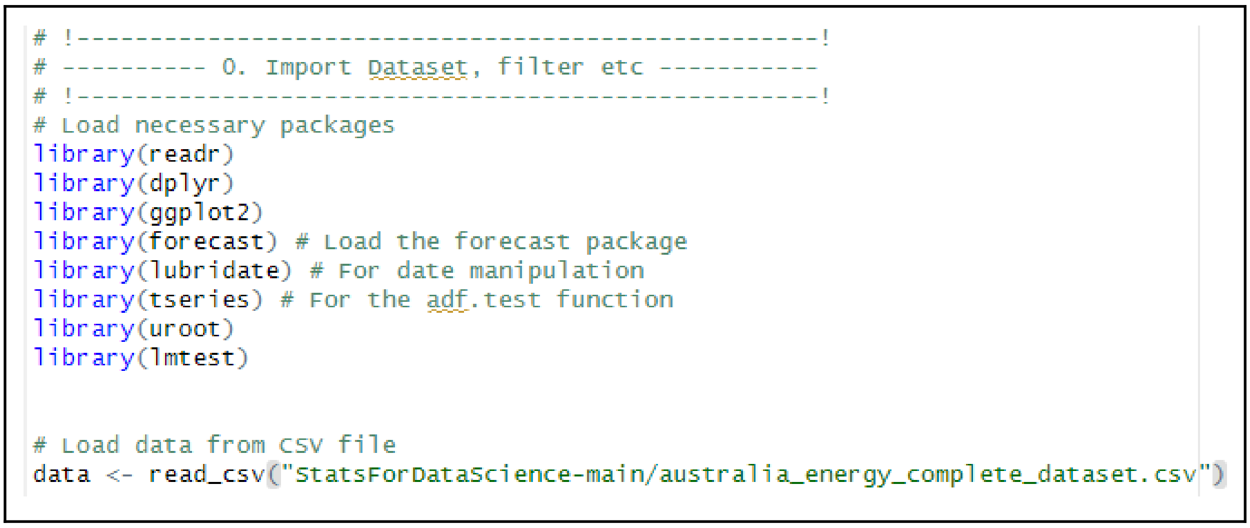
3. *What is a Neural Network?* (n.d.). IBM. Retrieved September 24, 2024, from https://www.ibm.com/topics/neural-networks

7.0 Appendices

Appendix A

Package Usage and Variable Documentation

Figure A1 shows a list of necessary packages used in this assignment.



```
# !--------------------------------------------------!
# ---------- 0. Import Dataset, filter etc -----------
# !--------------------------------------------------!
# Load necessary packages
library(readr)
library(dplyr)
library(ggplot2)
library(forecast) # Load the forecast package
library(lubridate) # For date manipulation
library(tseries) # For the adf.test function
library(uroot)
library(lmtest)


# Load data from CSV file
data <- read_csv("StatsForDataScience-main/australia_energy_complete_dataset.csv")
```

Figure A1: Load packages and data

Table A1 shows the other 12 variables with detailed descriptions that are in the dataset but excluded in the dataframe since our main focus is on Date and Demand variables.

Variable	Data Type	Description
RRP	Float	Recommended Retail Price (RRP) per megawatt hour (MWh) in Australian Dollar (AUD$)
demand_pos_RRP	Float	The total amount of electricity used on the day when the price (RRP) was positive (pos)
RRP_positive	Float	The average price when the electricity price (RRP) was positive, weighted by how much electricity used at those times in megawatts (MWh). Positive prices occurs when the demand exceeds electricity supply
demand_neg_RRP	Float	The total amount of electricity used on the day when the price (RRP) was negative (neg)

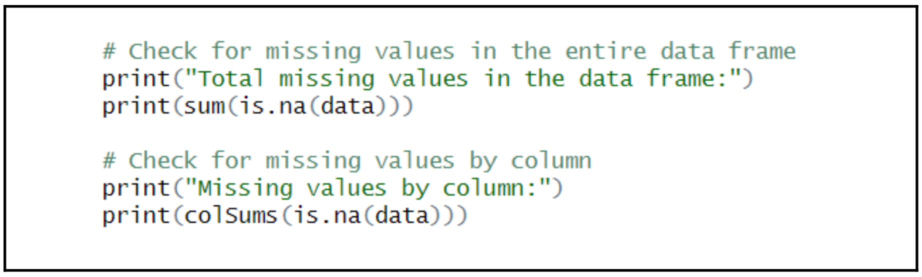
RRP_negative	Float	The average price when the electricity price (RRP) was negative, also weighted by how much electricity was used at those times in megawatts (MWh). Negative prices occurs when the electricity supply exceeds demand
frac_at_neg_RRP	Float	A fraction (portion) of the day when the electricity price was negative.
min_temperature	Float	The lowest temperature recorded during the day, measured in degrees Celsius.
max_temperature	Float	The highest temperature recorded during the day, measured in degrees Celsius.
solar_exposure	Float	Total daily sunlight energy received during the day, measured in megajoules (MJ/m^2).
rainfall	Float	The total amount of rain that fell during the day, measured in millimeters (mm).
school_day	Boolean	Value indicating whether the students were at school or not on that day.
holiday	Boolean	Value indicating whether the day was a public holiday (state or national) or not.

Table A1: Variables Excluding the Key Variables with Descriptions

Appendix B

Data Quality Assurance Process and Visualisation

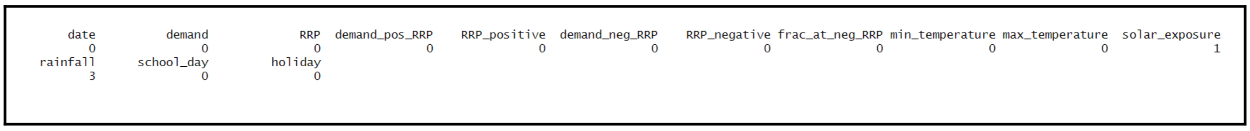
Referring to Figure B1, We first check the entire dataframe to see whether there exists any missing values in any of the columns in the dataset.



```r
# Check for missing values in the entire data frame
print("Total missing values in the data frame:")
print(sum(is.na(data)))

# Check for missing values by column
print("Missing values by column:")
print(colSums(is.na(data)))
```

Figure B1: Check missing values

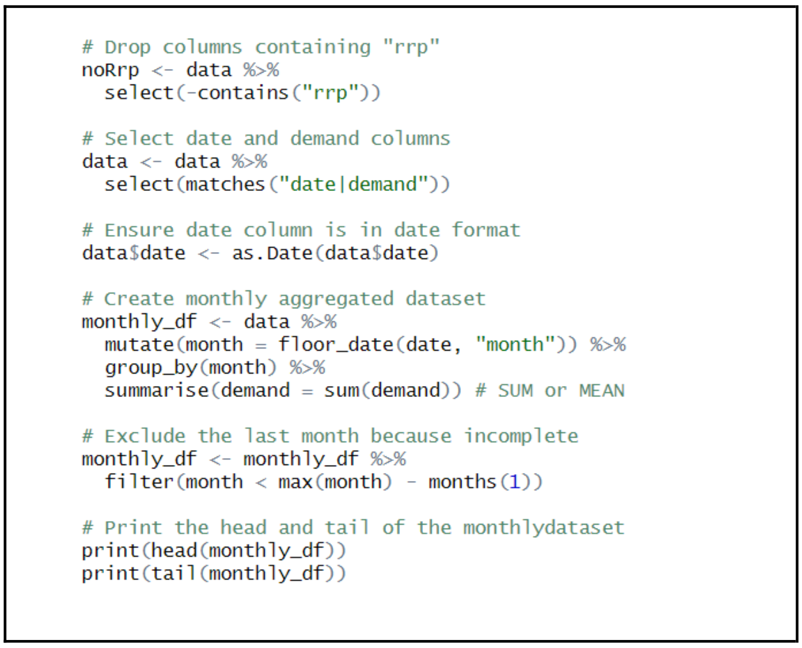
The output in Figure B2 reveals that **solar_exposure** has **1** missing value and **rainfall** has **3** missing values.



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

Figure B2: Number of missing values of each columns

We address this issue by dropping all columns related to Recommended Retail Price (RRP) and only select the key variables (Date and Demand). We also ensure all values in the Date variable are in the right format (YYYY-MM-DD) and excluding the last month.



```r
# Drop columns containing "rrp"
noRrp <- data %>%
  select(-contains("rrp"))

# Select date and demand columns
data <- data %>%
  select(matches("date|demand"))

# Ensure date column is in date format
data$date <- as.Date(data$date)

# Create monthly aggregated dataset
monthly_df <- data %>%
  mutate(month = floor_date(date, "month")) %>%
  group_by(month) %>%
  summarise(demand = sum(demand)) # SUM or MEAN

# Exclude the last month because incomplete
monthly_df <- monthly_df %>%
  filter(month < max(month) - months(1))

# Print the head and tail of the monthlydataset
print(head(monthly_df))
print(tail(monthly_df))
```

Figure B3: Code snippet to ensure quality of the data

To confirm that the dataset is clean with only relevant variables selected and observations in the Date variable are in the right format, we print the head (first 6 observations) together with tail (last 6 observations) of the dataframe.

Head of Monthly Dataset	**Tail of Monthly Dataset**
`  month        demand`	`  month        demand`
`  <date>        <dbl>`	`  <date>        <dbl>`
`1 2015-01-01 3737883.`	`1 2020-03-01 3370007.`
`2 2015-02-01 3605115.`	`2 2020-04-01 3229426.`
`3 2015-03-01 3649652.`	`3 2020-05-01 3657557.`
`4 2015-04-01 3588198.`	`4 2020-06-01 3880420.`
`5 2015-05-01 3877235.`	`5 2020-07-01 4102915.`
`6 2015-06-01 4024138.`	`6 2020-08-01 3827747.`

Table B1: Output of Head and Tail of monthly demand over time

Satisfied with the result, we then safely move on to plot a basic time series plot to provide us insights into the data's structure and behavior as can be seen in Figure B4 with the time series plot shown in Figure B5 .

```
# Plot the monthly aggregated time series
ggplot(monthly_df, aes(x = month, y = demand)) +
  geom_line() +
  ggtitle("Monthly Demand Over Time") +
  labs(x = "Date", y = "Energy Demand")
```

