

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

School of Computer Science and Engineering

Nanyang Technological University

AY2023/24

AI6123 Time Series Analysis

Group Project

Group Members:

Qiu Junping	G2304210E
Wong Seik Man	G2303494L
Kee Ming Yuan	G2304842E
Yip Chen Fei	G2304486K
Lin Rui Eddie	G2304213G
Yu Yongshan	G2302824H

1. Data Visualization

The objective of this assignment is to fit an appropriate Autoregressive Integrated Moving Average (ARIMA) model — or Seasonal ARIMA (SARIMA) — to the monthly sales data of anti-diabetic drugs in Australia, spanning from 1992 to 2008. These figures represent the total monthly prescriptions of pharmaceutical products classified under the Anatomical Therapeutic Chemical (ATC) Classification System code A10, as recorded by the Australian Health Insurance Commission.

The raw time series data, provided in “drug.txt,” was imported and plotted to visually identify patterns and trends. The resulting visualization is shown in **Figure 1** below (refer to the Python code of the plot in the Appendix).

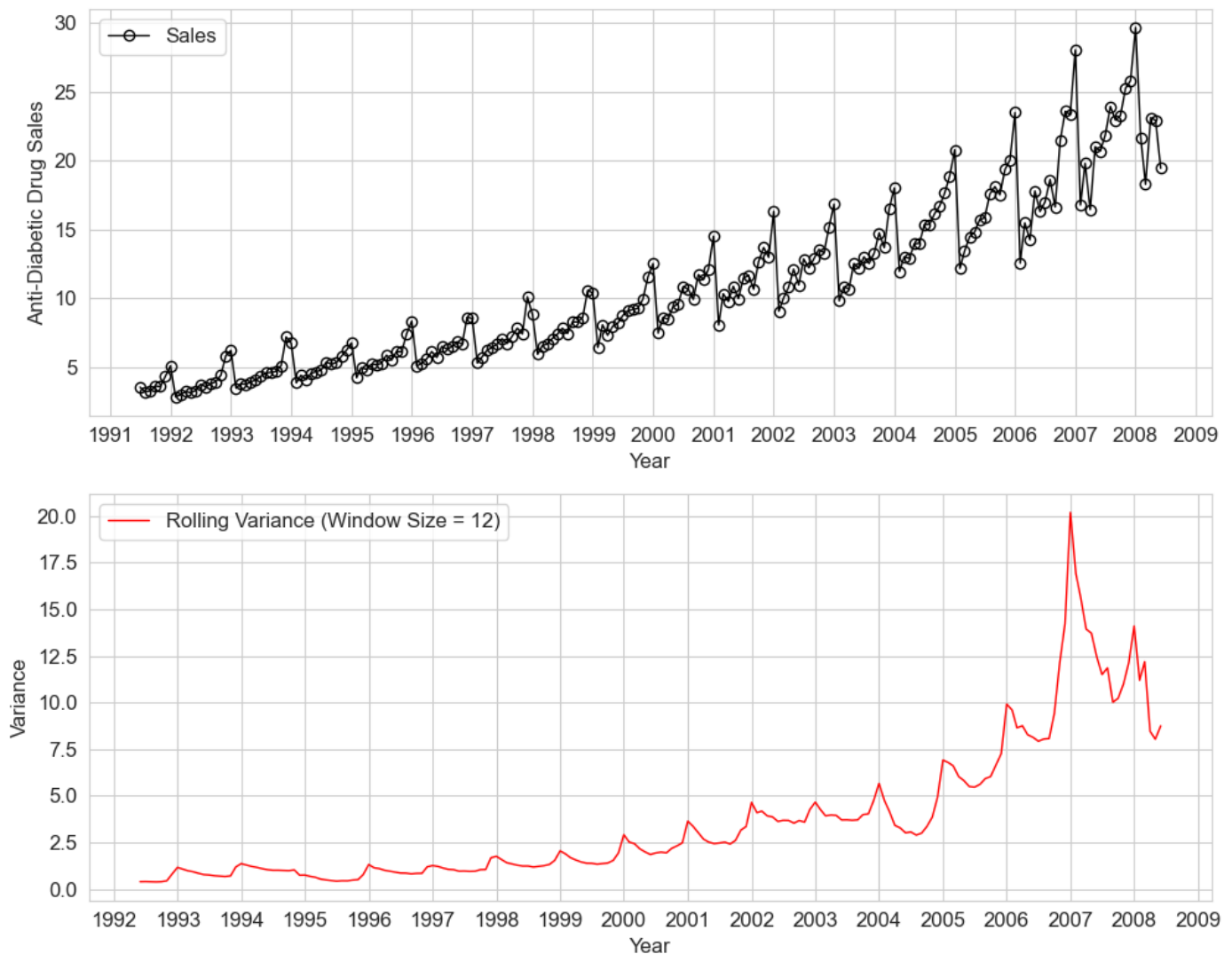


Figure 1: Plot of monthly anti-diabetic drug sales data (black) from 1992 to 2008, juxtaposed against the corresponding 12-month rolling-variance (red).

In **Figure 1**, we observe a (1) *seasonal* pattern of increasing sales throughout the year, peaking around December every year, which suggests a 12-month (annual) seasonal pattern in the data. There is also a clear (2) long-term *upward trend* in the drug's sales figures. Additionally, (3) the 12-month rolling *variance of the sales data increases over time*, suggesting that the data could benefit from a power transformation (e.g., Box-Cox transformation) to stabilize the variance before fitting an appropriate time series model.

2. Making the Time Series Stationary

The first step in time series analysis is to make the data stationary. This involves transforming the data and removing non-stationary components, such as, cyclicity, seasonality, and trends, to ensure it meets the conditions of weak stationarity: (1) an expected value that is independent of both time and period, (2) constant variance, and (3) constant autocovariance.

2.1. Stabilizing the Variance

Time-series data with variance that increases with time can be stabilized by a power transformation function like the Box-Cox transformation.

The Box-Cox transformation is defined by a power parameter, lambda (λ), which can be adjusted to optimally transform and normalize the data. The optimal λ can be empirically determined by finding the value of λ that maximizes the likelihood of the transformed data. Once applied, the Box-Cox transformation can stabilize the variance (making it more constant across the data range), correct for skewness, and help meet the homogeneity of variance condition of stationarity. The transformation is given by the formula:

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \ln(y), & \text{if } \lambda = 0 \end{cases} \quad (1)$$

The Box-Cox transformation is applied to the data using the Python 'scipy.stats' module. The optimal fitted lambda value was 0.061505584870954325; note that -0.008658942 was obtained from R. The transformed data are shown in **Figure 2** below (refer to the Python code of the plot in the Appendix).

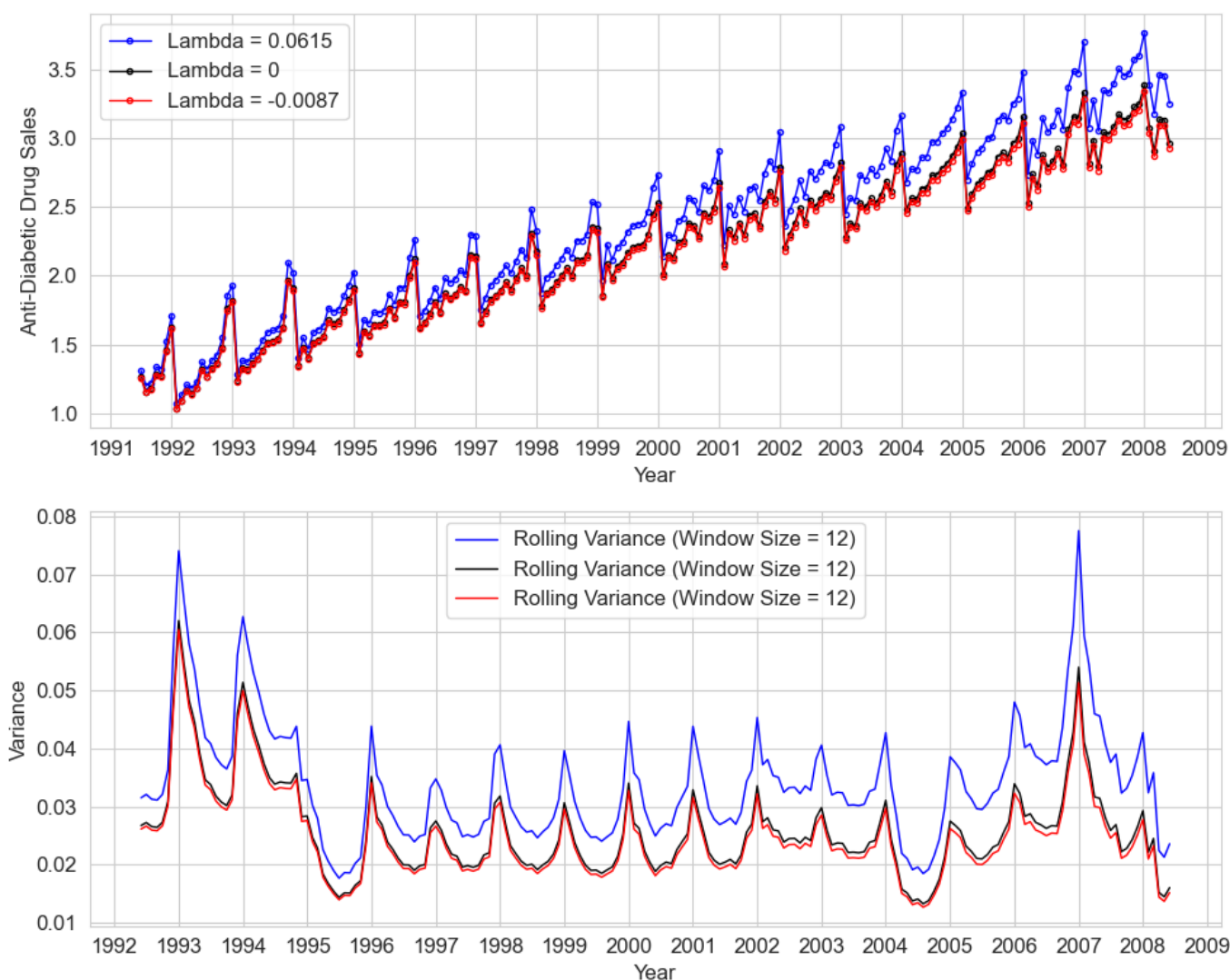


Figure 2: Plot of Box-Cox transformed data with $\lambda = 0.061505584870954325$ (blue), 0 (black) and -0.008658942 (red) respectively, juxtaposed against the corresponding 12-month rolling variance.

In **Figure 2**, we observe that for all 3 lambda values ($\lambda = 0$ was also shown for baseline comparison), Box-Cox transformation has successfully stabilized the variance throughout the time series.

To select the optimal lambda values, we also plotted the histogram of the Box-Cox transformed data in **Figure 3** below (refer to the Python code of the plot in the Appendix).

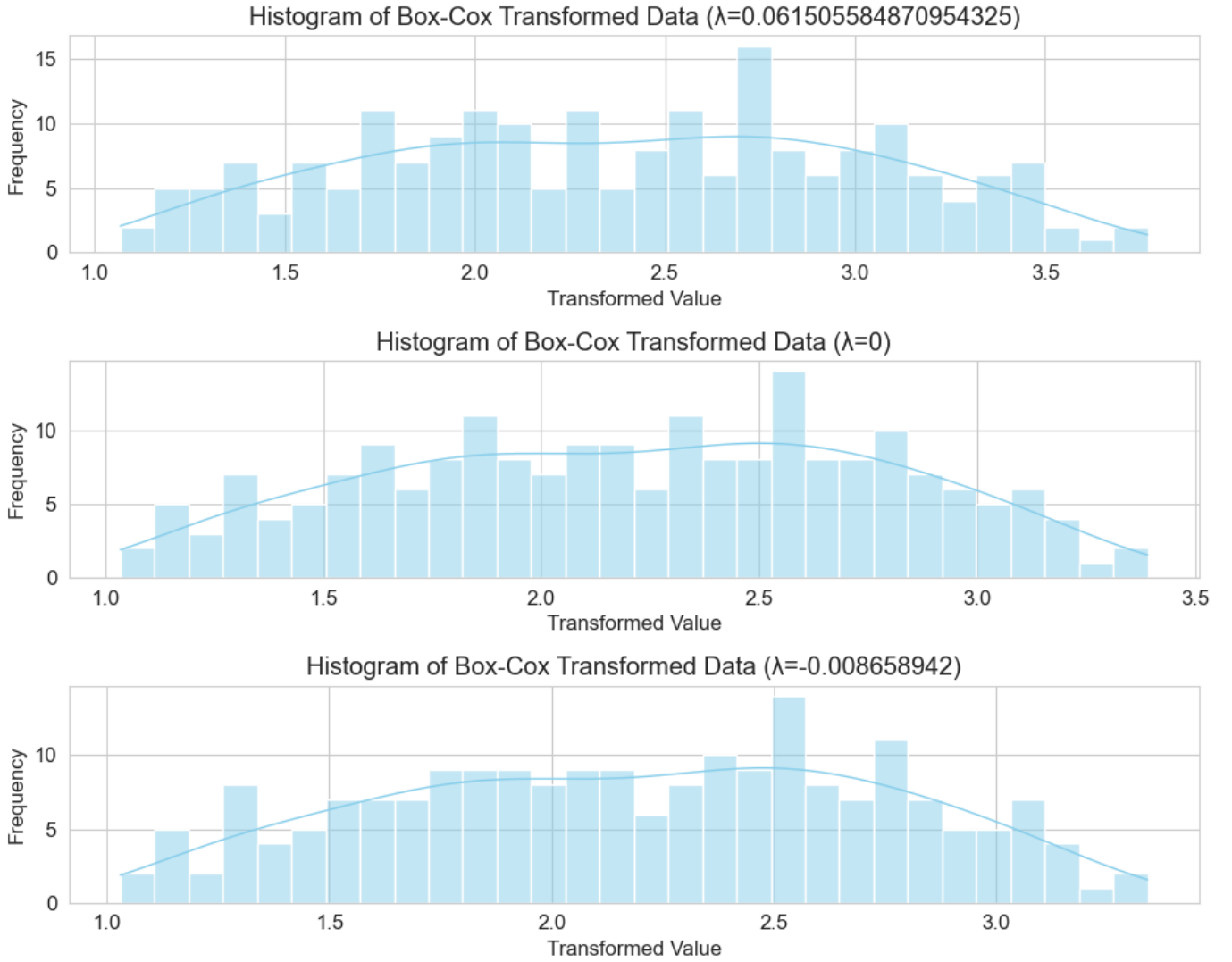


Figure 3: Histogram of Box-Cox transformed time series data.

By visual inspection, Box-Cox transformation with $\lambda = -0.008658942$ best fits a normal distribution, and was selected for Box-Cox transformation for subsequent time series analysis. Hence, the formula for the Box-Cox transformation is as follows:

$$y(\lambda) = \frac{y^{-0.008658942} - 1}{-0.008658942} \quad (2)$$

In conclusion, the Box-Cox transformation helped the time series data meet more conditions for stationarity by stabilizing variance across the dataset, making it better suited for further time series analysis. However, we also noted that the seasonal pattern and long-term upward trend persist, which is to be expected as the Box-Cox transformation does not remove the seasonal or trend component of the data.