Figure B4: Time Series Plot

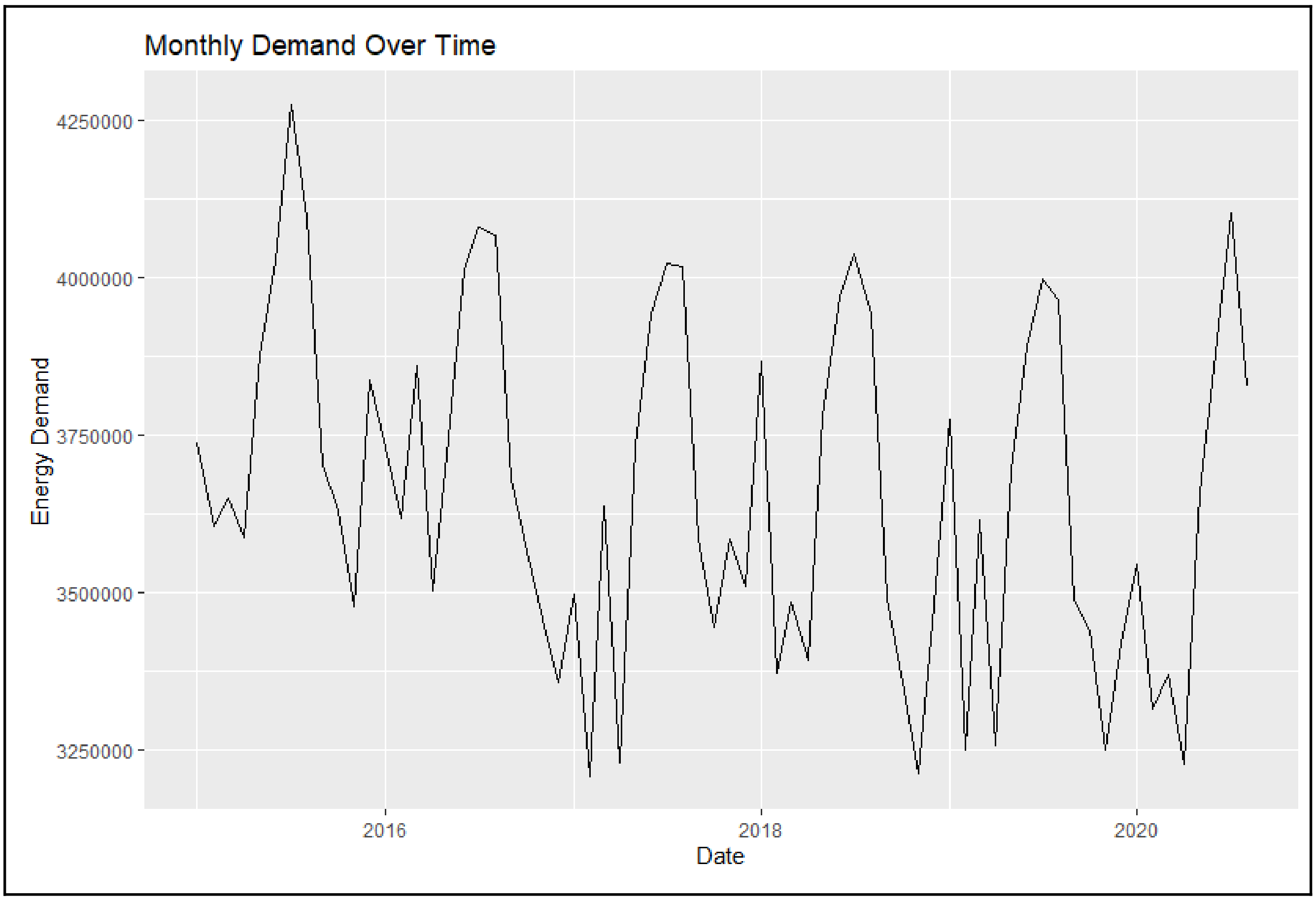
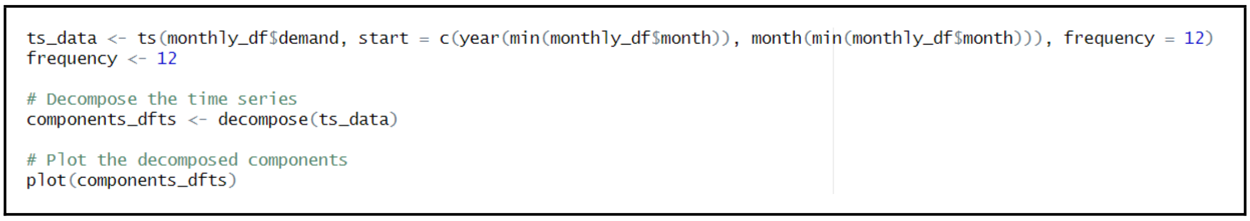


Figure B5: Time series plot of monthly demand

Appendix C

Exploratory Data Analysis (EDA) Process

For the process of Exploratory Data Analysis (EDA), we first convert the data into a time series object and decompose it into its trend, seasonal and random components as can be shown in Figure C1 below:



```
ts_data <- ts(monthly_df$demand, start = c(year(min(monthly_df$month)), month(min(monthly_df$month))), frequency = 12)
frequency <- 12

# Decompose the time series
components_dfts <- decompose(ts_data)

# Plot the decomposed components
plot(components_dfts)
```

Figure C1: Decomposition of time series

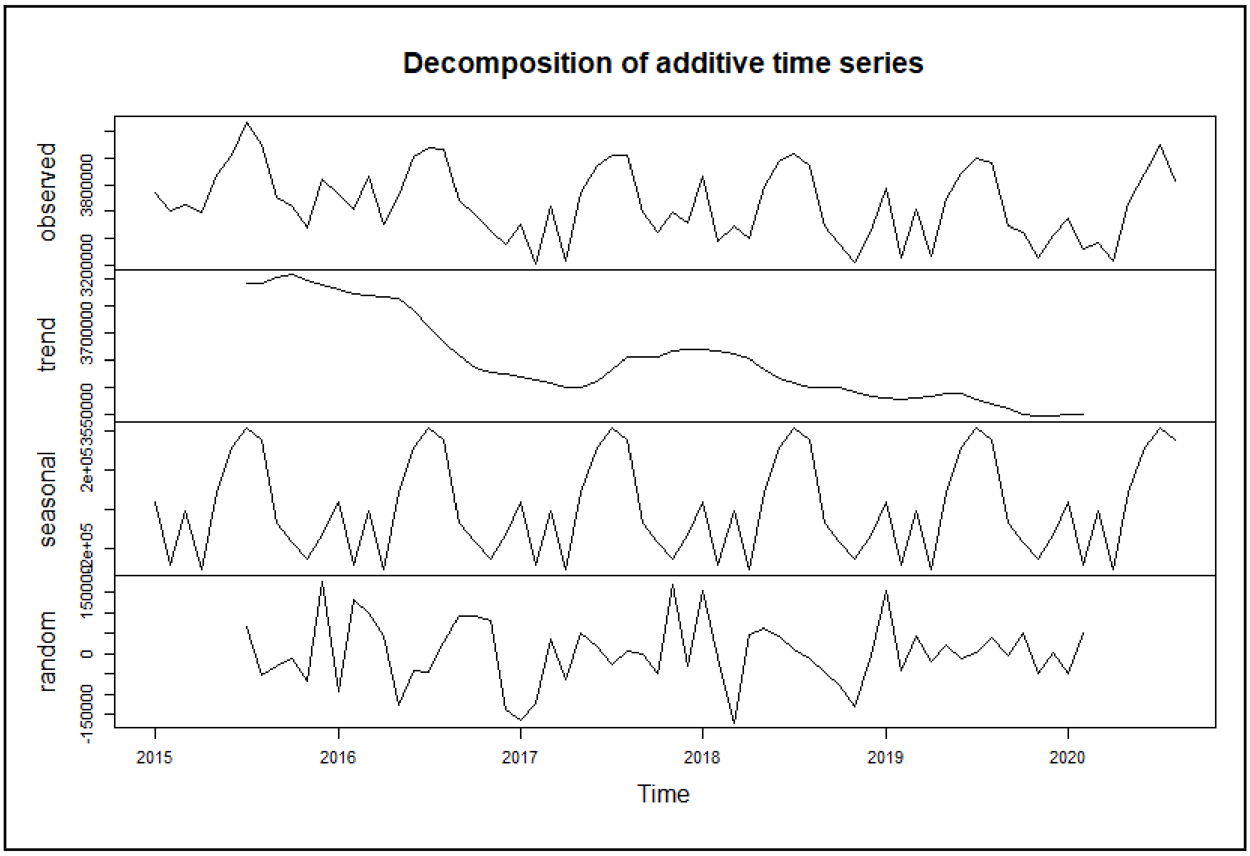


Figure C2: Decomposition of Additive Time Series

We then proceed to plot seasonality to visualize seasonal patterns in order for us to understand when peaks and dips usually occur. This could also help us when we can expect high or low demand.

```
# Seasonal Plots:
ggseasonplot(ts_data, year.labels = TRUE, year.labels.left = TRUE) +
  ylab("Energy Demand") +
  ggtitle("Seasonal plot: Australia Energy Demand by Month")
```

Figure C3: Code for Seasonal Plot

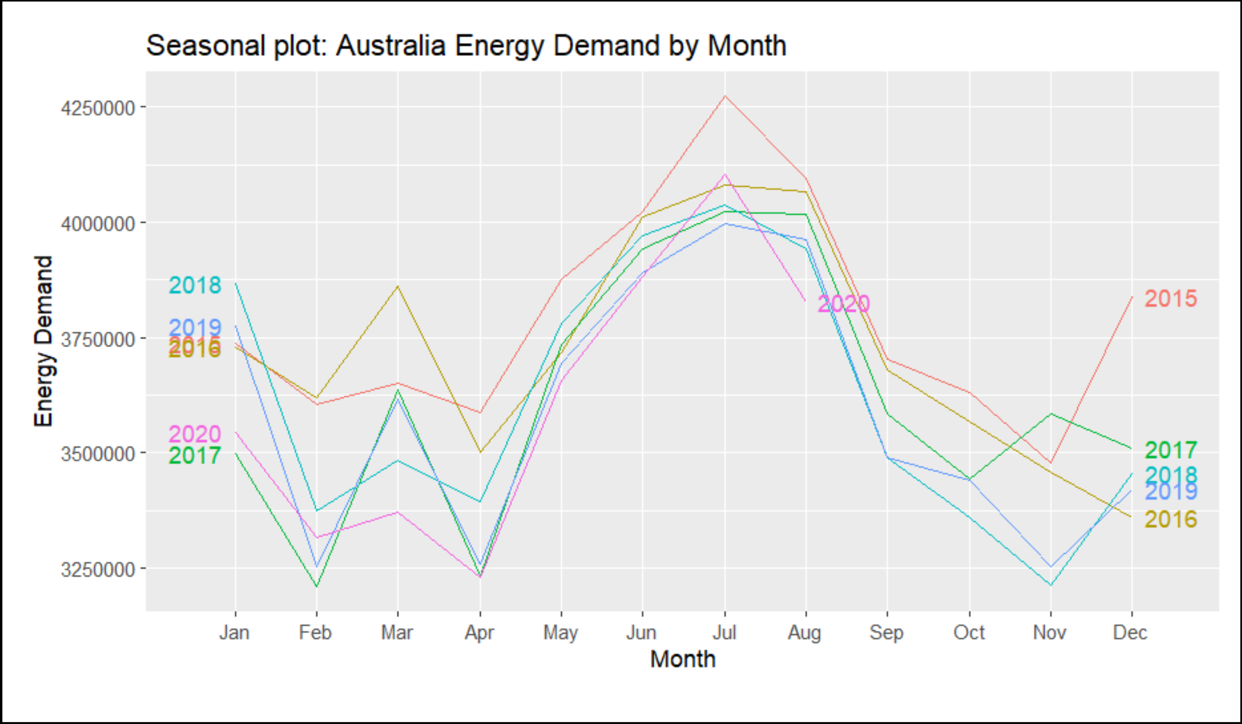


Figure C4: Seasonal Plot of Energy Demand by month

We also plot Seasonal Subseries to see how the data looks by month. We can also see how the demand in each month varies across different years with this plot.

```
# Plot the seasonal subseries plot using ggsubseriesplot
ggsubseriesplot(ts_data) +
  ylab("Energy Demand") +
  ggtitle("Seasonal subseries plot: Australia Energy Demand by Month")
```

Figure C5: Code to Plot Seasonal Subseries

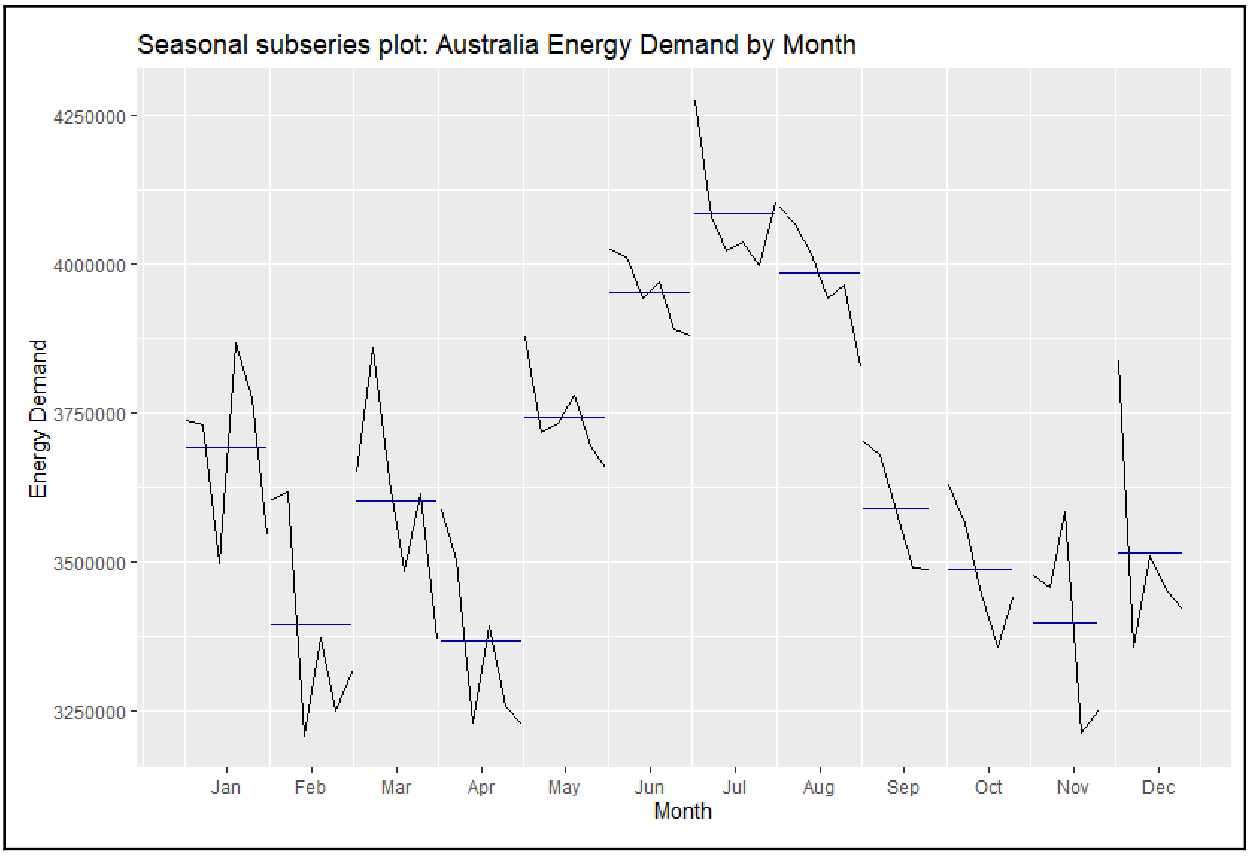
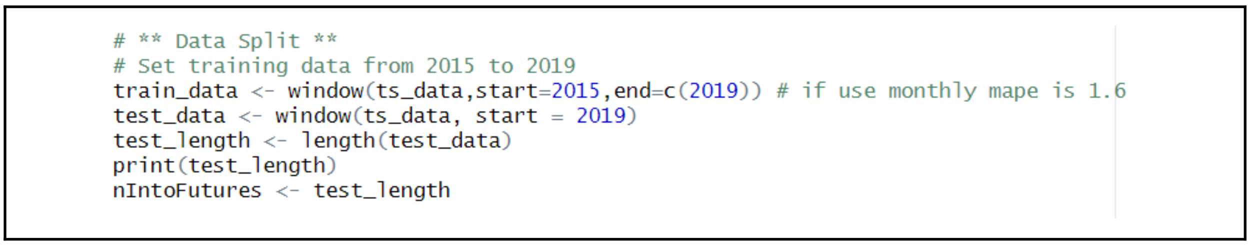


Figure C6:Subseries plot of monthly data

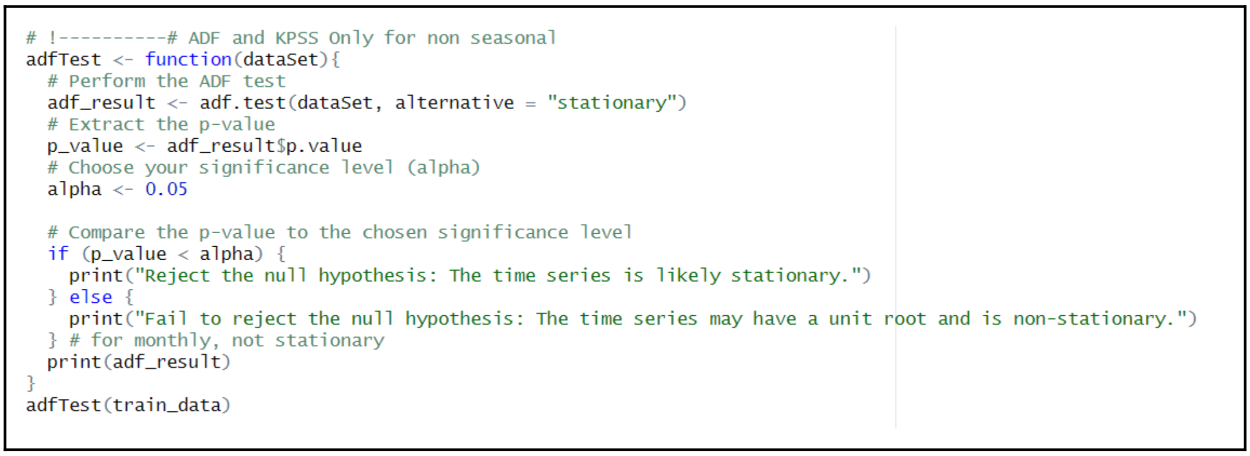
Splitting the dataset into training and testing is needed for us to use it to fit the model as well as to evaluate the model's performance. Based on Figure C7, we split the training data from the year 2015 to 2019 and the data in the year 2020 is for testing usage.



```
# ** Data Split **
# Set training data from 2015 to 2019
train_data <- window(ts_data,start=2015,end=c(2019)) # if use monthly mape is 1.6
test_data <- window(ts_data, start = 2019)
test_length <- length(test_data)
print(test_length)
nIntoFutures <- test_length
```

Figure C7: Splitting Data into Training and Testing

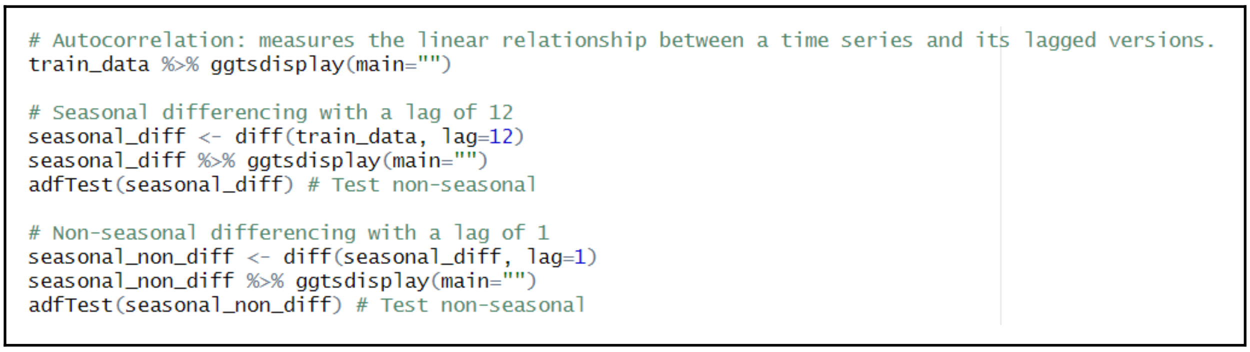
After that, we check whether the time series is stationary with the ADF test as can be seen in Figure C8.



```
# !----------# ADF and KPSS Only for non seasonal
adfTest <- function(dataSet){
  # Perform the ADF test
  adf_result <- adf.test(dataSet, alternative = "stationary")
  # Extract the p-value
  p_value <- adf_result$p.value
  # Choose your significance level (alpha)
  alpha <- 0.05

  # Compare the p-value to the chosen significance level
  if (p_value < alpha) {
    print("Reject the null hypothesis: The time series is likely stationary.")
  } else {
    print("Fail to reject the null hypothesis: The time series may have a unit root and is non-stationary.")
  } # for monthly, not stationary
  print(adf_result)
}
adfTest(train_data)
```

Figure C8: adf test in practical

The relationship between time series and its lagged values are measured with Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF). These could help us to identify parameters for our models like ARIMA



```
# Autocorrelation: measures the linear relationship between a time series and its lagged versions.
train_data %>% ggtsdisplay(main="")

# Seasonal differencing with a lag of 12
seasonal_diff <- diff(train_data, lag=12)
seasonal_diff %>% ggtsdisplay(main="")
adfTest(seasonal_diff) # Test non-seasonal

# Non-seasonal differencing with a lag of 1
seasonal_non_diff <- diff(seasonal_diff, lag=1)
seasonal_non_diff %>% ggtsdisplay(main="")
adfTest(seasonal_non_diff) # Test non-seasonal
```

Figure C9: ACF and PACF plot method

Appendix D

Description of Model Candidates

ARIMA:

Autoregressive Integrated Moving Average (ARIMA), a class of time series prediction modeling technique. It is denoted as ARIMA(p,d,q) where p is the non-seasonal autoregressive term d is the degree of non-seasonal differencing and q is the non-seasonal moving average terms. Arima models require stationary time series usually achieved by doing differencing. Stationarity can be checked using the Augmented Dickey-Fuller (ADF) or Kwiatkowski–Phillips–Schmidt–Shin (KPSS) Tests.Due to the seasonality in the data, SARIMA(p,d,q)(P,D,Q)m model is suggested. P, D, Q have the same function as the lowercase same alphabet but it's used to deal with seasonality of the data, and m stands for number of time steps of one seasonal period. For our models, we use m = 12 as it's a monthly data.