2.2. Removing the Seasonal Component

The seasonal component of the time series is a pattern that repeats at regular intervals over time, typically within a fixed and known period, such as daily, monthly, quarterly, or annually.

Seasonal components can obscure long-term trends and cyclical behaviours, making it difficult to perform accurate forecasting. Additionally, seasonality violates the condition that the expected value of the time series is independent of both time and period. Therefore, removing seasonality helps meet this stationarity condition, thereby improving the validity of the model outputs and the reliability of predictions.

The 12-period (12-month) seasonality pattern previously observed in Section 1 is addressed by applying a differencing of lag 12 to the Box-Cox transformed data. The differenced data is shown in **Figure 4** below (refer to the Python code of the plot in the Appendix).

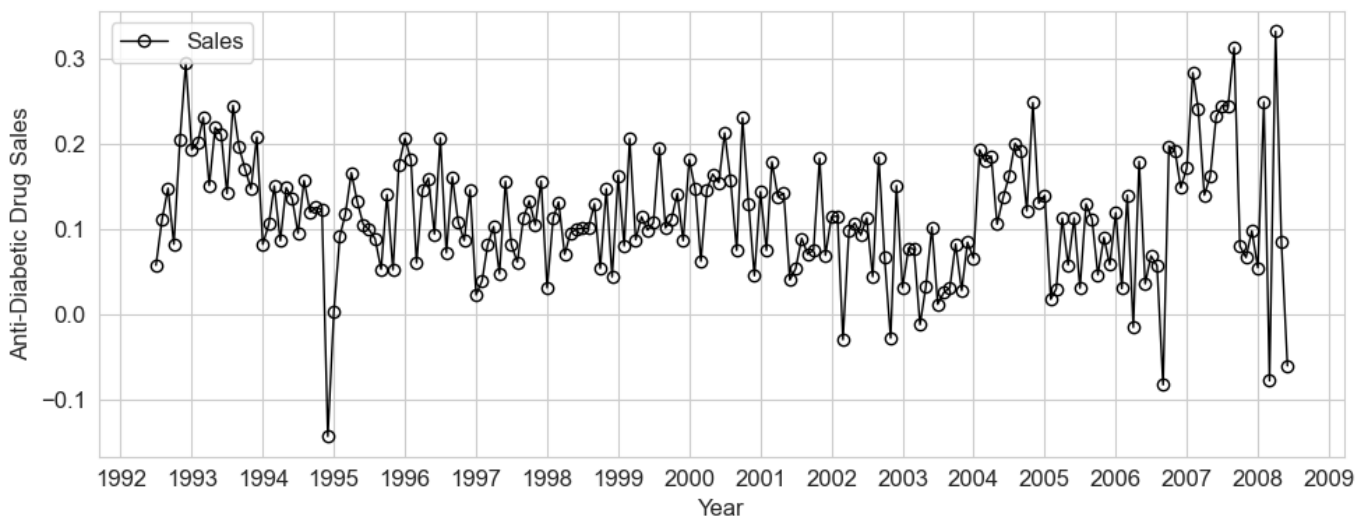


Figure 4: Plot of drug sales data after lag-12 differencing.

From **Figure 4**, it can be observed that the 12-month seasonal pattern is no longer apparent, suggesting that the seasonal component has been effectively removed.

Furthermore, there appears to be no obvious trend in the data with its seasonal component removed, which suggests that further trend differencing may not be required; in other words, the time series already appears to be stationary.

However, for completeness, we will proceed to remove the trend component in the next subsection and examine the results. We will then determine whether the removal of the trend component is necessary via diagnostic tests; such as examining the Autocorrelation Function (ACF), and conducting an Augmented Dickey-Fuller (ADF) test.

2.3. Removing the Trend Component

Trend in the time series data, characterized by a consistent increase or decrease over time, violates the condition of stationarity, which requires the expected value to be independent of time or period. Therefore, removing the trend component is crucial before fitting a time series model to achieve stationarity. This allows the model to more accurately capture the autocorrelation structure within the stationary residuals (refer to the Python code of the plot in the Appendix).

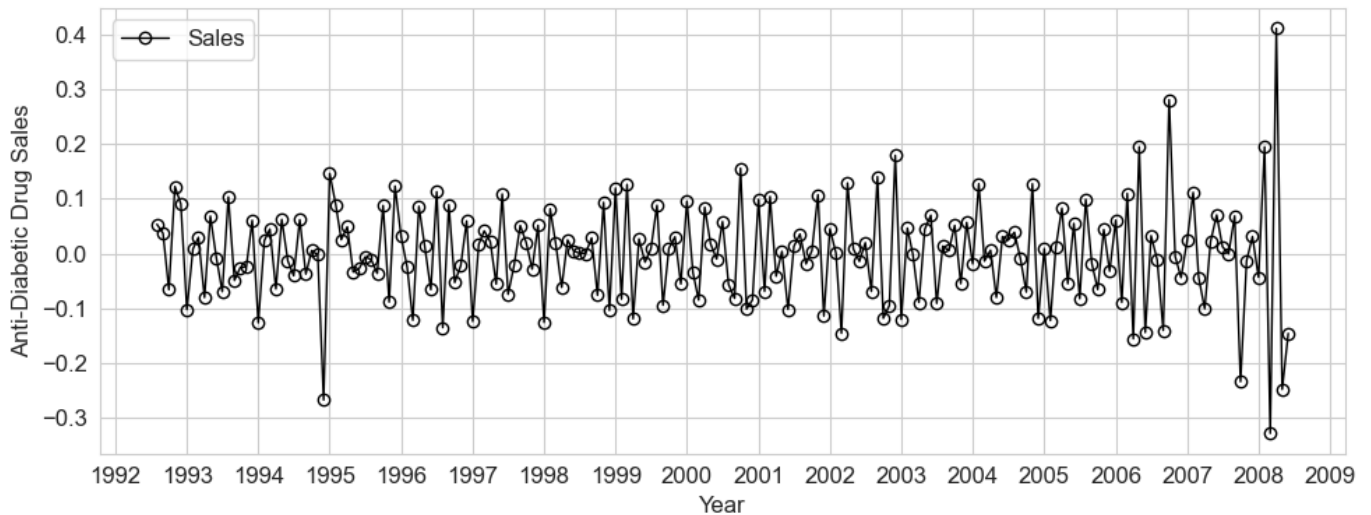


Figure 5: Plot of monthly anti-diabetic drug sales after Box-Cox transformation and lag 1 differencing.

After applying lag-1 differencing to the data with its seasonal component removed – a common method for removing the trend component from a time series – the resulting **Figure 5** plot suggests that the long-term trend has been largely eliminated. The data now fluctuates around a mean that appears constant over time, rather than exhibiting a systematic increase or decrease. This outcome indicates that the differencing process has successfully stabilized the expected value of the time series, a crucial step in rendering the data stationary.

2.4. Diagnostic Tests

2.4.1. Autocorrelation Function (ACF) Test

The ACF of the data with its seasonal and trend component removed is shown in **Figure 6** below (refer to the Python code of the plot in the Appendix).

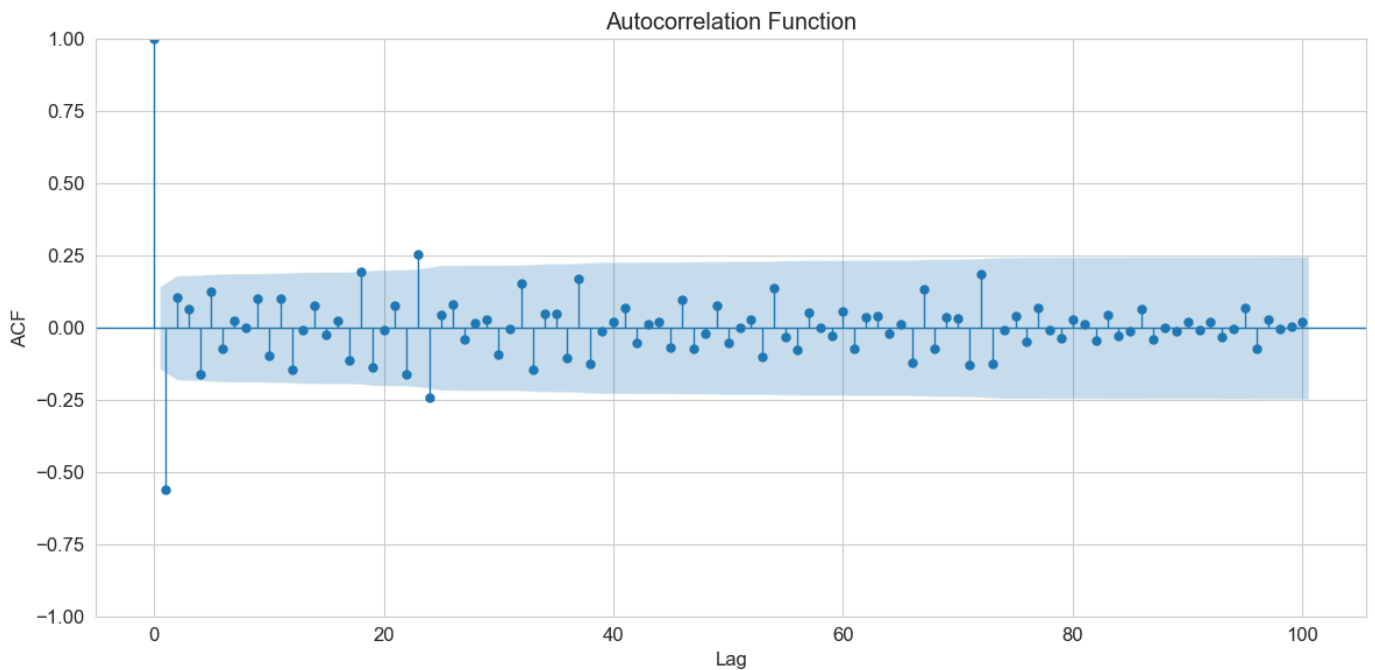


Figure 6: ACF of the drug sales data after Box-Cox transformation, lag-12 and subsequent lag-1 differencing.

From **Figure 6**, we observe that the lag-1 autocorrelation of a series is less than -0.5, suggesting a strong negative autocorrelation at lag 1. This is an indicator that the series may be over-differenced. This negative autocorrelation at lag 1 suggests that consecutive values are alternating in a predictable pattern, which is not characteristic of a purely random process and suggests that the differencing has gone too far, introducing artificial, systematic variation into the series.

Therefore, we reject applying further lag-1 differencing to the lag-12 data with its seasonal component removed.

2.4.2. Augmented Dickey-Fuller (ADF) Test

The ADF test is a statistical method used to determine the presence of a unit root in a time series dataset, which indicates non-stationarity. The default null hypothesis of the ADF test posits that the time series has a unit root, suggesting non-stationarity. Conversely, the alternative hypothesis contends that the data is stationary, lacking a unit root.

This test is crucial in time series analysis to distinguish between stationary and non-stationary behaviours, facilitating appropriate model selection and forecasting accuracy. The ADF test was implemented in R and results are shown in **Figure 7** below.


```
adf.test(drug_transformed_seasonal_diff)
```

Augmented Dickey-Fuller Test

```
data: drug_transformed_seasonal_diff  
Dickey-Fuller = -3.8071, Lag order = 5, p-value = 0.02  
alternative hypothesis: stationary
```

Figure 7: Results of ADF test on the sales data.

From **Figure 7**, the Augmented Dickey-Fuller (ADF) test p-value is 0.02. The test statistic is significantly below typical critical values for stationarity, and the p-value is below a common 95% significance level (like 0.05), leading us to reject the null hypothesis of a unit root (non-stationarity).

Therefore, we conclude that the Box-Cox transformed, lag-12 differenced time series ('drug_transformed_seasonal_diff') is stationary. This means that the time series does not have a unit root and has expected values that are independent of time and period, constant variance, and constant autocorrelation. This data preprocessing steps will be used in subsequent sections of the report.

3. Time Series Modelling

The pre-processed time series can be modelled by a SARIMA(p, d, q, P, D, Q, S) model. Where p, d , and q represent the **non-seasonal** autoregressive (AR) component, differencing, and moving average (MA) components respectively; P, D and Q represent the **seasonal** AR, differencing, and MA components respectively; and S indicates the period of the time series.

These parameters can be determined by analysing their ACF and PACF plots.

3.1. ACF Plot

ACF of the pre-processed data is plotted in R.

```
acf(drug_transformed_seasonal_diff,lag.max=36)
```

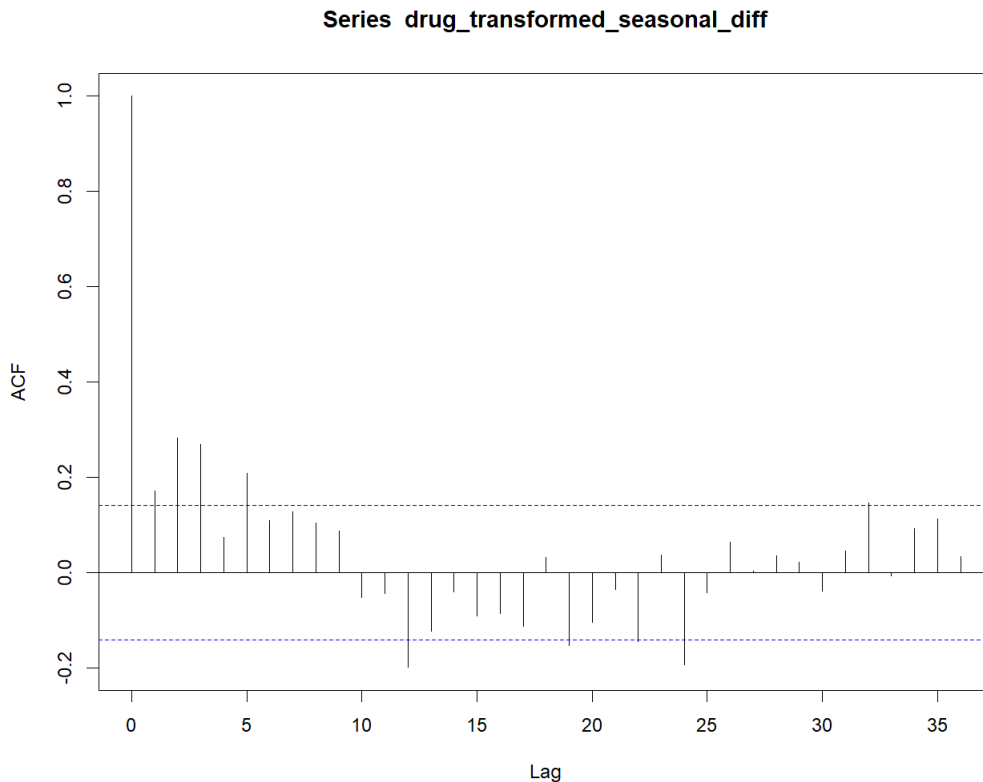


Figure 8: ACF of the drug sales data after Box-Cox transformation and lag-12 differencing.