AUTO-ARIMA:

Auto ARIMA is an automated version of the **ARIMA** model that simplifies the model selection process by automatically identifying the best combination of **p** (autoregressive term), **d** (degree of differencing), and **q** (moving average term) based on the data. It uses information criteria such as **AICc** to select the model that best fits the data while avoiding overfitting. The Auto ARIMA model also handles **seasonality** by automatically identifying the seasonal parameters **P**, **D**, and **Q** for the **SARIMA** model. This makes Auto ARIMA highly useful when dealing with time series data with both non-seasonal and seasonal components, as it reduces the need for manual parameter tuning.

Holt-Winters (Triple Exponential Smoothing):

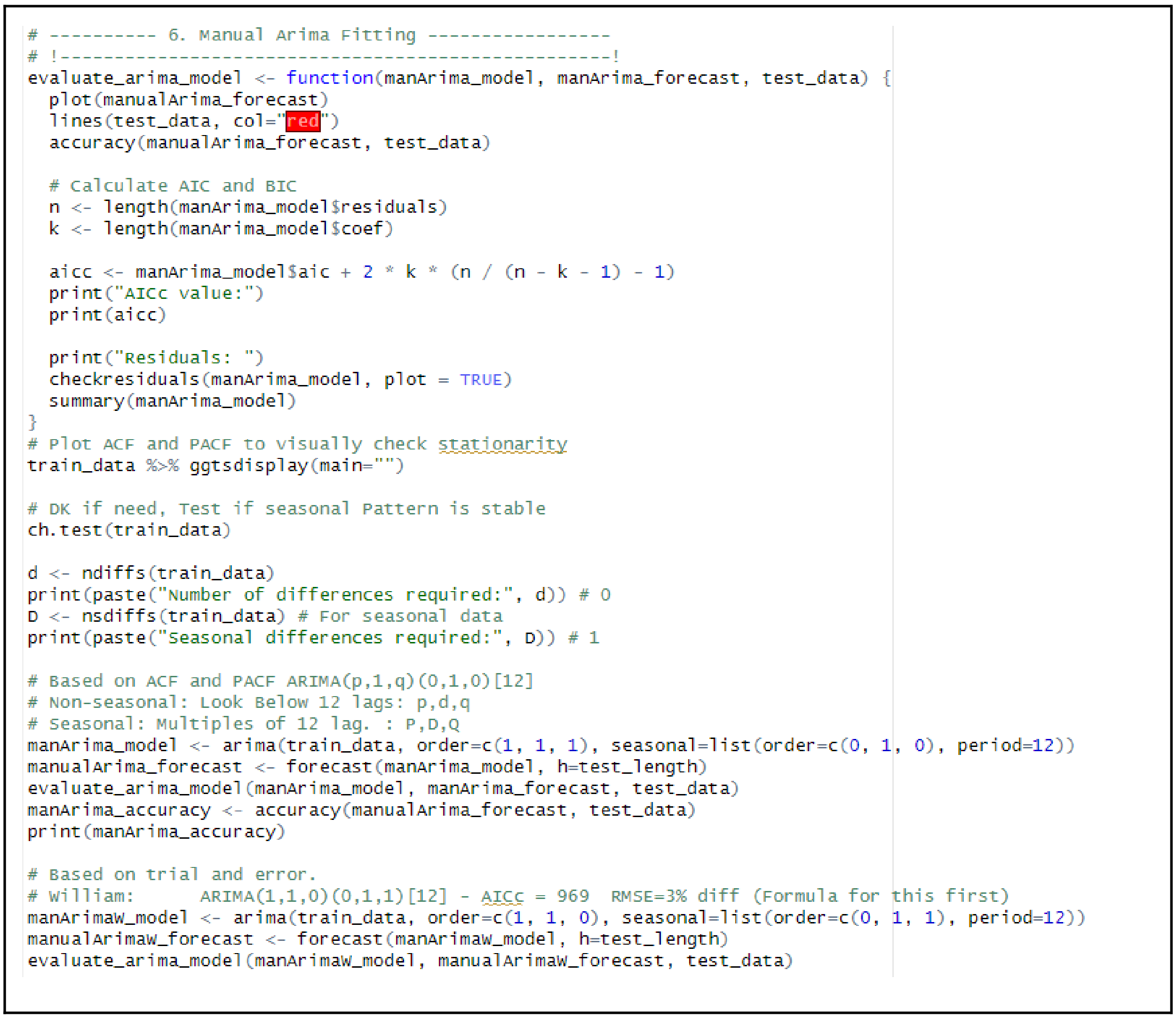
The **Holt-Winters Exponential Smoothing** model is another time series forecasting method used for data with **trend** and **seasonality**. It consists of three components: **level**, **trend**, and **seasonality**. This method adjusts the forecasted values based on recent changes in level, trend , and seasonality. Holt-Winters comes in two variations: **additive** and **multiplicative** models. In the **additive model**, seasonal variations are constant over time, while in the **multiplicative model**, seasonal variations change proportionally to the level of the time series. Holt-Winters is well-suited for data that exhibits both seasonal and trend components, and it performs well in providing smooth forecasts.

Seasonal Naive Model:

The **Seasonal Naive Model** is a simple forecasting technique used for data with **seasonal patterns**. In this model, the forecast for a given time period is set to be the same as the value from the same period in the previous season. For instance, in monthly data, the forecast for March 2019 would be the same as the actual value from March 2018. This method assumes that the data exhibits a **strong seasonal pattern** that repeats exactly from year to year. While the Seasonal Naive Model is straightforward and easy to implement, it does not account for trends or changes over time, making it most effective when the seasonality is strong and consistent.

Appendix E

Model Evaluation Code



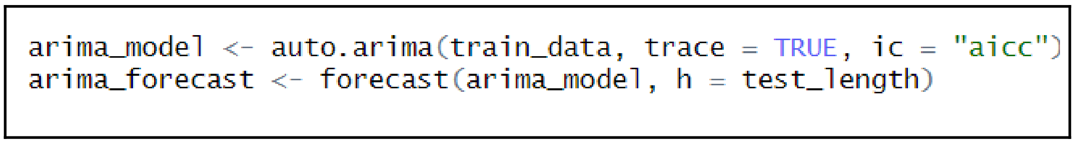
```r
# ---------- 6. Manual Arima Fitting -----------------
# !--------------------------------------------------!
evaluate_arima_model <- function(manArima_model, manArima_forecast, test_data) {
  plot(manualArima_forecast)
  lines(test_data, col="red")
  accuracy(manualArima_forecast, test_data)

  # Calculate AIC and BIC
  n <- length(manArima_model$residuals)
  k <- length(manArima_model$coef)

  aicc <- manArima_model$aic + 2 * k * (n / (n - k - 1) - 1)
  print("AICc value:")
  print(aicc)

  print("Residuals: ")
  checkresiduals(manArima_model, plot = TRUE)
  summary(manArima_model)
}
# Plot ACF and PACF to visually check stationarity
train_data %>% ggtsdisplay(main="")

# DK if need, Test if seasonal Pattern is stable
ch.test(train_data)

d <- ndiffs(train_data)
print(paste("Number of differences required:", d)) # 0
D <- nsdiffs(train_data) # For seasonal data
print(paste("Seasonal differences required:", D)) # 1

# Based on ACF and PACF ARIMA(p,1,q)(0,1,0)[12]
# Non-seasonal: Look Below 12 lags: p,d,q
# Seasonal: Multiples of 12 lag. : P,D,Q
manArima_model <- arima(train_data, order=c(1, 1, 1), seasonal=list(order=c(0, 1, 0), period=12))
manualArima_forecast <- forecast(manArima_model, h=test_length)
evaluate_arima_model(manArima_model, manArima_forecast, test_data)
manArima_accuracy <- accuracy(manualArima_forecast, test_data)
print(manArima_accuracy)

# Based on trial and error.
# William:      ARIMA(1,1,0)(0,1,1)[12] - AICc = 969  RMSE=3% diff (Formula for this first)
manArimaW_model <- arima(train_data, order=c(1, 1, 0), seasonal=list(order=c(0, 1, 1), period=12))
manualArimaW_forecast <- forecast(manArimaW_model, h=test_length)
evaluate_arima_model(manArimaW_model, manualArimaW_forecast, test_data)
```

Figure E1: Screenshot of Manual Arima Model accuracy code{accuracy Function part}

Appendix F

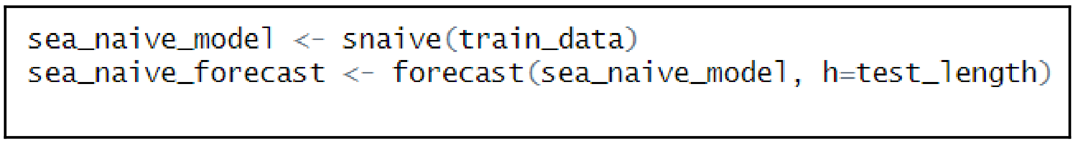
2nd to 4th Model Fitting Reference

2nd Model Fitting: Auto ARIMA(1,0,0)(0,1,1)[12] with drift



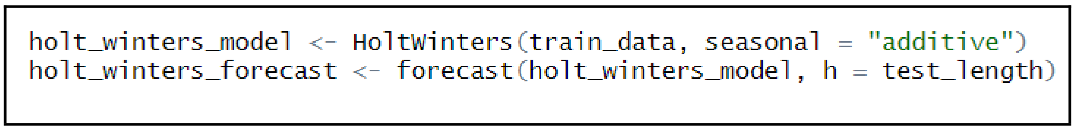
```
arima_model <- auto.arima(train_data, trace = TRUE, ic = "aicc")
arima_forecast <- forecast(arima_model, h = test_length)
```

3rd Model Fitting: Seasonal Naive



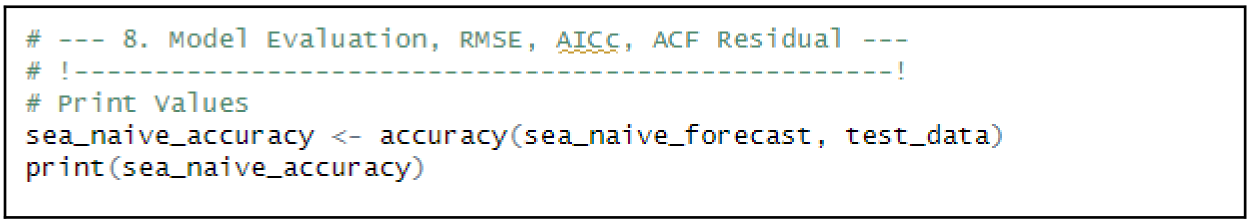
```
sea_naive_model <- snaive(train_data)
sea_naive_forecast <- forecast(sea_naive_model, h=test_length)
```