From **Figure 8**, we observed that the ACF cutoffs after lag 5, which suggests that the moving average (MA) component (denoted by q) of the non-seasonal part of the SARIMA model, could be 5. Additionally, a significant peak at lag 24 indicates a seasonal pattern, suggesting that the seasonal MA component (denoted by Q) might be 2.

3.2. PACF Plot

Partial Autocorrelation Function (PACF) of the pre-processed data is plotted in R.

```
pacf(drug_transformed_seasonal_diff,lag.max=36)
```

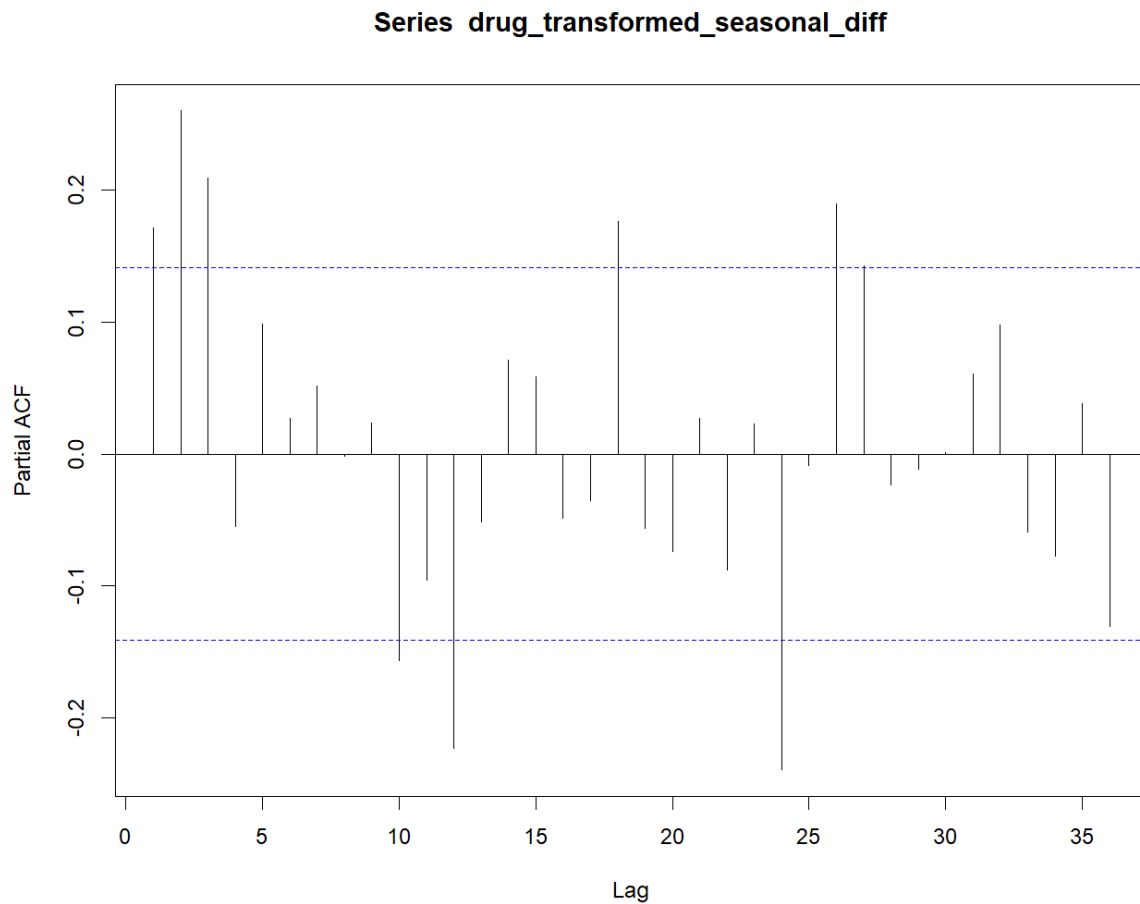


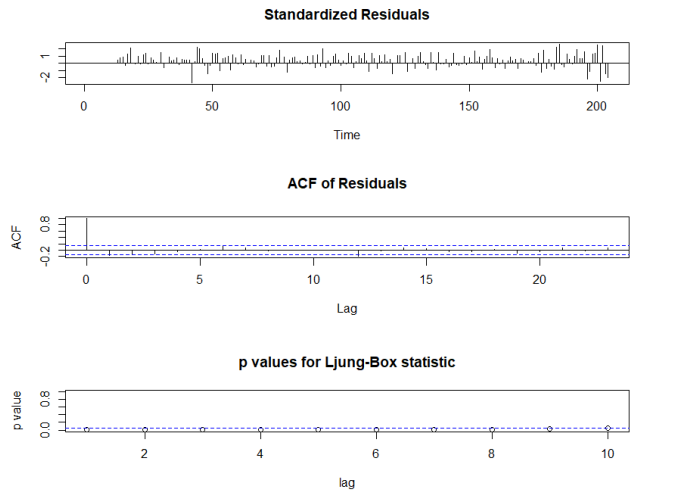
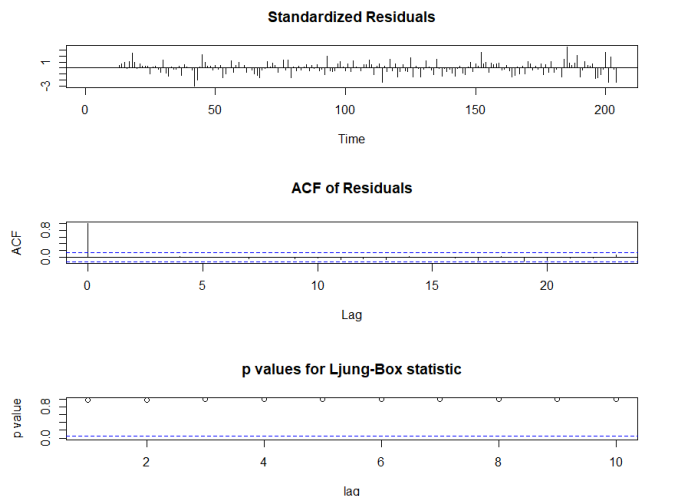
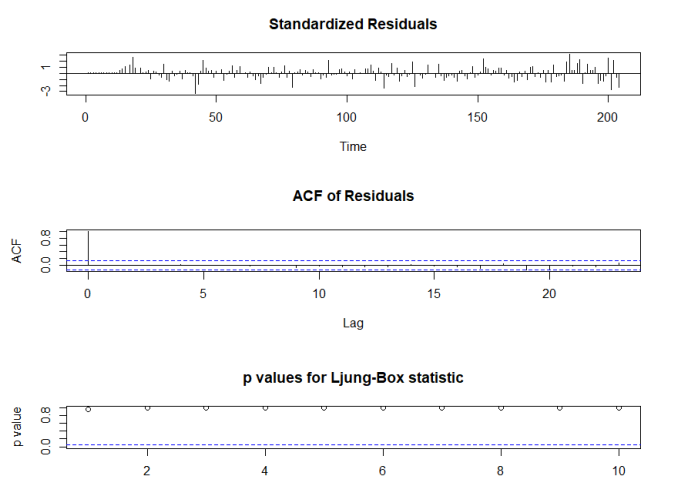
Figure 9: PACF of the drug sales data after Box-Cox transformation and lag-12 differencing.

From **Figure 9**, we observe that the PACF dies down after lag 10, suggesting that the order of the Autoregressive (AR) component of the non-seasonal part of the SARIMA model (denoted by p) could be 10. Additionally, a significant peak at lag 24 indicates a seasonal pattern, suggesting that the seasonal AR component (denoted by P) might be set to 2.

3.3. SARIMA model

Based on our observations of the ACF and PACF in Section 2.1 and 2.2, the optimal SARIMA(p, d, q, P, D, Q, s) parameters were determined to be SARIMA(10, 0, 5, 2, 1, 2, 12). In this section, we will systematically determine which combination of parameters yields the best diagnostic testing results.

Table 1: SARIMA model R code and corresponding adequacy results.

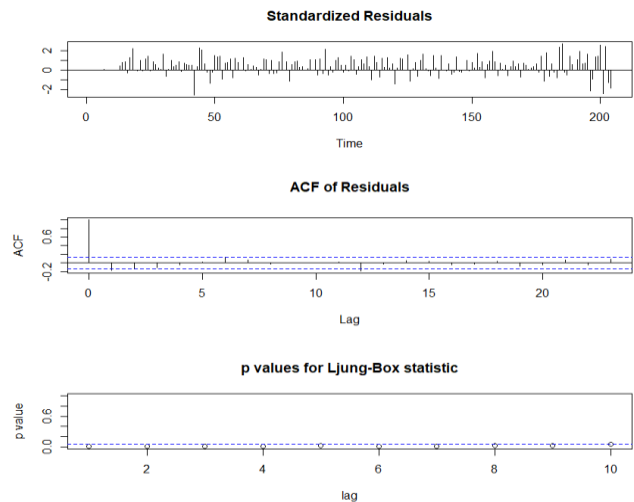
<p>Model 1:</p> <pre>### PACF ACF ### P=0 Q=2 (Seasonal @12) ### p=0 q=5 (non-Seasonal) fit_drug_1 <- arima(x = drug_transformed, order = c(0, 0, 5), seasonal = list(order = c(0, 1, 2), period = 12)) tsdiag(fit_drug_1) fit_drug_1</pre> <p>AIC= -420.45</p>	 <p>The diagnostic plots for Model 1 show standardized residuals over time, the ACF of residuals, and p-values for the Ljung-Box statistic. The residuals are centered around zero, and the ACF values are mostly within the confidence bounds, indicating a good fit.</p>
<p>Model 2:</p> <pre>### PACF ACF ### P=2 Q=0 (Seasonal @12) ### p=10 q=0 (non-Seasonal) fit_drug_2 <- arima(x = drug_transformed, order = c(10, 0, 0), seasonal = list(order = c(2, 1, 0), period = 12)) tsdiag(fit_drug_2) fit_drug_2</pre> <p>AIC= -515.12</p>	 <p>The diagnostic plots for Model 2 show standardized residuals over time, the ACF of residuals, and p-values for the Ljung-Box statistic. The residuals are centered around zero, and the ACF values are mostly within the confidence bounds, indicating a good fit.</p>
<p>Model 3:</p> <pre>### PACF ACF ### P=0 Q=2 (Seasonal @12) ### p=10 q=0 (non-Seasonal) fit_drug_3 <- arima(x = drug_transformed, order = c(10, 0, 0), seasonal = list(order = c(0, 1, 2), period = 12)) tsdiag(fit_drug_3) fit_drug_3</pre> <p>AIC = -519.68</p>	 <p>The diagnostic plots for Model 3 show standardized residuals over time, the ACF of residuals, and p-values for the Ljung-Box statistic. The residuals are centered around zero, and the ACF values are mostly within the confidence bounds, indicating a good fit.</p>

Model 4:

```

### PACF ACF
### P=2 Q=0 (Seasonal @12)
### p=0 q=5 (non-Seasonal)
fit_drug_4<- arima(x = drug_transformed, order = c(0, 0,
5), seasonal = list(order = c(2, 1, 0), period = 12))
tsdiag(fit_drug_4)
fit_drug_4
AIC = -420.95

```

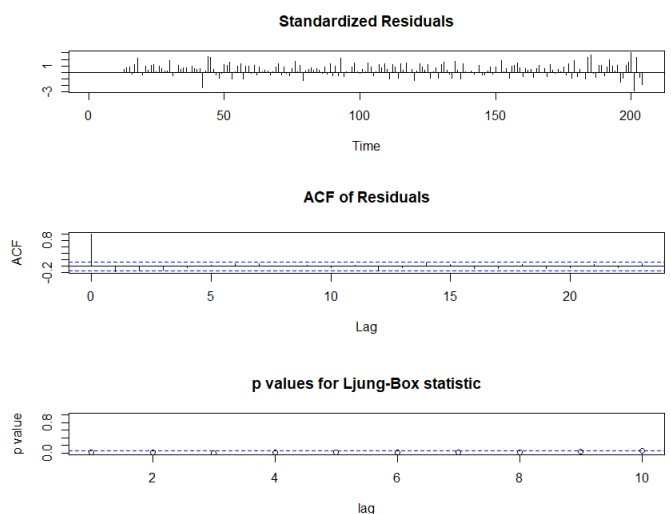
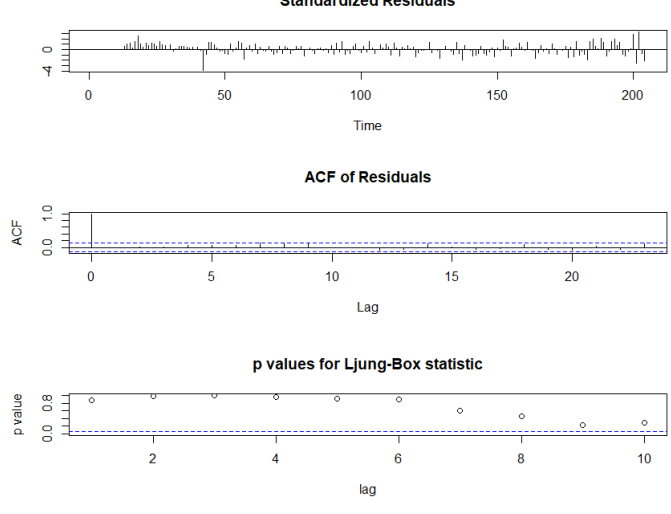
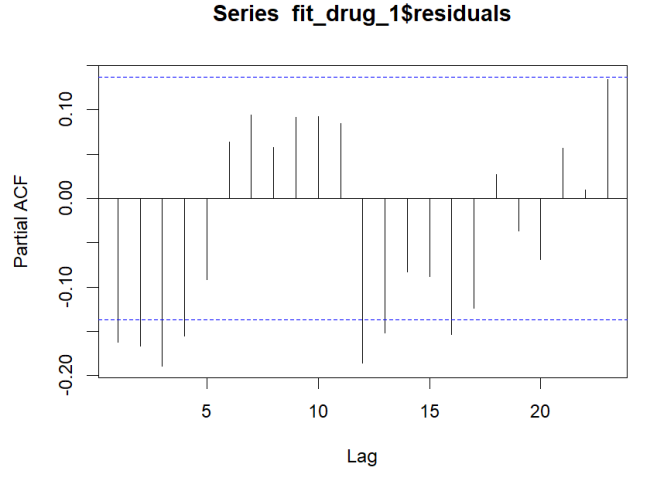
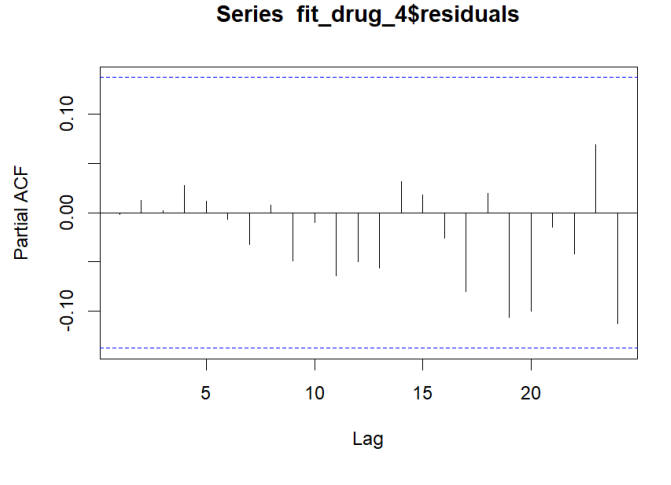


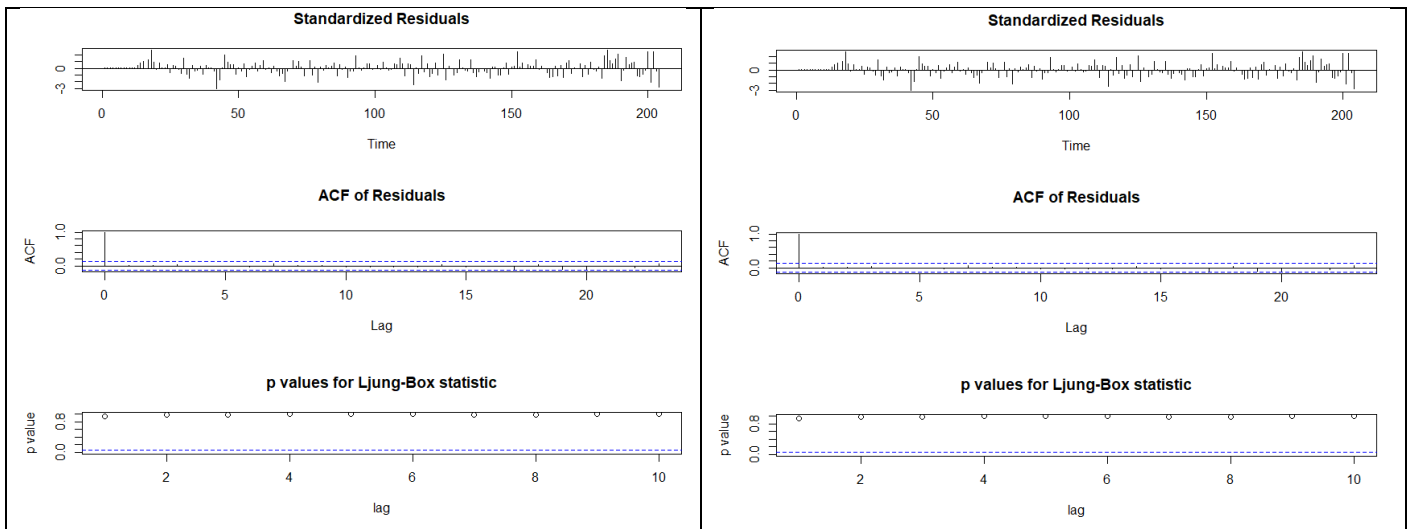
Based on Table 1, only models 2 and 3 are adequate, the rest of the models are inadequate. As seen in Table 1, for all 4 models, the standardized residual is scattered without any pattern or trend as seen. Nevertheless, for the 2 models which are not adequate, some of the ACF of residuals exceeded the threshold boundaries as outlined in blue and the p values for Ljung-Box statistics are not big, they fall within the blue dotted boundary which means the p values are negligible. Hence, model 1 and 4 can be improved.

3.4. Improved SARIMA models

Table 2: Improved SARIMA model R code and corresponding adequacy results.

Improved Model 1a	Improved Model 4a
<pre>acf(fit_drug_1\$residuals)</pre> <p>Series fit_drug_1\$residuals</p>	<pre>acf(fit_drug_4\$residuals)</pre> <p>Series fit_drug_8\$residuals</p>

<p>P=0, Q=3, p=0, q=6 fit_drug_1newa <- arima(x = drug_transformed, order = c(0, 0, 6), seasonal = list(order = c(0, 1, 3), period = 12)) tsdiag(fit_drug_1newa) AIC= -420.67</p>	<p>P=2, Q=1, p=0, q=6 fit_drug_2newa<- arima(x = drug_transformed, order = c(0, 0, 6), seasonal = list(order = c(2, 1, 1), period = 12)) tsdiag(fit_drug_2newa) AIC= -444.01</p>
	
<p>Improved Model 1</p>	<p>Improved Model 4</p>
<p>pacf(fit_drug_1\$residuals)</p> 	<p>pacf(fit_drug_4\$residuals)</p> 
<p>P=1, Q=2, p=4, q=5 fit_drug_1new<- arima(x = drug_transformed, order = c(4, 0, 5), seasonal = list(order = c(1, 1, 2), period = 12)) tsdiag(fit_drug_1new) fit_drug_1new AIC= -522.02</p>	<p>P=4, Q=0, p=4, q=5 fit_drug_2new <- arima(x = drug_transformed, order = c(4, 0, 5), seasonal = list(order = c(4, 1, 0), period = 12)) tsdiag(fit_drug_2new) fit_drug_2new AIC = -520.46</p>

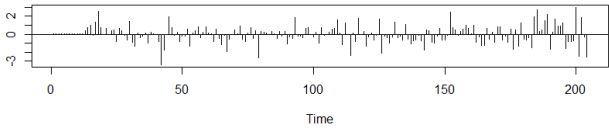
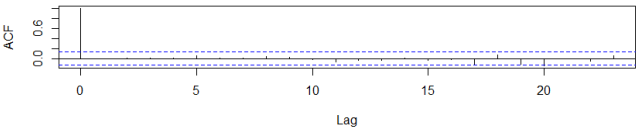
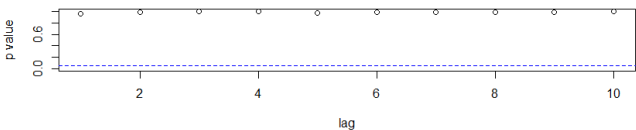
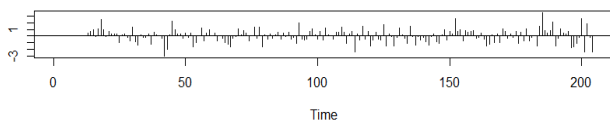
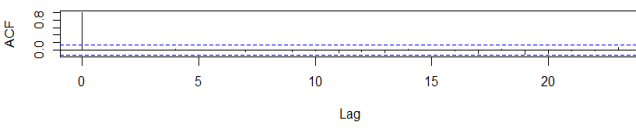
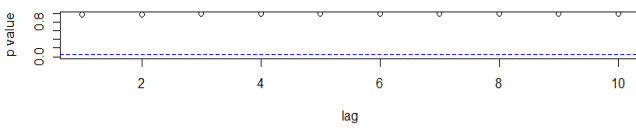
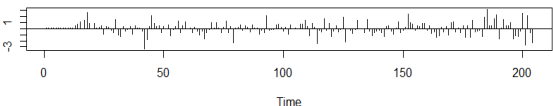
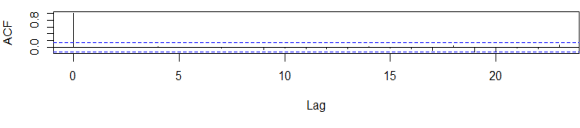
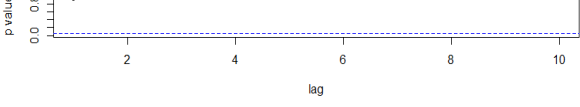
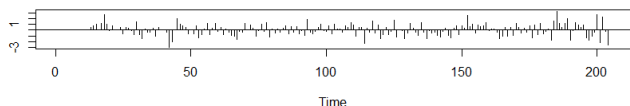
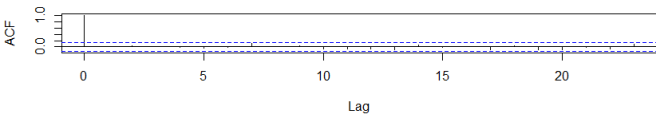
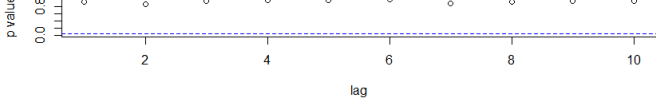
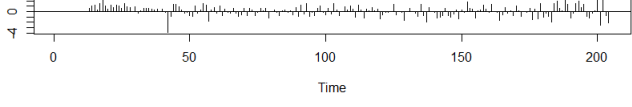
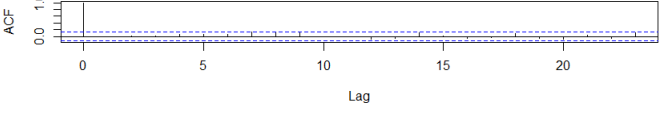
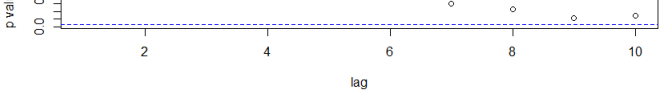


After improvement, we see that the standardized residual for Improved Model 1, Improved Model 4a and Improved Model 4 models are scattered without any pattern or trend. The ACF of residuals are within the threshold boundaries outlined in blue and the p values for Ljung-Box statistics are big and exceeds the blue dotted boundary which means the p values are not negligible. Those 3 models are adequate.

Nevertheless, we see that the p values for Ljung-Box statistics are small and does not exceed the blue dotted boundary for Improved Model 1a which means the p values are not negligible. Improved Model 1a is **not** adequate.

3.5. Diagnostic of the model

Table 3: Collated diagnostic plot of the top 5 models based on their AICs

Improved Model 1	Model 2
<p>Standardized Residuals</p>  <p>ACF of Residuals</p>  <p>p values for Ljung-Box statistic</p> 	<p>Standardized Residuals</p>  <p>ACF of Residuals</p>  <p>p values for Ljung-Box statistic</p> 
Model 3	Improved Model 4
<p>Standardized Residuals</p>  <p>ACF of Residuals</p>  <p>p values for Ljung-Box statistic</p> 	<p>Standardized Residuals</p>  <p>ACF of Residuals</p>  <p>p values for Ljung-Box statistic</p> 
Improved Model 4a	
<p>Standardized Residuals</p>  <p>ACF of Residuals</p>  <p>p values for Ljung-Box statistic</p> 	

Ideally, the standardized residuals (top subplot) over time should look like white noise; i.e., they are normally distributed with mean zero and constant variance. The residuals of all 5 models appear to be randomly scattered around the zero line without any systematic pattern, which is a good sign that the model is capturing the data's structure well.

The middle subplot is the ACF of the residuals. We look for any significant autocorrelation at different lags because significant autocorrelation could suggest that the model has not fully captured the data's dynamics. In all 5 diagnostic plots, all autocorrelations seem to be within the confidence bounds (blue dotted lines), indicating that there are no significant autocorrelations and that the residuals are essentially random.

The bottom subplot shows the p-values of the Ljung-Box test for various lags. This test checks the null hypothesis that the residuals are independently distributed, meaning they exhibit no autocorrelation. A p-value below a certain significance level (commonly 0.05) would reject this null hypothesis, indicating autocorrelation in the residuals. All 5 diagnostic plots suggests that all p-values are well above the common significance levels, so there is no evidence to reject the null hypothesis; thus, the residuals can be considered independent.

Overall, all 5 diagnostic plots suggest that all 5 model fits the data well and all 5 models are adequate. The residuals appear to be random and independently distributed, which implies that the model has adequately captured the time series' patterns, and the residuals display no signs of autocorrelation.

3.6. Consolidated Model AICs

Table 4: AIC values from the 8 models

Model Name	SARIMA value				Model Adequacy	AIC
	P	Q	p	q		
Model 1	0	2	0	5	Not Adequate	-420.45
Model 2	2	0	10	0	Adequate	-515.12
Model 3	0	2	10	0	Adequate	-519.68
Model 4	2	0	0	5	Not Adequate	-420.95
Improved Model 1	1	2	4	5	Adequate	-522.02 (Best SARIMA model)
Improved Model 4	4	0	4	5	Adequate	-520.46
Improved Model 1a	0	3	0	6	Not Adequate	-420.67
Improved Model 4a	2	1	0	6	Adequate	-444.01

Looking at the table above, we consider the **Improved Model 1** to be the best model as it has the lowest AIC. The equation of the Improved Model 1 is as follows:

$$Y_t = \nabla_s^{12} X_t \text{ (12 months seasonal pattern, no time differencing)} \quad (3)$$

$$\varphi_4(B)\phi_1(B^s)Y_t = \vartheta_5\theta_2Z_t$$

3.7. Model fit Visualization

```

drug_transformed_fitted_value1 <- drug_transformed - fit_drug_1new$residuals
drug_fitted_value1 <- InvBoxCox(drug_transformed_fitted_value1, lambda_drug)
drug_transformed_fitted_value2 <- drug_transformed - fit_drug_2$residuals
drug_fitted_value2 <- InvBoxCox(drug_transformed_fitted_value2, lambda_drug)
drug_transformed_fitted_value3 <- drug_transformed - fit_drug_3$residuals
drug_fitted_value3 <- InvBoxCox(drug_transformed_fitted_value3, lambda_drug)
drug_transformed_fitted_value4 <- drug_transformed - fit_drug_2new$residuals
drug_fitted_value4 <- InvBoxCox(drug_transformed_fitted_value4, lambda_drug)
drug_transformed_fitted_value5 <- drug_transformed - fit_drug_2newa$residuals
drug_fitted_value5 <- InvBoxCox(drug_transformed_fitted_value5, lambda_drug)

plot(drug, type = 'o', xlim = c(0, 204), ylim = c(0, 31), pch=16, cex = 0.5, xlab = "Observation", ylab = " Anti-Diabetic Dr
ug Sales", main = "Comparing all Adequate Models with Original Data")
lines(drug_fitted_value1,type = 'l', lty=2, col='red')
lines(drug_fitted_value2,type = 'l', lty=2, col='blue')
lines(drug_fitted_value3,type = 'l', lty=2, col='green')
lines(drug_fitted_value4,type = 'l', lty=2, col='pink')
lines(drug_fitted_value5,type = 'l', lty=2, col='orange')
legend("topleft", legend = c("Original", "Improved Model 1","Model 2","Model 3","Improved Model 4", "Improved Model
4a"), fill = c("black","red","blue","green","pink","orange"),cex = 0.7)

plot(drug, type = 'o', xlim = c(0, 204), ylim = c(0, 31), pch=16, cex = 0.5, xlab = "Observation", ylab = " Anti-Diabetic Dr
ug Sales", main = "Comparing top 2 Adequate Models with Original Data")
lines(drug_fitted_value1,type = 'l', lty=2, col='red')
lines(drug_fitted_value4,type = 'l', lty=2, col='blue')
legend("topleft", legend = c("Original", "Improved Model 1", "Improved Model 4"), fill = c("black", "red", "blue"),cex = 0.7)