4th Model Fitting: Holt-Winters



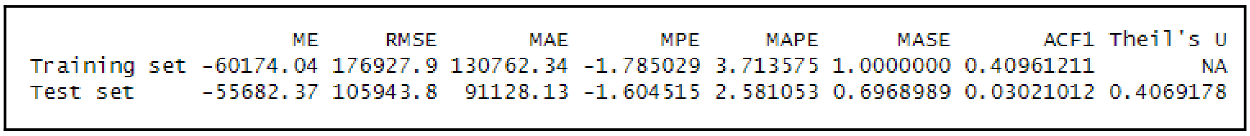
```
holt_winters_model <- HoltWinters(train_data, seasonal = "additive")
holt_winters_forecast <- forecast(holt_winters_model, h = test_length)
```

Figure F1 : Screenshot of Other Models accuracy code{accuracy Function part}



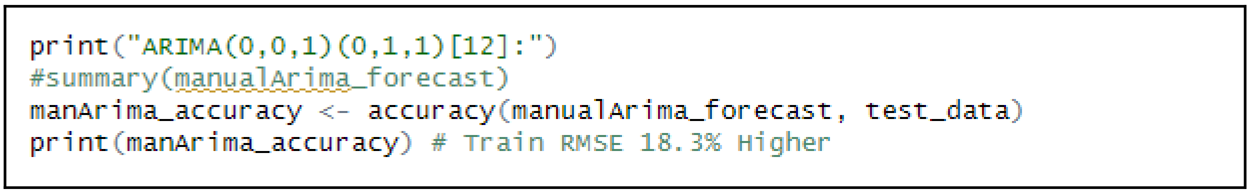
```
# --- 8. Model Evaluation, RMSE, AICc, ACF Residual ---
# !--------------------------------------------------!
# Print Values
sea_naive_accuracy <- accuracy(sea_naive_forecast, test_data)
print(sea_naive_accuracy)
```

Figure F2: seasonal naive accuracy calculation



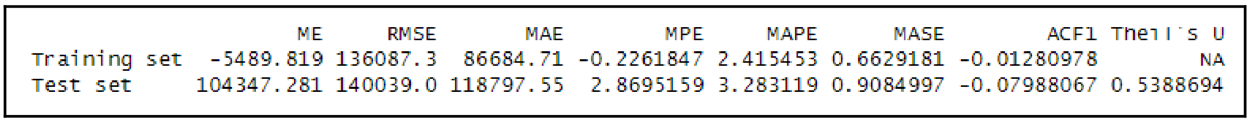
```
                    ME     RMSE       MAE       MPE     MAPE      MASE       ACF1 Theil's U
Training set -60174.04 176927.9 130762.34 -1.785029 3.713575 1.0000000 0.40961211        NA
Test set     -55682.37 105943.8  91128.13 -1.604515 2.581053 0.6968989 0.03021012 0.4069178
```

Figure F3: seasonal naive accuracy check



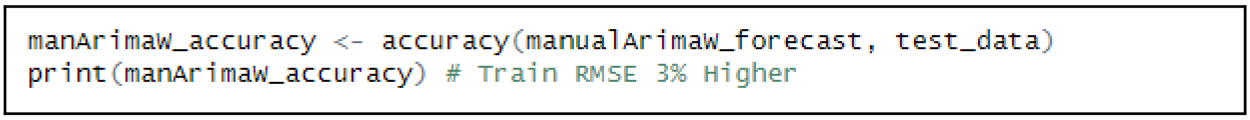
```
print("ARIMA(0,0,1)(0,1,1)[12]:")
#summary(manualArima_forecast)
manArima_accuracy <- accuracy(manualArima_forecast, test_data)
print(manArima_accuracy) # Train RMSE 18.3% Higher
```

Figure F4: Arima (0,0,1)(0,1,1)[12] accuracy calculation



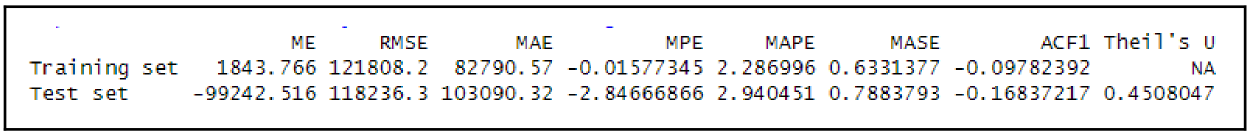
```
                      ME      RMSE       MAE        MPE     MAPE      MASE        ACF1 Theil's U
Training set   -5489.819 136087.3  86684.71 -0.2261847 2.415453 0.6629181 -0.01280978        NA
Test set      104347.281 140039.0 118797.55  2.8695159 3.283119 0.9084997 -0.07988067 0.5388694
```

Figure F5: Arima (0,0,1)(0,1,1)[12] accuracy check



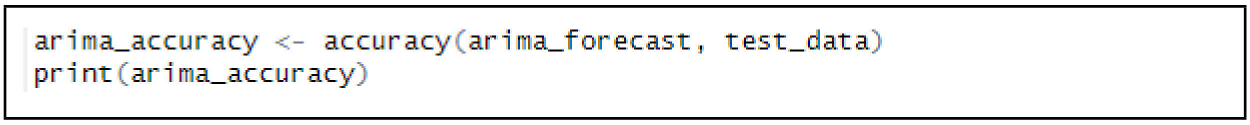
```
manArimaW_accuracy <- accuracy(manualArimaW_forecast, test_data)
print(manArimaW_accuracy) # Train RMSE 3% Higher
```

Figure F6: Arima (1,1,1)(0,1,0)[12] accuracy calculation



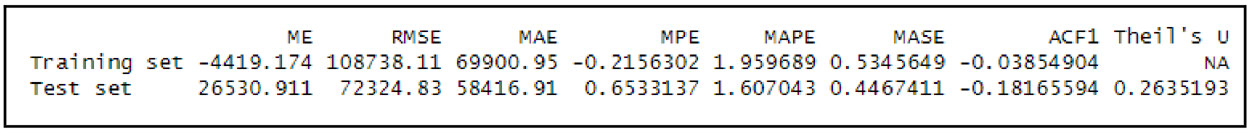
```
                     ME     RMSE       MAE         MPE     MAPE      MASE        ACF1 Theil's U
Training set   1843.766 121808.2  82790.57 -0.01577345 2.286996 0.6331377 -0.09782392        NA
Test set     -99242.516 118236.3 103090.32 -2.84666866 2.940451 0.7883793 -0.16837217 0.4508047
```

Figure F7: Arima (1,1,1)(0,1,0)[12] accuracy check



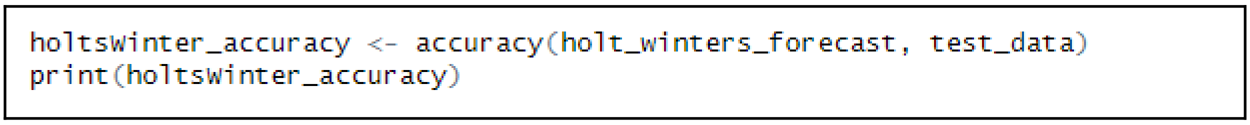
```
arima_accuracy <- accuracy(arima_forecast, test_data)
print(arima_accuracy)
```

Figure F8: auto-arima accuracy calculation



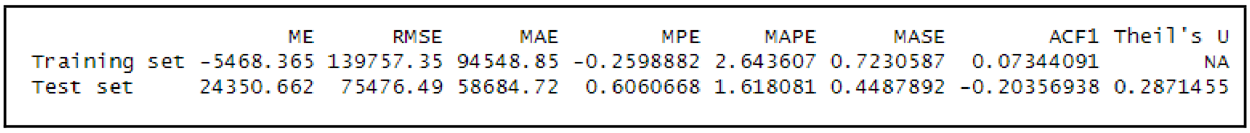
```
                    ME      RMSE      MAE        MPE     MAPE      MASE        ACF1 Theil's U
Training set -4419.174 108738.11 69900.95 -0.2156302 1.959689 0.5345649 -0.03854904        NA
Test set     26530.911  72324.83 58416.91  0.6533137 1.607043 0.4467411 -0.18165594 0.2635193
```

Figure F9: auto-arima accuracy check



```
holtsWinter_accuracy <- accuracy(holt_winters_forecast, test_data)
print(holtsWinter_accuracy)
```

Figure F10: holt-winters accuracy calculation

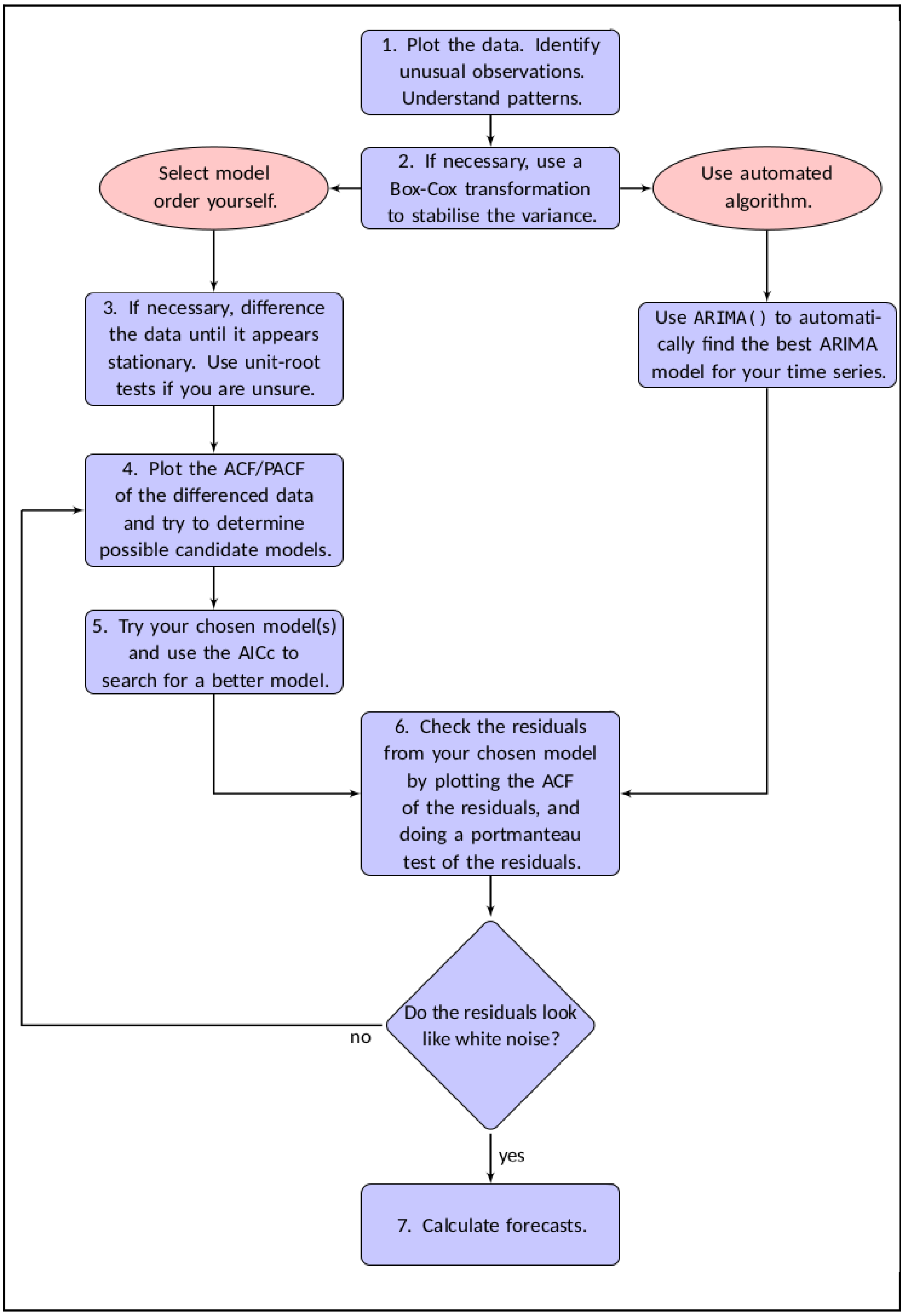


```
                    ME      RMSE      MAE        MPE     MAPE      MASE        ACF1 Theil's U
Training set -5468.365 139757.35 94548.85 -0.2598882 2.643607 0.7230587  0.07344091        NA
Test set     24350.662  75476.49 58684.72  0.6060668 1.618081 0.4487892 -0.20356938 0.2871455
```

Figure F11: holt-winters accuracy check

Appendix G

Arima Flow Chart



1. Plot the data. Identify unusual observations. Understand patterns.
2. If necessary, use a Box-Cox transformation to stabilise the variance.
Select model order yourself.
Use automated algorithm.
3. If necessary, difference the data until it appears stationary. Use unit-root tests if you are unsure.
Use ARIMA() to automatically find the best ARIMA model for your time series.
4. Plot the ACF/PACF of the differenced data and try to determine possible candidate models.
5. Try your chosen model(s) and use the AICc to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals.
Do the residuals look like white noise?
no
yes
7. Calculate forecasts.

Figure G1: ARIMA Flowchart

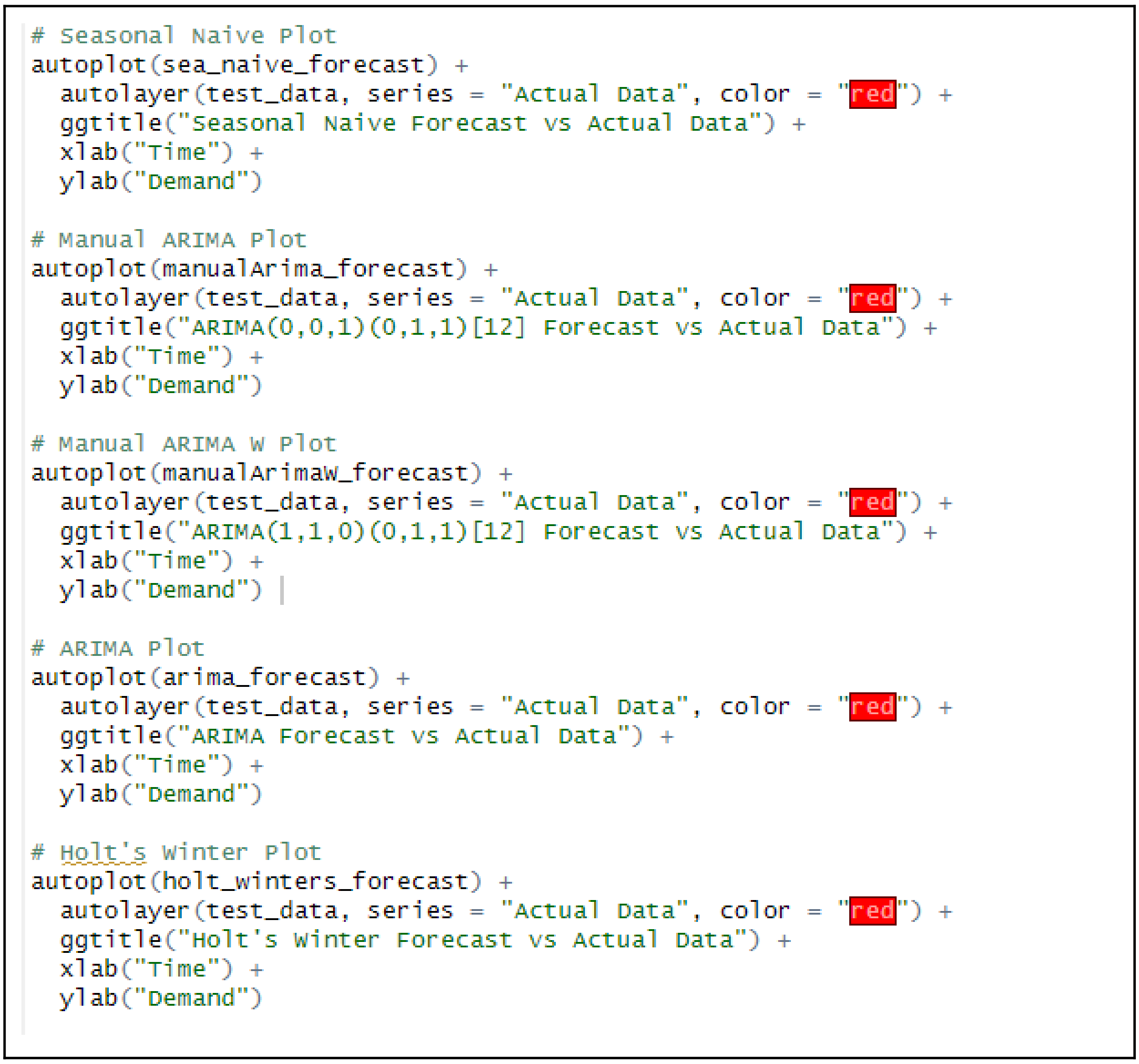
Formula for RMSE Difference in percentage

$$\boxed{\text{RMSE Difference (\%)} = \left(\frac{\text{RMSE}_{\text{Train}} - \text{RMSE}_{\text{Test}}}{\text{RMSE}_{\text{Train}}}\right) \times 100}$$

Figure G2: Formula for RMSE Difference in percentage

Appendix H

2nd to 4th Model Forecasting Reference



```r
# Seasonal Naive Plot
autoplot(sea_naive_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("Seasonal Naive Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Manual ARIMA Plot
autoplot(manualArima_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(0,0,1)(0,1,1)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Manual ARIMA W Plot
autoplot(manualArimaW_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(1,1,0)(0,1,1)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# ARIMA Plot
autoplot(arima_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Holt's Winter Plot
autoplot(holt_winters_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("Holt's Winter Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")
```