```

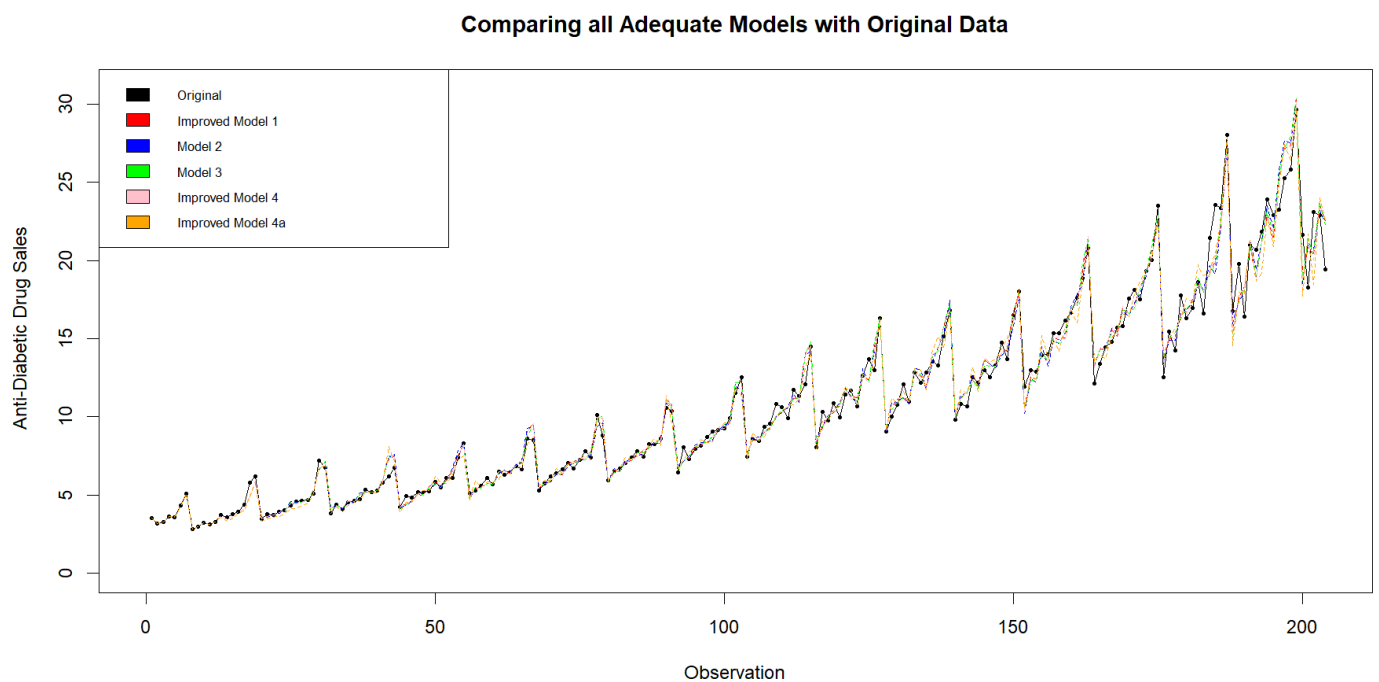


Figure 10: Comparing the fit of all adequate models with the raw data.

Inspecting **Figure 10** above, all 5 adequate models seem to fit the raw drug sales data (non-stationary data which is not Box-Cox transformed) well. Visually, Model 2, Improved Model 4a and Model 3 do not fit the raw data as good as the Improved Model 1 and Improved Model 4 as there are more deviations from the raw data. This is to be expected, as Model 2, Improved Model 4a and Model 3 have higher AIC than the Improved Model 1 and Improved Model 4.

Comparing top 2 Adequate Models with Original Data

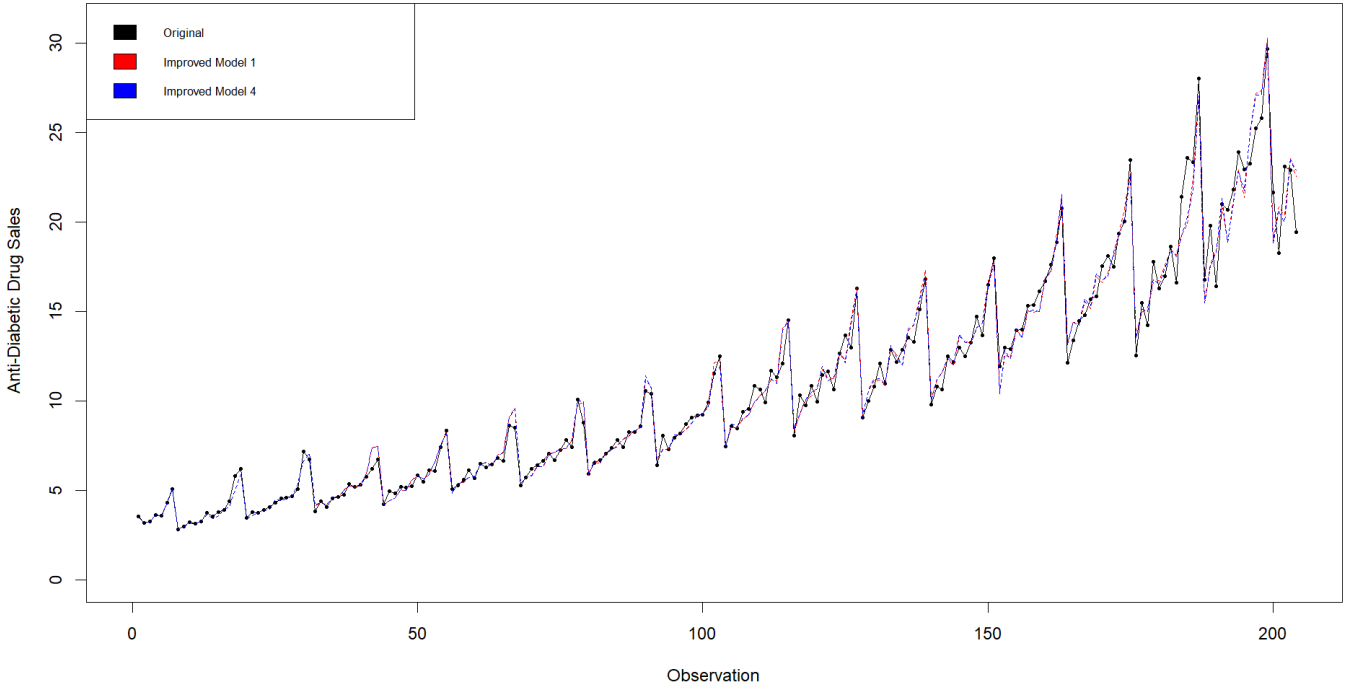


Figure 11: Comparing the fit of the best 2 adequate models with the raw data.

Inspecting **Figure 11** above, we do see that the plot of Improved Model 1 lies closer to the plot of raw data mode, so Improved Model 1 fits the raw data better than Improved Model 4. This is to be expected, as the AIC of Improved Model 1 is lower than that of Improved Model 4.

4. Spectral Analysis

As the data exhibits a seasonal pattern, spectral analysis was used to determine the frequencies within the time series data. The equation representing the cosine-sine pairs, corresponding to the frequency of each period, is shown in Equation 4 below.

$$y_t = \sum_{j=1}^k \alpha_j \cos(2\pi f_j t) + b_j \sin(2\pi f_j t) + Z_t \quad (4)$$

Given that spectral analysis is only applicable to stationary datasets, we use the pre-processed data set, 'drug_transformed_seasonal_diff'. This dataset is derived after applying a Box-Cox transformation and removing the seasonal component from the raw data, ensuring stationarity.

4.1. Periodogram

The periodogram of the pre-processed data is calculated and plotted in R.

```
periodogram(drug_transformed_seasonal_diff) ;abline(h=0)
```

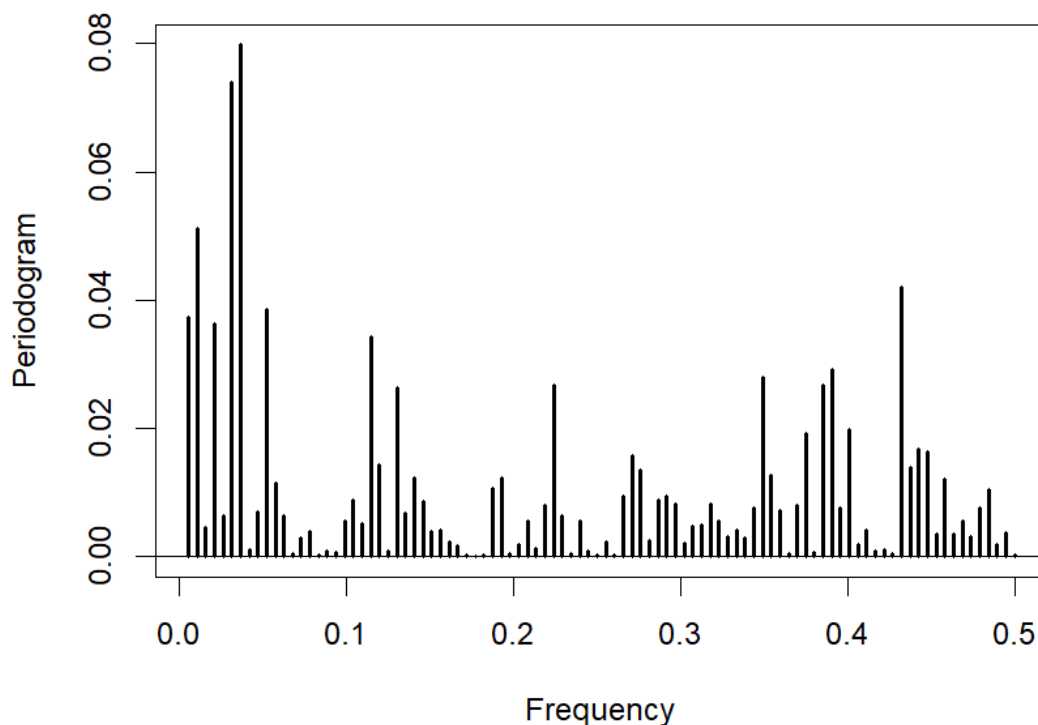


Figure 12: Periodogram of stationary drugs data which is after Box-Cox transformation and seasonal component removed.

Inspecting the periodogram above, we observe numerous spikes, each indicating a frequency that significantly contributes to the cyclical patterns within the data. Therefore, we select significant spectral density estimate values exceeding 0.04. The 0.04 threshold was chosen to identify the most substantial spectral densities.

```
per <- periodogram(drug_transformed_seasonal_diff)
significant_indices <- which(periodogram(drug_transformed_seasonal_diff)$spec > 0.04)
points(per$freq[significant_indices], per$spec[indices_to_highlight], col = "red", pch = 16)
```

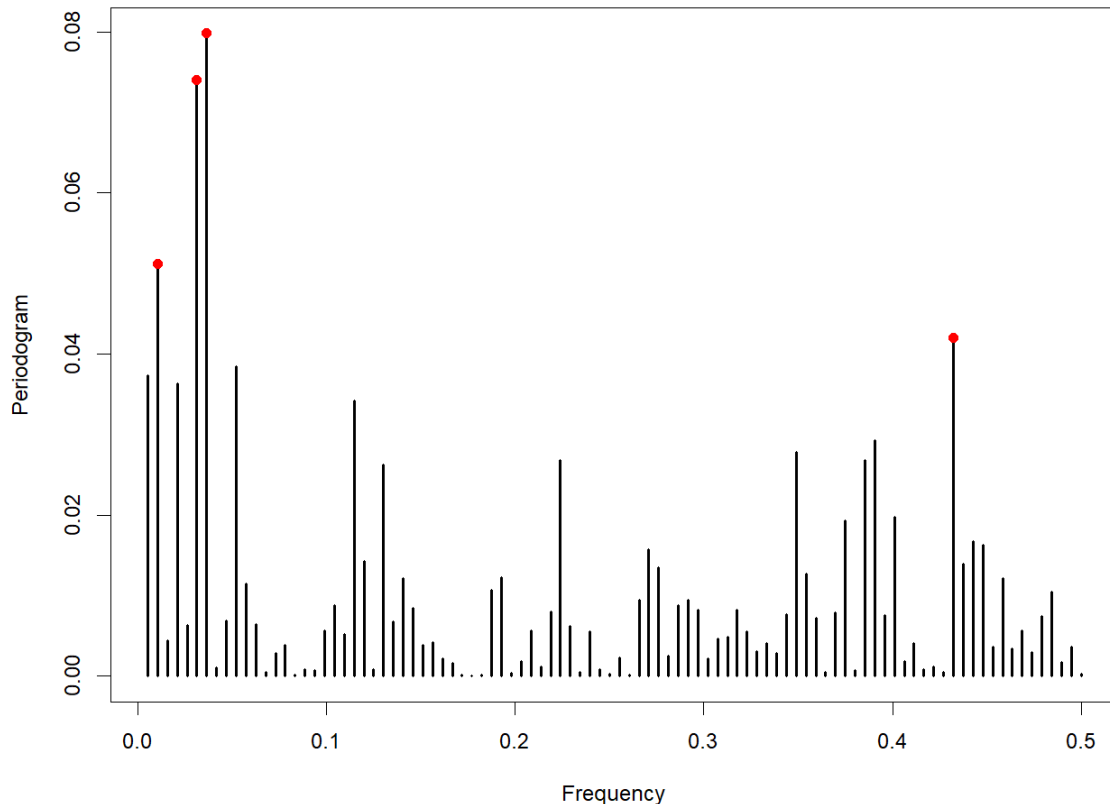


Figure 13: Periodogram with spectral values > 0.04 highlighted with red dots.

Figure 13 highlights 4 frequencies with spectral density estimates exceeding 0.04. The values of the frequencies were derived via R and shown below.

```
print(significant_indices)
[1] 2 6 7 83

significant_frequencies <- per$freq[significant_indices]
print(significant_frequencies)
[1] 0.01041667 0.03125000 0.03645833 0.43229167
```

The least-square approach is used to find the values of α_j and β_j .

```
significant_spec <- period$spec[significant_indices]
regressors <- data.frame(drug_transformed_seasonal_diff)
time_index <- seq_along(drug_transformed_seasonal_diff)
for(i in 1:length(significant_frequencies)) {
+   freq <- significant_frequencies[i]
+   regressors[[paste0('sin_', i)]] <- sin(2 * pi * freq * time_index)
+   regressors[[paste0('cos_', i)]] <- cos(2 * pi * freq * time_index)
+ }
spectral_model <- lm(drug_transformed_seasonal_diff ~ ., data = regressors)
summary(spectral_model)
```

Call:

```
lm(formula = y ~ cos_term1 + sin_term1 + cos_term2 + sin_term2 +
    cos_term3 + sin_term3 + cos_term4 + sin_term4, data = data_df)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.213600	-0.039317	0.000842	0.044308	0.234057

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.1163659	0.0046205	25.185	< 2e-16 ***

```
cos_term1  0.0228336  0.0065344  3.494 0.000596 ***
sin_term1  -0.0034246  0.0065344  -0.524 0.600856
cos_term2  -0.0277609  0.0065344  -4.248 3.42e-05 ***
sin_term2  -0.0007513  0.0065344  -0.115 0.908594
cos_term3  -0.0278169  0.0065344  -4.257 3.31e-05 ***
sin_term3   0.0076098  0.0065344   1.165 0.245707
cos_term4  -0.0041073  0.0065344  -0.629 0.530413
sin_term4   0.0205257  0.0065344   3.141 0.001963 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06402 on 183 degrees of freedom
Multiple R-squared:  0.2478,    Adjusted R-squared:  0.2149
F-statistic: 7.536 on 8 and 183 DF, p-value: 1.107e-08
```

`anova(fit)`

Analysis of Variance Table

Response: y

```
      Df Sum Sq Mean Sq F value    Pr(>F)
cos_term1  1 0.05005 0.050052 12.2107 0.000596 ***
sin_term1  1 0.00113 0.001126  0.2747 0.600855
cos_term2  1 0.07398 0.073984 18.0491 3.423e-05 ***
sin_term2  1 0.00005 0.000054  0.0132 0.908594
cos_term3  1 0.07428 0.074283 18.1221 3.306e-05 ***
sin_term3  1 0.00556 0.005559  1.3562 0.245707
cos_term4  1 0.00162 0.001620  0.3951 0.530413
sin_term4  1 0.04045 0.040445  9.8670 0.001963 **
Residuals 183 0.75012 0.004099
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.2. Model from Logistic Regression

The below is the equation for the model from Logistic Regression:

$$\begin{aligned}
 y_t = & 0.0228336 \cos(2(0.01041667)\pi t) - 0.0034246 \sin(2(0.01041667)\pi t) - 0.0277609 \cos(2(0.03125)\pi t) \\
 & - 0.0007513 \sin(2(0.03125)\pi t) - 0.0278169 \cos(2(0.03645833)\pi t) \\
 & + 0.0076098 \sin(2(0.03645833)\pi t) - 0.0041073 \cos(2(0.43229167)\pi t) \\
 & + 0.0205257 \sin(2(0.43229167)\pi t) + 0.1163659
 \end{aligned} \tag{5}$$

From the above results in RStudio, we see that the R-squared value is 0.2149. This indicates that 21.49% of the variance in the dependent variable 'drug_transformed_seasonal_diff' is explained by the independent variables in the model which includes 4 sine and 4 cosine terms. The null hypothesis posits that the independent variables have no effect on the dependent variable. Since the p-value of 1.107×10^{-08} is an extremely small value and is less than 0.05, we reject the null hypothesis at 95% significance level. Therefore, we conclude that at least one of the independent variables in the model has a statistically significant relationship with the dependent variable. The F-statistic of 7.536 indicates that the regression model is statistically significant.

However, the relatively low R-squared value suggests that the independent variables in the model collectively explain only a small proportion of the variance in the dependent variable. This means that the model may not capture all the relevant factors influencing the dependent variable, or that the relationship between the predictors and the dependent variable is weak. While the model may be statistically significant, its practical usefulness for explaining and predicting the Anti-Diabetic Drug Sales may be limited due to the low R-squared value. Further refinement or exploration of the model may be necessary to improve its predictive accuracy and explanatory power.

4.3. Model Fit visualization with Stationary Drugs Data from Spectral Analysis

```
plot(drug_transformed_seasonal_diff, xlab = "Observation number", ylab = "Transformed Stationary Drug Sales")
lines(drug_transformed_seasonal_diff, type="l", col = "black")
lines(drug_transformed_seasonal_diff - fit$residuals, type="l", col = "red")
legend("bottom", legend = c("Stationary Drug Sales", "Spectral Analysis Model"), fill = c("black", "red"))
```

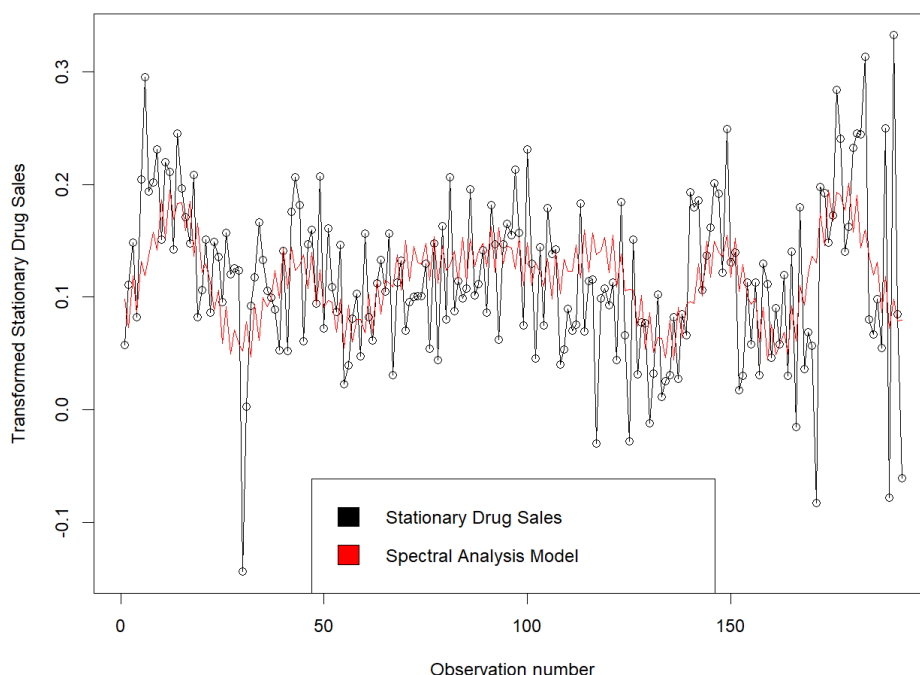


Figure 14: Comparison of Stationary Drug Sales data with Predicted data from Spectral Analysis Model

As seen in **Figure 14** above, the model from spectral analysis does not fit the stationary drug sales data well. It does not capture most of the extreme spikes or deep troughs in drug sales. This outcome is expected, as the R-squared value is low. However, to assess the fitness of the model more accurately, we proceeded to reverse the seasonal differencing and convert the fitted values back to the original scale.

```
# Create 12 new timestamps for predictions
new_time_index <- (max(time_index) + 1):(max(time_index) + 12)

# Create a data frame of 12 prediction with calculation of sine and cosine
new_regressors <- data.frame(matrix(ncol = 0, nrow = length(new_time_index)))
for(i in 1:length(significant_frequencies)) {
  freq <- significant_frequencies[i]
  new_regressors[, paste0('sin_', i)] <- sin(2 * pi * freq * new_time_index)
  new_regressors[, paste0('cos_', i)] <- cos(2 * pi * freq * new_time_index)
}

# Make predictions
predicted_values <- predict(spectral_model, new_regressors)
spectral_fitted <- c(spectral_model$fitted.values, predicted_values)

# Reconstitute values after reversing the lag-12 differencing. For the first 12 observations lost due to differencing,
# we use the original values from `drug_transformed`.
lag <- 12
base_values <- head(drug_transformed, lag)
reconstituted_values <- numeric(length(spectral_fitted) + lag)
reconstituted_values[1:12] <- base_values
for(i in 1:length(spectral_fitted)) {
  + reconstituted_values[i + lag] <- reconstituted_values[i] + spectral_fitted[i]
  + }

# Reverse the box-cox transformation and plot the result
spectral_fitted_original_scale <- InvBoxCox(reconstituted_values, lambda_drug)
spectral_prediction <- spectral_fitted_original_scale[205:216]
spectral_fitted_original <- spectral_fitted_original_scale[13:204]
```

```
plot(drug, type='o', pch = 16, xlim=c(0,220),ylim=c(0,40))
lines(c(rep(NA,12),spectral_fitted_original), col='red')
lines(c(rep(NA,204),spectral_prediction), type='o', pch = 18, col='green')
```

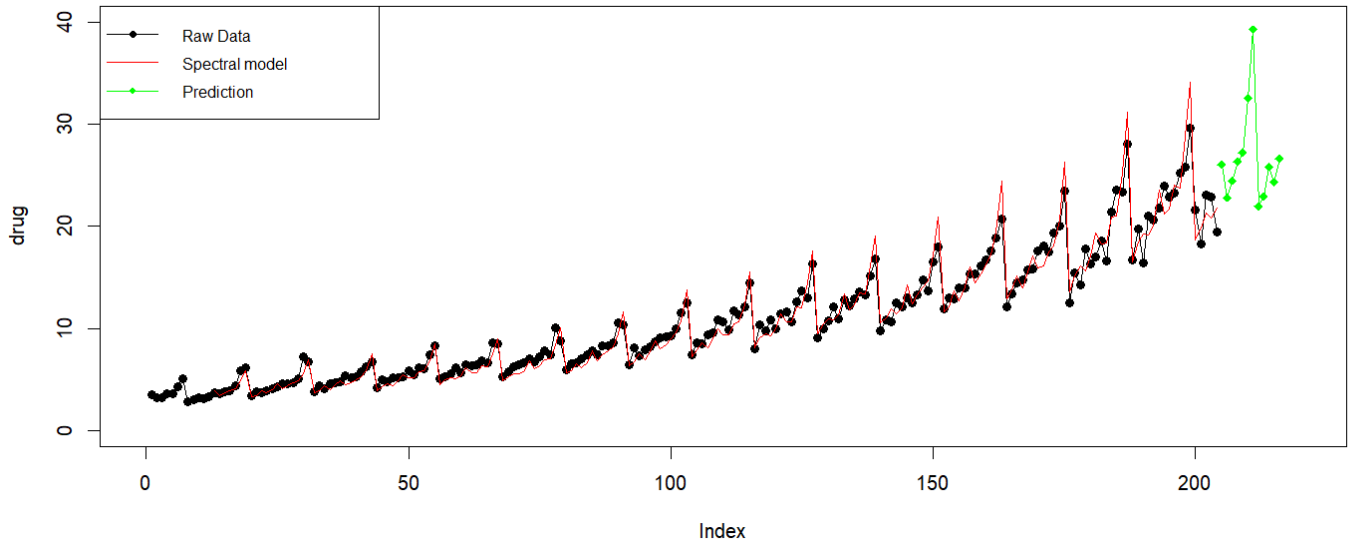


Figure 15: Fitted spectral model and its forecast.

The spectral model fit is overlaid in red, which traces the movements of the raw data quite closely, indicating a good fit. Notably, the model seems to capture the cyclic pattern and the general trend of the drug sales data. On the far right, the prediction is plotted in green, extending beyond the raw data. This represents the model's forecast for future values. It continues the pattern observed in the earlier data, suggesting a continuing upward trend, and maintaining the periodicity seen in the historical data.

```
loss_spectral <- drug[13:204] - spectral_fitted_original
rmse <- sqrt(mean(loss_spectral ^2))
mae <- mean(abs(loss_spectral))
mape <- mean(abs(loss_spectral /drug[13:204]))
cat("Model RMSE:", rmse, "MAE:", mae, "MAPE:", mape)

target <- drug[13:204]
SStot <- sum((target - mean(spectral_fitted_original))^2)
SSres <- sum((target - spectral_fitted_original)^2)
test_r_squared <- 1 - SSres / SStot
print(paste("Test R²:", test_r_squared))
```

Table 5: Performance metrics of the spectral model

Metric	Value
RMSE	1.124268
MAE	0.822363
MAPE	0.072114
Rsquared	0.962901

Table 5 presents the performance metrics of the spectral model, indicating a high degree of accuracy in its forecasts. The RMSE of 1.124268 and MAE of 0.822363 suggest that the model's predictions are relatively close to the actual values, with the errors being reasonably low. A MAPE of 0.072114 means that the model's

predictions deviate from the actual values by an average of 7.21%, which is typically considered good in many contexts. Furthermore, an R-squared value of 0.962901 reflects a very high proportion of variance in the dependent variable that is predictable from the independent variables, confirming the model's effectiveness in fitting the data. Overall, these metrics suggest that the spectral model performs very well in capturing the pattern and trends of the dataset.

5. Model Forecasting

5.1. Best SARIMA Models 3-Year Forecast

Since Improved Model 1 and Improved Model 4 are the top 2 best SARIMA models, we have plotted the fit of the model with the pre-processed data; R code shown below.

```
drug_transformed_fitted_value1new <- drug_transformed - fit_drug_1new$residuals
drug_transformed_fitted_value4new <- drug_transformed - fit_drug_2new$residuals
drug_fitted_value1new <- InvBoxCox(drug_transformed_fitted_value1new, lambda_drug)
drug_forecast1new <- InvBoxCox(predict(fit_drug_1new,n.ahead = 36)$pred, lambda_drug)
drug_fitted_value4new <- InvBoxCox(drug_transformed_fitted_value4new, lambda_drug)
drug_forecast4new <- InvBoxCox(predict(fit_drug_2new,n.ahead = 36)$pred, lambda_drug)
plot(drug, type="l", ylim = c(0, 45),lty=1, xlab = "Time Observation", ylab = "Anti-Diabetic Drug Sales",main = "Top 2 SARIMA 36 months Forecast",xlim=c(0,240))
lines(drug_fitted_value1new, col = "red", lty = 2)
lines(drug_fitted_value4new, col = "blue", lty = 2)
lines(drug_forecast1new, col = "red")
lines(drug_forecast4new, col = "blue")
legend("topleft", legend = c("Raw Data", "Improved Model 1", "Improved Model 4", "Improved Model 1 Forecast", "Improved Model 4 Forecast"), col = c("black", "red", "blue", "red", "blue"), lty = c(1, 2, 2, 1, 1),cex=0.6)
```

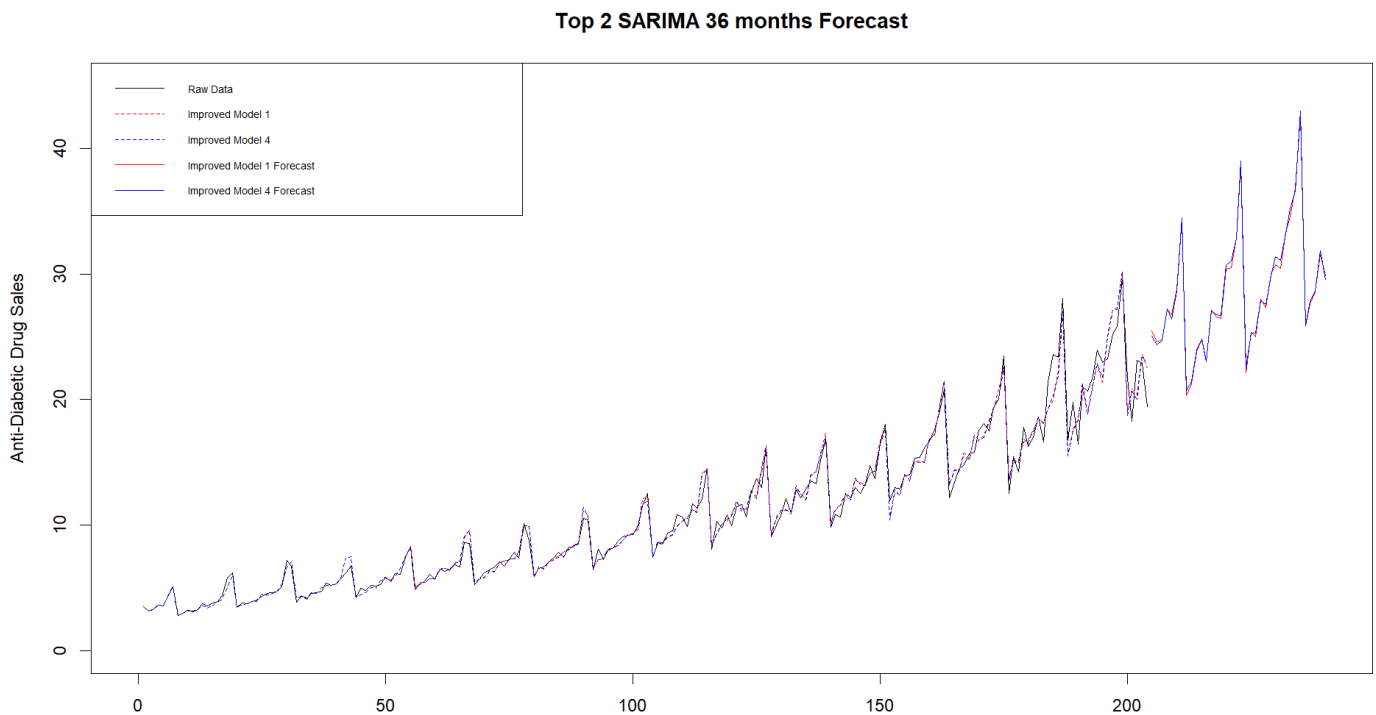


Figure 16: Comparison of 36-month forecast from Improved Model 1 and Improved Model 4.