Figure H1: Model plots

2nd Model Forecasting: Auto ARIMA(1,0,0)(0,1,1)[12] with drift

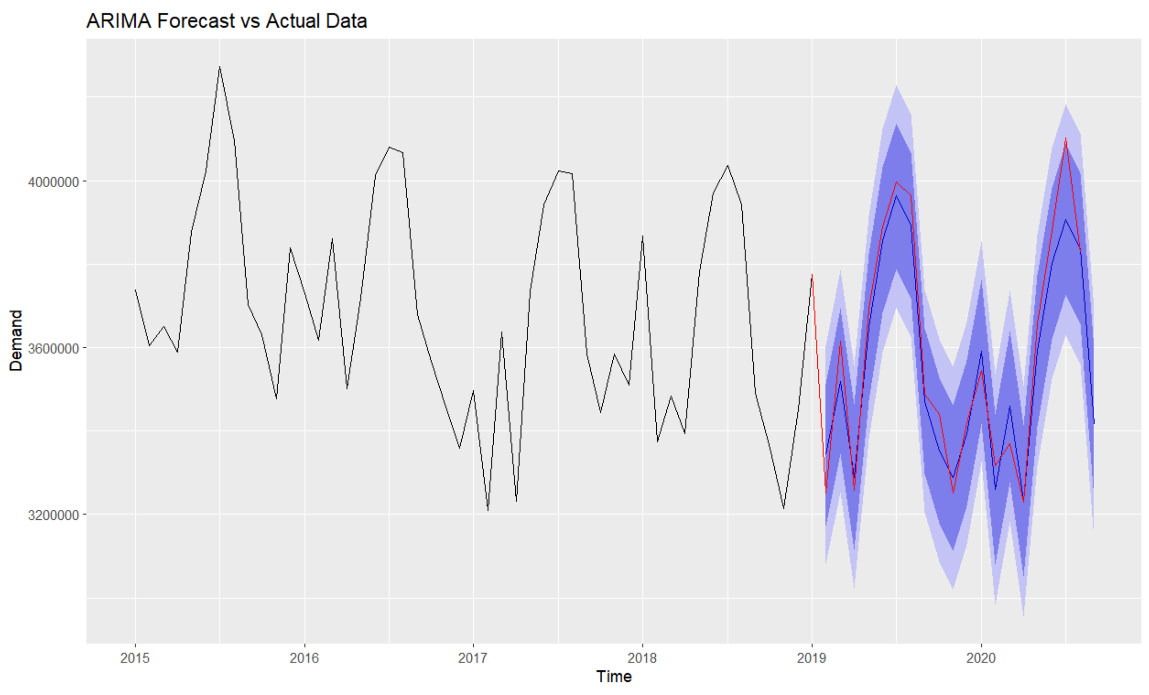


Figure H2: Auto-Arima prediction plot

3rd Model Forecasting: Seasonal Naive

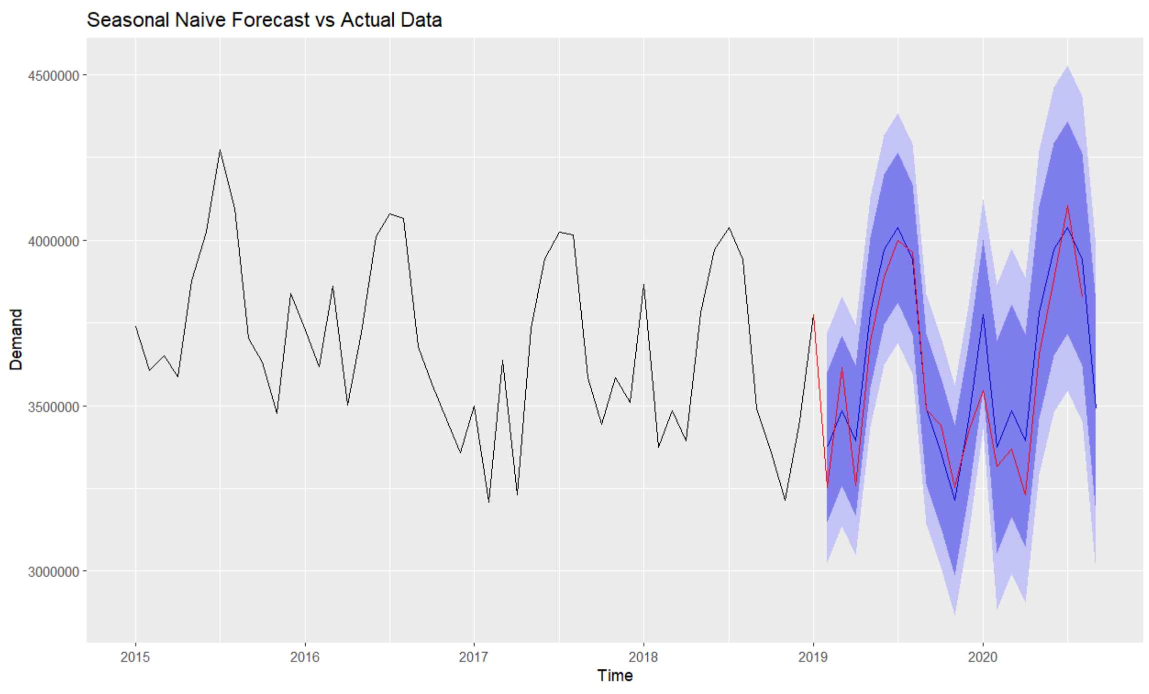


Figure H3: Seasonal Naive model prediction plot

4th Model Forecasting: Holt-Winters

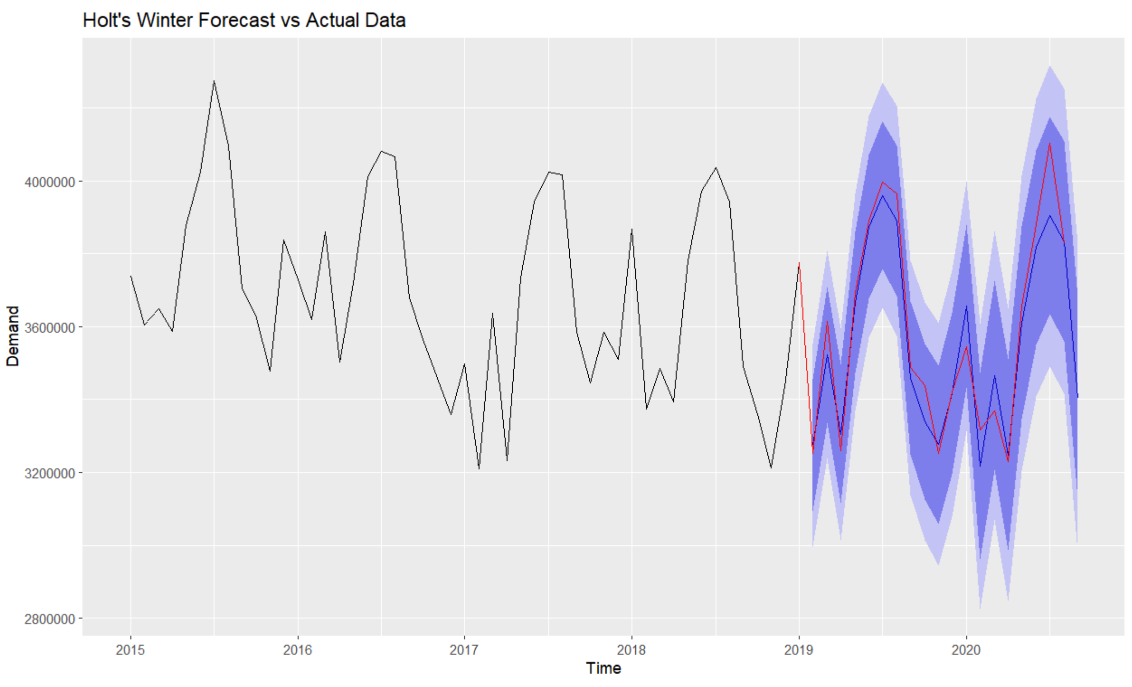


Figure H4: Holt-winters model prediction plot

5th Model Forecasting: ARIMA (0,0,1)(0,1,1)[12]

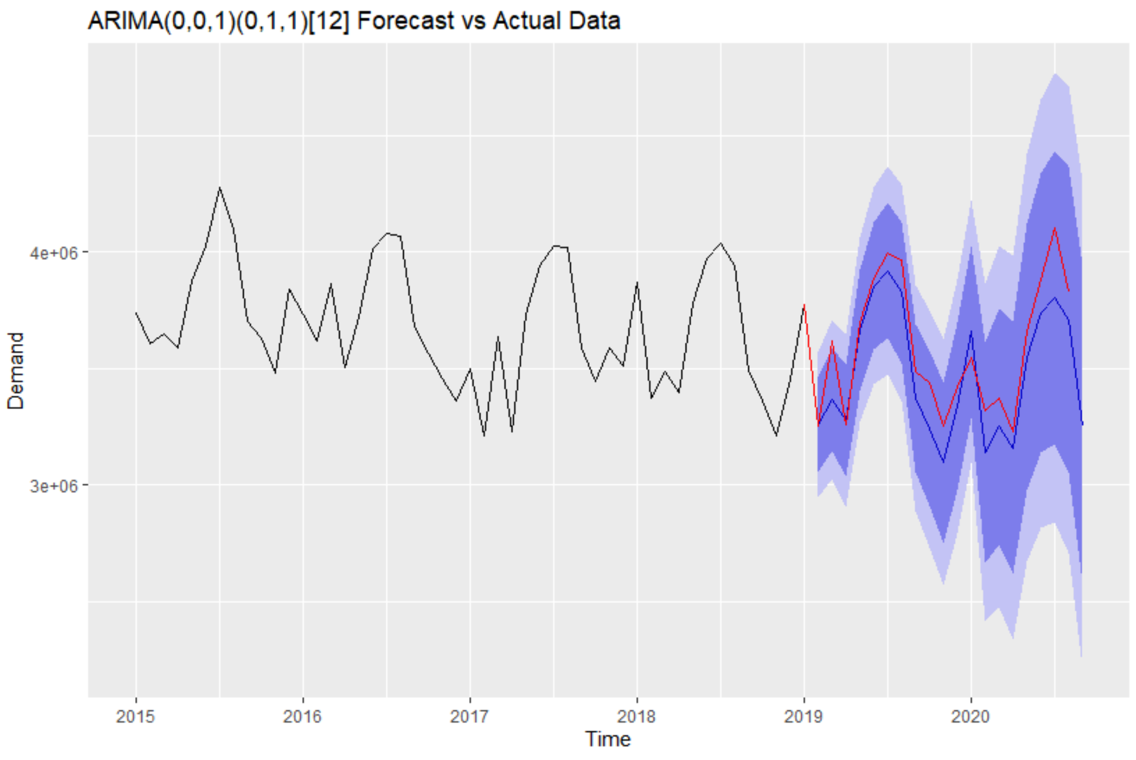
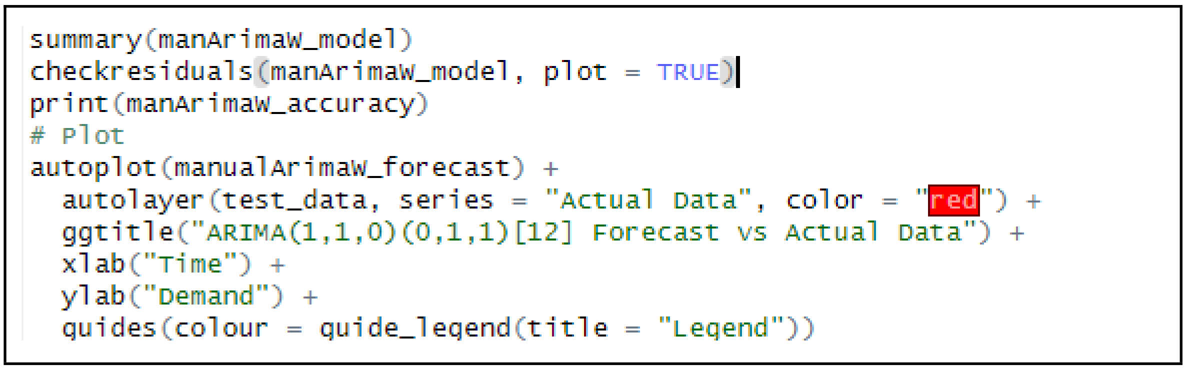


Figure H5: Arima (0,0,1)(0,1,1)[12] prediction plot



```
summary(manArimaW_model)
checkresiduals(manArimaW_model, plot = TRUE)
print(manArimaW_accuracy)
# Plot
autoplot(manualArimaW_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(1,1,0)(0,1,1)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand") +
  guides(colour = guide_legend(title = "Legend"))
```