The 36-month forecast results of Improved Model 1 and Improved Model 4 are very similar. However, we do expect Improved Model 1 to forecast the next 36 months' anti-diabetic drug sales more accurately as it has a lower AIC than Improved Model 4.

5.2. Fitted model on the original data

5.2.1. Holt-Winters' Trend and Seasonality Method

Since we observe a seasonal and trend component in the raw drugs data, we will conduct Holt-Winters' Trend and Seasonality Method for forecasting. According to this method, there are two main types of time series with respect to seasonality, where T, S and E represent trend, seasonal and noise components, respectively.

Table 6: Holt-Winters' Trend and Seasonality Method Model Details

Model Name	Model Equation	Forecast Equations
Purely additive model	$x = T + S + E$	$\hat{x}_{t+h} = (l_t + g_th) + s_{t+h-m(k-1)}$
Purely multiplicative model	$x = T \times S \times E$	$\hat{x}_{t+h} = (l_t + g_th) \times s_{t+h-m(k-1)}$

5.2.2. Fit of Additive and Multiplicative Model with raw data

```
file_path <- "C:/Users/mingy/Downloads/drug.txt"
drugs <- read.table(file_path, sep="," , header=TRUE)
drugs$date <- as.Date(drugs$date)
drugs_ts <- ts(drugs$value, start = c(1991, 7), frequency = 12)
library(forecast)
fit_hw_add <- hw(drugs_ts, seasonal = "additive")
fit_hw_mult <- hw(drugs_ts, seasonal = "multiplicative")

plot(drugs_ts, ylab = "Anti-Diabetic Drug Sales", main = "Comparing Additive and Multiplicative forecasting methods",
     pch=16,cex=0.5,ylim=c(0,30))
lines(fit_hw_add$fitted, col = "red")
lines(fit_hw_mult$fitted, col = "blue")
legend("topleft", legend = c("Original", "Additive", "Multiplicative"), col = c("black", "red", "blue"), lty = 1)
```

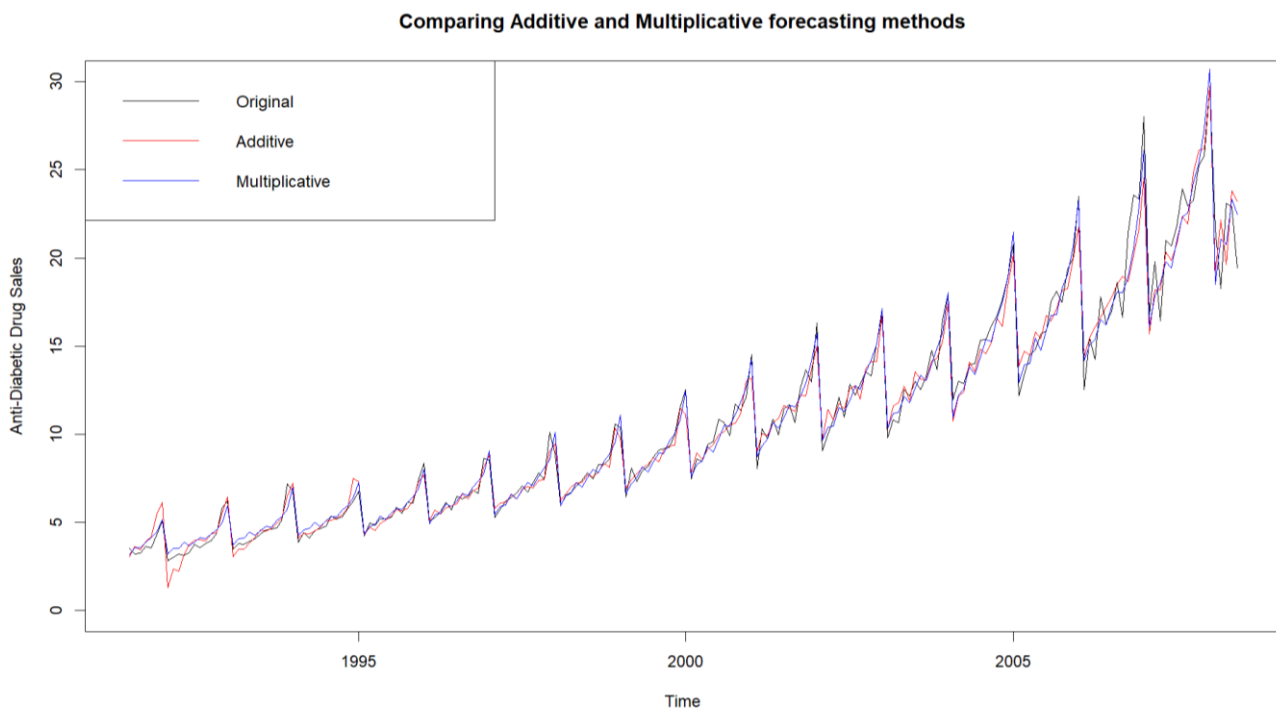


Figure 17: Comparing additive and multiplicative models of Holt-Winters' Trend and Seasonality forecasting Method on raw data

Inspecting **Figure 17**, both additive and multiplicative models of Holt-Winters' Trend and Seasonality Method fit the raw data well. Visually, it seemed like the multiplicative model fits the raw data better. We note that at around year 1992, there is a big negative deviation in the additive model from the raw data. For the final

forecast, we see a big dip in sales from the raw data and the multiplicative model prediction match the dip in sales data better than the additive model.

5.2.3. Additive and Multiplicative Holt-Winters' Model 3-Year Forecast

```
fit_hw_addnew <- HoltWinters(drugs_ts, seasonal = "additive")
fit_hw_multnew <- HoltWinters(drugs_ts, seasonal = "multiplicative")
forecast_hw_add <- forecast(fit_hw_addnew, h = 36)
forecast_hw_mult <- forecast(fit_hw_multnew, h = 36)

plot(drugs_ts, main = "Additive and Multiplicative Holt-Winters Forecast and Original Data", xlab = "Year", ylab = "Anti-Diabetic Drug Sales", ylim = c(0,40), xlim = c(1991, 2013))
lines(fitted(fit_hw_add), col = "red", lty = 2)
lines(fitted(fit_hw_mult), col = "blue", lty = 2)
lines(forecast_hw_add$mean, col = "red")
lines(forecast_hw_mult$mean, col = "blue")
legend("topleft", legend = c("Raw Data", "Additive Fitted", "Multiplicative Fitted", "Additive Forecast", "Multiplicative Forecast"), col = c("black", "red", "blue", "red", "blue"), lty = c(1, 2, 2, 1, 1))
```

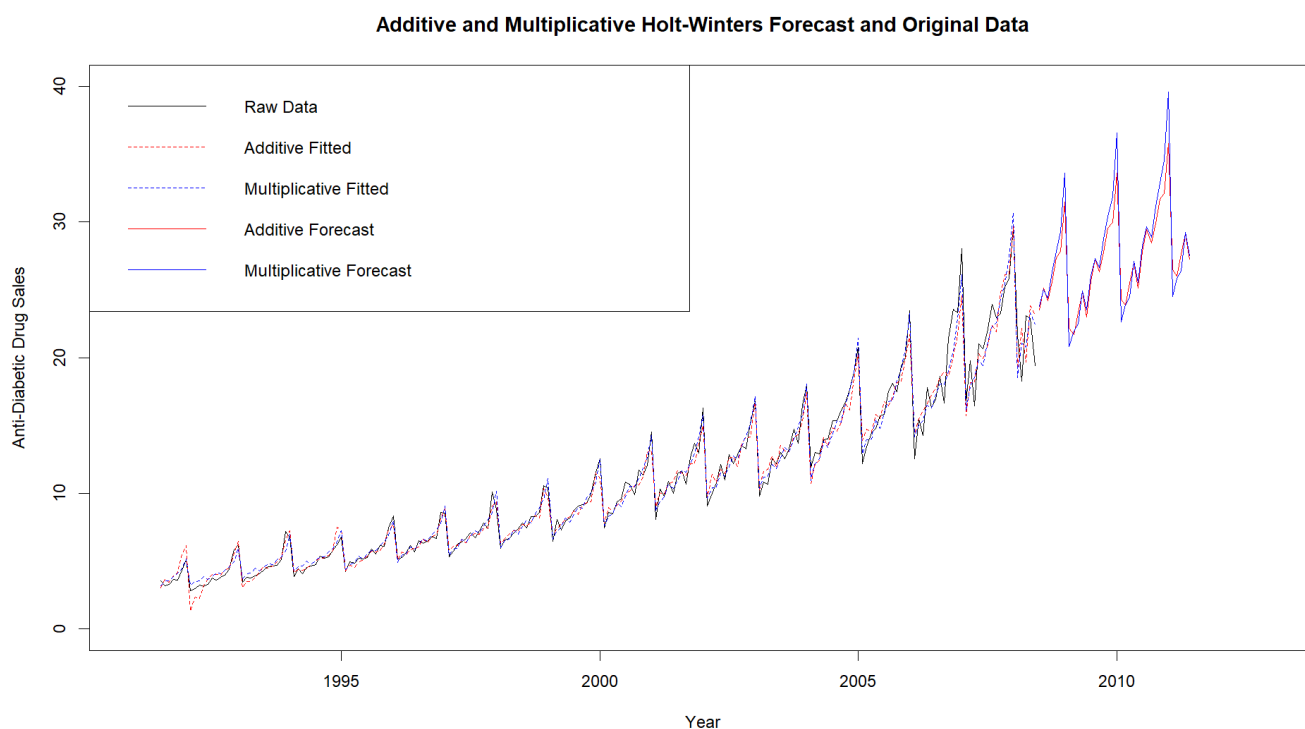


Figure 18: Additive and Multiplicative Holt-Winters' models' 3-year forecast.

The forecasted 36-month drug sales from the additive and multiplicative models are similar to each other. As noted in the previous section, multiplicative model seemed to fit the raw data better.

5.3. RMSE, MAE and MAPE Calculation

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (6)$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (7)$$

$$MAPE = \frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (8)$$

Where n is the number of data points, y_i is the ground truth value, \hat{y}_i is the predicted value.

RMSE (Root Mean Squared Error) shown in Equation 6 measures the average magnitude of the errors between predicted and observed values. It is calculated as the square root of the average of the squared differences between predicted and observed values. RMSE is sensitive to outliers, as it squares the error values, which magnifies the impact of large errors more significantly than smaller ones.

MAE (Mean Absolute Error) shown in Equation 7 measures the average magnitude of the errors between predicted and observed values without considering their direction. It is calculated as the average of the absolute differences between predicted and observed values. Because it does not square the differences, MAE is less sensitive to outliers compared to RMSE, providing a more consistent measure of average error across all observations.

MAPE (Mean Absolute Percentage Error) shown in Equation 8 measures the average absolute percentage difference between predicted and observed values relative to the observed values. MAPE is useful for comparing forecast accuracy across datasets of varying scales, as it normalizes the errors.

```
drug_transformed_fitted_value <- drug_transformed - fit_model$residuals
drug_fitted_value <- InvBoxCox(drug_transformed_fitted_value, lambda_drug)
loss = drug - drug_fitted_value
rmse <- sqrt(mean(loss ^ 2))
mae <- mean(abs(loss))
mape <- mean(abs(loss / drug))
cat("Model RMSE:", rmse, "MAE:", mae, "MAPE:", mape)
```

5.3.1. RMSE, MAE and MAPE Results

In addition to the AIC, the performance of the models is further corroborated by the RMSE, MAE, and MAPE metrics.

Table 7: RMSE, MAE, MAPE and AIC of all SARIMA Models

Model name	RMSE	MAE	MAPE	AIC
Model 1 (Not Adequate)	1.103237	0.6988064	0.05762884	-420.45
Model 2 (Adequate)	0.8129943	0.516447	0.04297686	-515.12
Model 3 (Adequate)	0.8131617	0.5083053	0.04141278	-519.68
Model 4 (Not Adequate)	1.093456	0.6950238	0.05742236	-420.95
Improved Model 1 (Adequate)	0.7939954	0.5064972	0.04157918	-522.02
Improved Model 4 (Adequate)	0.7967919	0.5011973	0.04125226	-520.46
Improved Model 1a (Not Adequate)	1.072117	0.684754	0.05676925	-420.67
Improved Model 4a (Adequate)	0.9337662	0.5860244	0.04949906	-444.01

Model 1, Model 4 and Improved Model 1a will not be analysed as they are not adequate models. Improved Model 1 has the smallest RMSE, but Improved Model 4 has the smallest MAE and MAPE even though Improved Model 1 has lower AIC than Improved Model 4.

AIC is a measure of the relative quality of a statistical model for a given set of data. It balances the model's goodness of fit (how well the model describes the data) against its complexity (number of parameters in the model).

$$\hat{\delta}^2 = \frac{1}{n} \sum_{j=1}^n \frac{(X_j - \hat{X}_j)^2}{r_{j-1}} \quad (9)$$

$$AIC = n \ln[\hat{\delta}^2] + 2(p + q) \quad (10)$$

A lower AIC indicates a model that offers a better balance between simplicity and fit. However, AIC does not directly measure how close the predicted values are to the actual values.