Figure H6: Best model {ARIMA(1,1,0)(0,1,1)[12]}

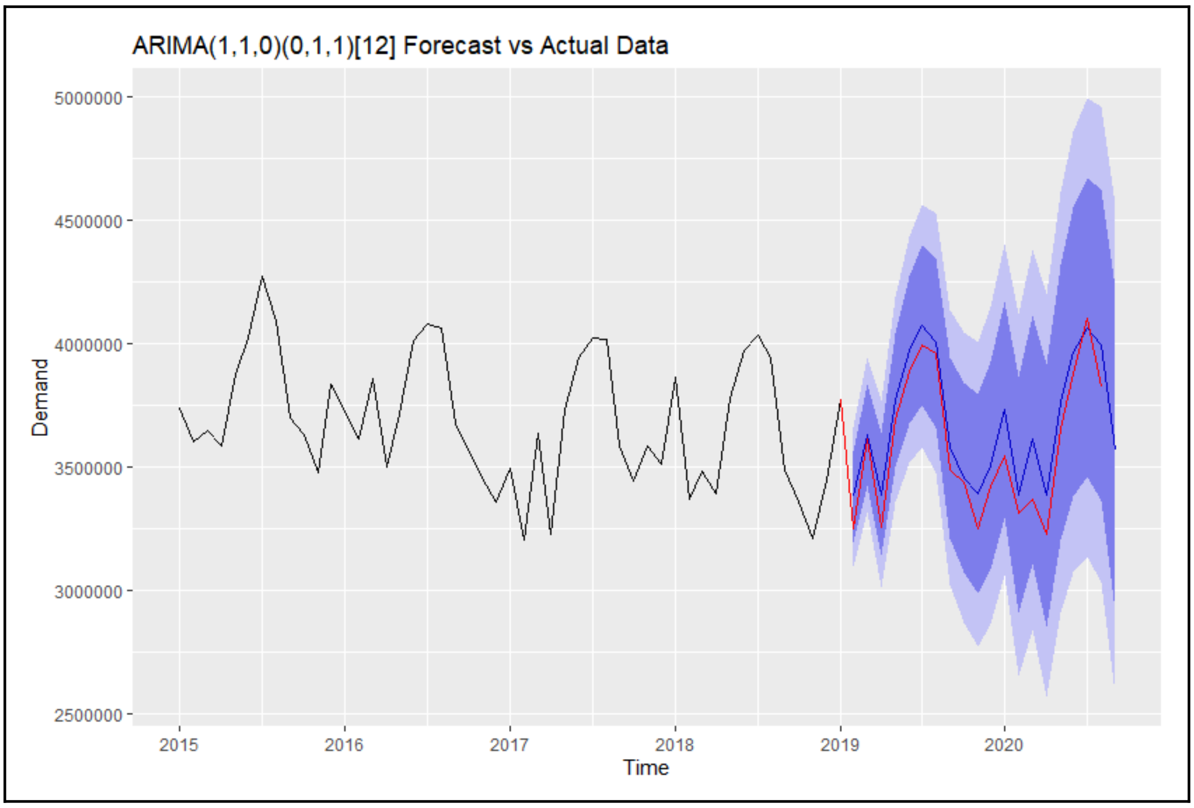


Figure H7: ARIMA(1,1,0)(0,1,1)[12] prediction plot

Appendix I

1 Seasonal and 1 Non-Seasonal Differencing

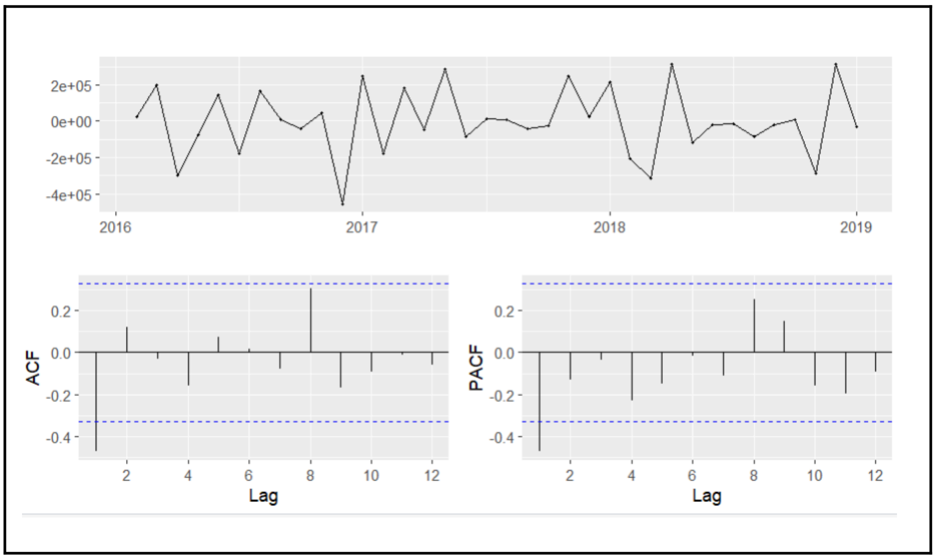


Figure I1: plot after 1 seasonal and 1 non-seasonal diff

Appendix J

Code

```
# Steps: (Using MONTHLY DATA for models)
# !---------------------------------------------------!
# ---------- 0. Import Dataset, filter etc -----------
# !---------------------------------------------------!
# Load necessary packages
library(readr)
library(dplyr)
library(ggplot2)
library(forecast)
library(lubridate) # For date manipulation
library(tseries) # For the adf.test function
library(uroot)
library(lmtest)


# Load data from CSV file
data <-
read_csv("C:/Users/wbrya/OneDrive/Documents/australia_energy_comple
te_dataset.csv")

# Check for missing values in the entire data frame
print("Total missing values in the data frame:")
print(sum(is.na(data)))

# Check for missing values by column
print("Missing values by column:")
print(colSums(is.na(data)))
# Missing values in columns rainfall and solar_exposure, we will
not be using these anyways.

# Drop columns containing "rrp"
noRrp <- data %>%
  select(-contains("rrp"))

# Select date and demand columns
data <- data %>%
  select(matches("date|demand"))

# Ensure date column is in date format
data$date <- as.Date(data$date)

# Create a data frame with dates and demand for the daily time
series
daily_df <- data.frame(date = data$date, demand = data$demand)

# Create monthly aggregated dataset
monthly_df <- data %>%
  mutate(month = floor_date(date, "month")) %>%
  group_by(month) %>%
  summarise(demand = sum(demand)) # SUM or MEAN

# Exclude the last month because incomplete
monthly_df <- monthly_df %>%
  filter(month < max(month) - months(1))

# Print the head of the monthly and daily dataset
print(head(monthly_df))
print(tail(monthly_df))







# !-----------------------------------------------------!
# ---------- 1. Time Series plot ---------------------
# !-----------------------------------------------------!
# Plot the monthly aggregated time series
ggplot(monthly_df, aes(x = month, y = demand)) +
  geom_line() +
  ggtitle("Monthly Demand Over Time") +
  labs(x = "Date", y = "Energy Demand")
```

```
# !--------------------------------------------------!
# ---------- 2. Convert to time series object --------
# !--------------------------------------------------!
ts_data <- ts(monthly_df$demand, start =
c(year(min(monthly_df$month)), month(min(monthly_df$month))),
frequency = 12)
frequency <- 12

# Decompose the time series
components_dfts <- decompose(ts_data)

# Plot the decomposed components
plot(components_dfts)

# Seasonal Plots:
ggseasonplot(ts_data, year.labels = TRUE, year.labels.left = TRUE)
+
  ylab("Energy Demand") +
  ggtitle("Seasonal plot: Australia Energy Demand by Month")

# Plot the seasonal subseries plot using ggsubseriesplot
ggsubseriesplot(ts_data) +
  ylab("Energy Demand") +
  ggtitle("Seasonal subseries plot: Australia Energy Demand by
Month")








# !--------------------------------------------------!
# ---------- 3. Data Split --------------------------
# !--------------------------------------------------!
# Set training data from 2015 to 2019
train_data <- window(ts_data,start=2015,end=c(2019))
test_data <- window(ts_data, start = 2019)
test_length <- length(test_data)
print(test_length)
nIntoFutures <- test_length








# !--------------------------------------------------!
# ---------- 4. Stationary test: ADF test ------------
# !--------------------------------------------------!
adfTest <- function(dataSet){
  # ADF test
  adf_result <- adf.test(dataSet, alternative = "stationary")
  # Extract the p-value
  p_value <- adf_result$p.value
  # Choose significance level
  alpha <- 0.05

  # Compare the p-value
  if (p_value < alpha) {
    print("Reject the null hypothesis: The time series is likely
stationary.")
  } else {
    print("Fail to reject the null hypothesis: The time series may
have a unit root and is non-stationary.")
  } # for monthly, not stationary
  print(adf_result)
}
adfTest(train_data)








# !---------------------------------------------------!
# ---------- 5. Correlogram and Pacf -----------------
# !---------------------------------------------------!
# Autocorrelation: measures the linear relationship between a time
series and its lagged versions.
train_data %>% ggtsdisplay(main="")

# Seasonal differencing with a lag of 12
seasonal_diff <- diff(train_data, lag=12)
seasonal_diff %>% ggtsdisplay(main="")
adfTest(seasonal_diff) # Test non-seasonal

# Non-seasonal differencing with a lag of 1
seasonal_non_diff <- diff(seasonal_diff, lag=1)
seasonal_non_diff %>% ggtsdisplay(main="")
adfTest(seasonal_non_diff) # Test non-seasonal








# !---------------------------------------------------!
# ---------- 6. Manual Arima Fitting -----------------
# !---------------------------------------------------!
evaluate_arima_model <- function(manArima_model, manArima_forecast,
test_data) {
  plot(manualArima_forecast)
  lines(test_data, col="red")
  accuracy(manualArima_forecast, test_data)

  # Calculate AIC and BIC
  n <- length(manArima_model$residuals)
  k <- length(manArima_model$coef)

  aicc <- manArima_model$aic + 2 * k * (n / (n - k - 1) - 1)
  print("AICc value:")
  print(aicc)

  print("Residuals: ")
  checkresiduals(manArima_model, plot = TRUE)
  summary(manArima_model)
}
# Based on ACF and PACF ARIMA(p,1,q)(0,1,0)[12]
# Non-seasonal: Look Below 12 lags: p,d,q
# Seasonal: Multiples of 12 lag. : P,D,Q
manArima_model <- arima(train_data, order=c(1, 1, 1),
seasonal=list(order=c(0, 1, 0), period=12))
manualArima_forecast <- forecast(manArima_model, h=test_length)
evaluate_arima_model(manArima_model, manArima_forecast, test_data)
manArima_accuracy <- accuracy(manualArima_forecast, test_data)
print(manArima_accuracy)

# Based on trial and error.
# ARIMA(1,1,0)(0,1,1)[12] - AICc = 969  RMSE=3% diff
manArimaW_model <- arima(train_data, order=c(1, 1, 0),
seasonal=list(order=c(0, 1, 1), period=12))
manualArimaW_forecast <- forecast(manArimaW_model, h=test_length)
evaluate_arima_model(manArimaW_model, manualArimaW_forecast,
test_data)







# !----------------------------------------------------!
# ---------- 7. Other Model fit ----------------------
# !----------------------------------------------------!
# 1. Seasonal Naive
sea_naive_model <- snaive(train_data)
sea_naive_forecast <- forecast(sea_naive_model, h=test_length)

# 2. Fit ARIMA model
arima_model <- auto.arima(train_data, trace = TRUE, ic = "aicc") #
based on the AICc criterion.
arima_forecast <- forecast(arima_model, h = test_length)

# 3. Fit Holt-Winters model
holt_winters_model <- HoltWinters(train_data, seasonal =
"additive")
holt_winters_forecast <- forecast(holt_winters_model, h =
test_length)
```

```
# !---------------------------------------------------!
# --- 8. Model Evaluation, RMSE, AICc, ACF Residual ---
# !---------------------------------------------------!
# Print Values
sea_naive_accuracy <- accuracy(sea_naive_forecast, test_data)
print(sea_naive_accuracy)

print("ARIMA(1,1,1)(0,1,0)[12]:")
#summary(manualArima_forecast)
manArima_accuracy <- accuracy(manualArima_forecast, test_data)
print(manArima_accuracy)

print("ARIMA(1,1,0)(0,1,1)[12]:")
#summary(manualArimaW_forecast)
manArimaW_accuracy <- accuracy(manualArimaW_forecast, test_data)
print(manArimaW_accuracy)

arima_accuracy <- accuracy(arima_forecast, test_data)
print(arima_accuracy)

holtsWinter_accuracy <- accuracy(holt_winters_forecast, test_data)
print(holtsWinter_accuracy)

# Seasonal Naive Plot
autoplot(sea_naive_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("Seasonal Naive Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Manual ARIMA Plot
autoplot(manualArima_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(1,1,1)(0,1,0)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Manual ARIMA W Plot
autoplot(manualArimaW_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(1,1,0)(0,1,1)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# ARIMA Plot
autoplot(arima_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")

# Holt's Winter Plot
autoplot(holt_winters_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("Holt's Winter Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand")







# !----------------------------------------------------!
# ---------- 9. Best model summary-- -----------------
# !----------------------------------------------------!
# !!! 1. Best Model: ARIMA(1,1,0)(0,1,1)[12]
summary(manArimaW_model)
checkresiduals(manArimaW_model, plot = TRUE)
print(manArimaW_accuracy)
# Plot
autoplot(manualArimaW_forecast) +
  autolayer(test_data, series = "Actual Data", color = "red") +
  ggtitle("ARIMA(1,1,0)(0,1,1)[12] Forecast vs Actual Data") +
  xlab("Time") +
  ylab("Demand") +
  guides(colour = guide_legend(title = "Legend"))
```