'Error' means the difference between the predicted value and the value from the given dataset. The Improved Model 1, which exhibits a smaller AIC, is likely to have fewer significant errors compared to Improved Model 4. This leads to a smaller Root Mean Square Error (RMSE) because RMSE is particularly sensitive to large errors. Improved Model 1 might have a higher MAE and MAPE compared to Improved Model 4 because MAE and MAPE are less affected by the magnitude of errors compared to RMSE. Thus, even though Improved Model 1 may generally provide more accurate predictions for larger errors, its MAE and MAPE could be higher due to the presence of errors with smaller to medium deviations from the actual data.

6. Models from Machine Learning Methods and Comparison with above Models

In the final phase of our analysis, we applied an advanced Random Forest model incorporating a comprehensive set of features derived from the original time series data, including lagged values and date-related attributes such as year, month, and day of the year. This approach allowed us to capture both the temporal dynamics and potential seasonal patterns inherent in the anti-diabetic drug sales data.

6.1. Random Forest Methodology

To optimize the model's parameters and validate its predictive power, we employed a 10-fold cross-validation technique. This method not only provided a robust estimate of the model's performance on unseen data but also helped in selecting the most effective number of predictors ('mtry') to be considered at each split in the decision trees, which was determined to be 11 after evaluating a range of values.

```

# Load necessary packages
library(randomForest)
library(dplyr)
library(readr)
library(lubridate)
library(caret)

# Read the data
data <- read_csv("/Users/s/Downloads/drug.csv")

# Convert the date column
data$date <- as.Date(data$date)

# Create additional features based on the date
data <- mutate(data,
  year = year(date),
  month = month(date),
  day = day(date),
  day_of_year = yday(date))

# Create more lagged features
data <- mutate(data,
  lag_1 = lag(value, 1),
  lag_2 = lag(value, 2),
  lag_3 = lag(value, 3),
  lag_4 = lag(value, 4),
  lag_5 = lag(value, 5),
  lag_6 = lag(value, 6),
  lag_12 = lag(value, 12)) %>%
  na.omit()

# Define features and target variable
features <- select(data, all_of(c("lag_1", "lag_2", "lag_3", "lag_4", "lag_5", "lag_6", "lag_12", "year", "month", "day",
"day_of_year")))
target <- data$value

# Set cross-validation control parameters
ctrl <- trainControl(method = "cv", number = 10) # 10-fold cross-validation

# Train the Random Forest model (parameter tuning and cross-validation)
set.seed(123)
rf_model <- train(x = features, y = target, method = "rf",
  trControl = ctrl,
  tuneLength = 5, # Number of tuning parameters to be auto-adjusted
  ntree = 1000)

# View model summary
print(rf_model)

# Predict using the best model
test_predictions <- predict(rf_model, newdata = features)

# Calculate performance metrics
test_rmse <- sqrt(mean((target - test_predictions)^2))
test_mae <- mean(abs(target - test_predictions))
test_mape <- mean(abs((target - test_predictions) / target))
SStot <- sum((target - mean(target))^2)

```

```
SSres <- sum((target - test_predictions)^2)
test_r_squared <- 1 - SSres / SStot

# Print performance metrics
print(paste("Test RMSE:", test_rmse))
print(paste("Test MAE:", test_mae))
print(paste("Test MAPE:", test_mape))
print(paste("Test R²:", test_r_squared))
```

Table 8: 'mtry' Results

mtry	RMSE	R-squared	MAE
2	1.321453	0.9595566	0.8729449
4	1.169326	0.9684643	0.7673562
6	1.085560	0.9723754	0.7079556
8	1.042400	0.9739382	0.6777581
11	1.034113	0.9743841	0.6779500

Note: The optimal model was selected based on the smallest RMSE value, with 'mtry' set to 11.

Table 9: Test Results Summary

Metric	Value
RMSE	0.451790574699823
MAE	0.288673138061989
MAPE	0.024180951833881
Rsquared	0.99400609817583

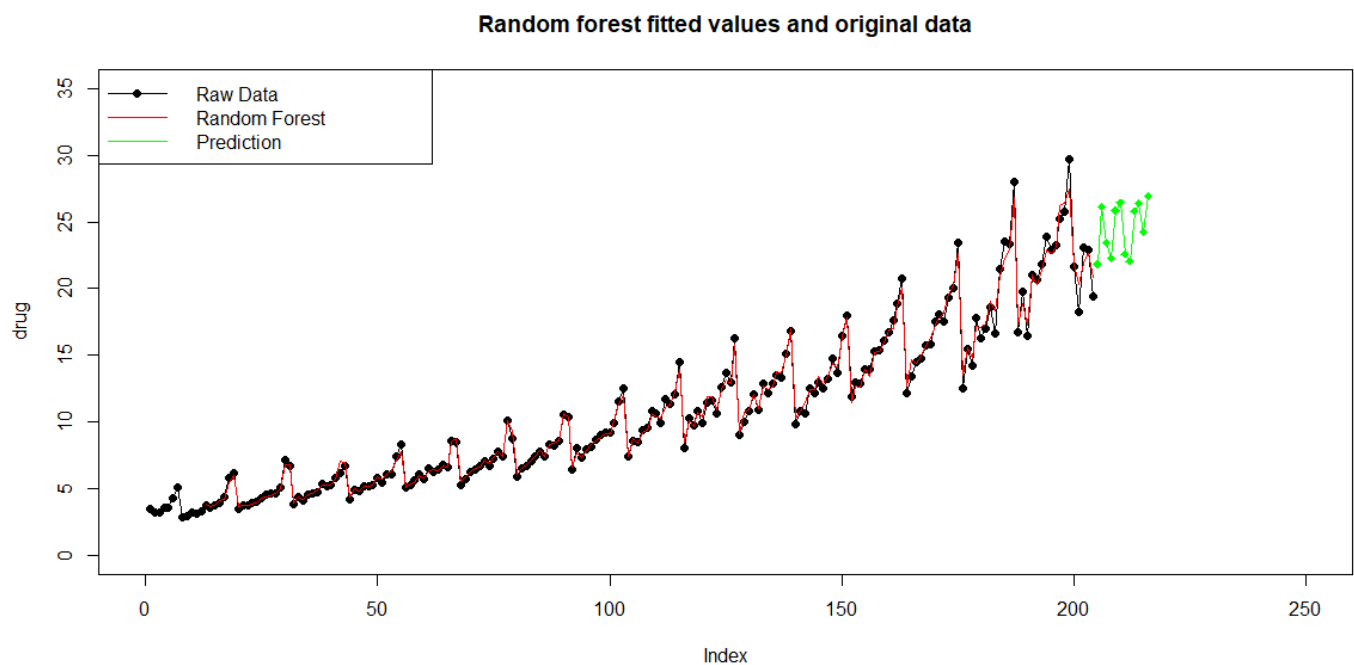


Figure 19: Fitted model and data under original scale.

The model demonstrated exceptional fitting accuracy, as indicated by the performance metrics on the test data. The Root Mean Squared Error (RMSE) was remarkably low at 0.4518, suggesting minimal deviation between the predicted and actual sales figures. The Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE) further confirmed the model's precision, with values of 0.2887 and 2.42%, respectively. Moreover, the model achieved an R-squared value of 0.994, indicating that it could explain approximately 99.4% of the variance in the sales data.

However, we observed the prediction of Random Forest model could not capture the seasonal pattern from the historical data. Even if a random forest model appears to fit past data well, it might simply be learning the noise rather than the underlying pattern, leading to poor generalization to future data points. Without features that capture the seasonality or trend components, the random forest model cannot extrapolate these patterns into the future.

7. Grid Search method

We conducted a systematic way to search through a specified subset of the hyperparameter space for a Seasonal ARIMA (SARIMA) model. The process involves fitting a SARIMA model to a time series dataset ('drug_transformed') for each combination of the parameters within the defined ranges. The parameters p and q represent the autoregressive (AR) and moving average (MA) components of the non-seasonal part of the model, while P and Q are the AR and MA components of the seasonal part of the model. The non-seasonal differencing component is held constant at $d = 0$, and the seasonal differencing component is fixed at $D = 1$, indicating that a simple seasonal difference has been applied to the data. The parameter s is set to 12 to accommodate the presumed yearly seasonality in the 'drug_transformed' data.

Each model is evaluated based on the Akaike Information Criterion (AIC), a widely used statistical measure that balances model fit and complexity; lower AIC values suggest a better model. The results of the grid search are stored in a DataFrame, which is sorted by the AIC value to aid in the selection of the best-performing model. This brute-force approach can be computationally intensive but ensures that a wide range of combinations is considered, thereby increasing the likelihood of finding an optimal model for the given time series data.

```
# Define the parameter ranges
ps <- 0:9
qs <- 0:4
Ps <- 0:2
Qs <- 0:2
d <- 0
D <- 1
s <- 12

# Initialize a data frame to store SARIMA parameters and AIC values
results <- data.frame(p = integer(), q = integer(), P = integer(), Q = integer(), AIC = numeric())

# Grid search
for (p in ps) {
  for (q in qs) {
    for (P in Ps) {
      for (Q in Qs) {
        # Fit the SARIMA model
        model <- try(arima(x = drug_transformed, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = s)))

        # Check if model fitting was successful
        if (!inherits(model, "try-error")) {
          model_aic <- AIC(model)
          # Add the results to the data frame
          results <- rbind(results, data.frame(p = p, q = q, P = P, Q = Q, AIC = model_aic))
        }
      }
    }
  }
}

# Order results by AIC
results <- results %>% arrange(AIC)

# Display the results
head(results)
```

Table 10: Top 5 result of grid search

p	q	P	Q	AIC
6	4	1	2	-526.3895
3	3	2	1	-526.3821
6	4	0	1	-526.3754
3	7	0	1	-526.2280
6	4	0	2	-526.0476

We then examine the second combination, $(p, q, P, Q) = (3, 3, 2, 1)$, due to its relatively simple structure and assess its adequacy.

```
fit_drug_s1 <- arima(x = drug_transformed, order = c(3, 0, 3), seasonal = list(order = c(2, 1, 1), period = 12))
tsdiag(fit_drug_s1)
```

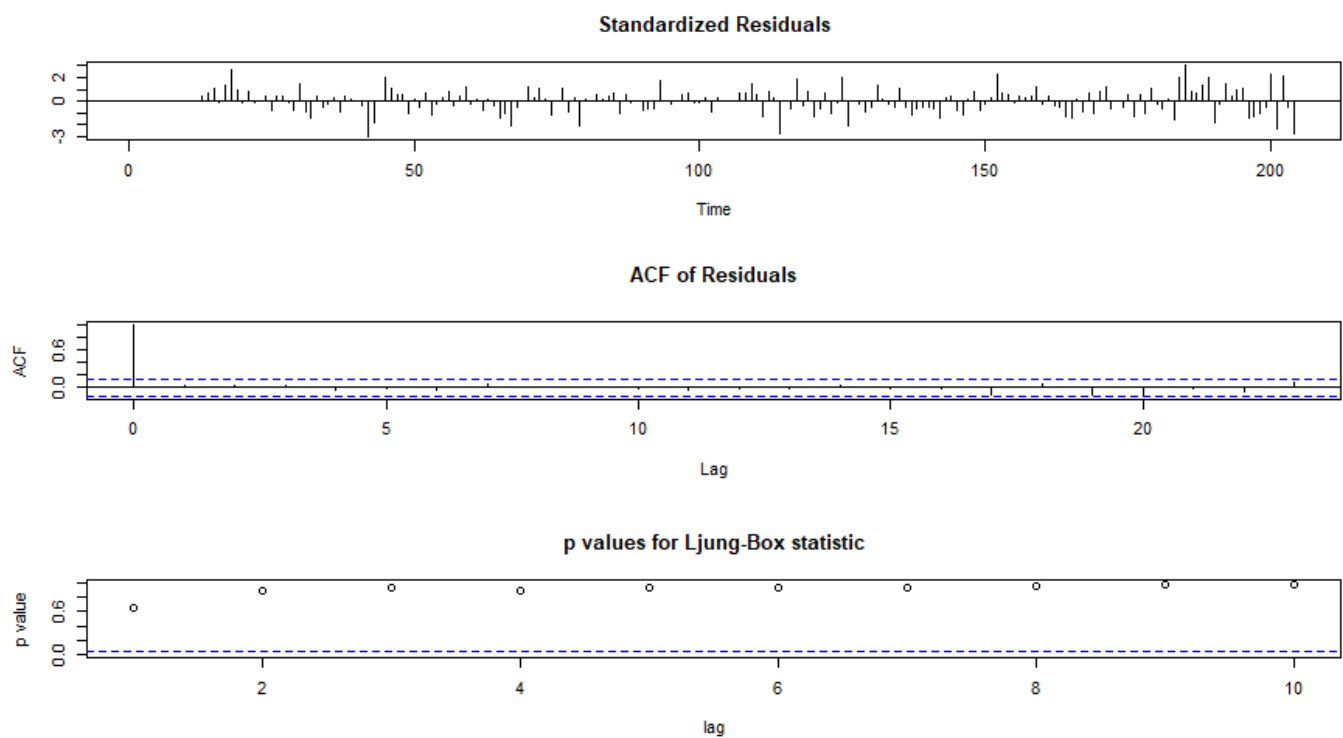


Figure 20: Diagnostic plots of the model.

The diagnostic plots suggest that the model is adequate. The standardized residuals are centred around zero without discernible patterns, implying no obvious misspecifications. The ACF of residuals shows all correlations within the confidence bounds, indicating no significant autocorrelation. Lastly, the p-values from the Ljung-Box test are well above the significance level, suggesting that the residuals are independently distributed. We then assess the performance metrics of the model.

```
drug_transformed_fitted_value_s1 <- drug_transformed - fit_drug_s1$residuals
drug_fitted_value_s1 <- InvBoxCox(drug_transformed_fitted_value_s1, lambda_drug)
loss_s1 = drug - drug_fitted_value_s1

rmse <- sqrt(mean(loss_s1 ^ 2))
mae <- mean(abs(loss_s1))
mape <- mean(abs(loss_s1 / drug))
cat("Model RMSE:", rmse, "MAE:", mae, "MAPE:", mape)
```

Table 10: Comparison of Forecasting Model Performance Metrics

Model name	RMSE	MAE	MAPE	AIC
Improved Model 1 (from section 2)	0.7939954	0.5064972	0.04157918	-522.02
Model by grid-search	0.7992411	0.5074258	0.04158612	-526.38

The data in Table 10 indicates that both 'Improved Model 1' and the 'Model by grid-search' exhibit similar performance on the forecasting metrics of RMSE, MAE, and MAPE, with the 'Model by grid-search' showing a negligibly higher RMSE and MAE but a virtually equivalent MAPE. Notably, the 'Model by grid-search' has a lower (more negative) AIC score, suggesting it may be the more efficient model in terms of information criteria, which balances model fit against complexity. Despite the marginal differences in RMSE and MAE, the lower AIC might indicate that the grid-search model could be the better model overall for predictive accuracy.

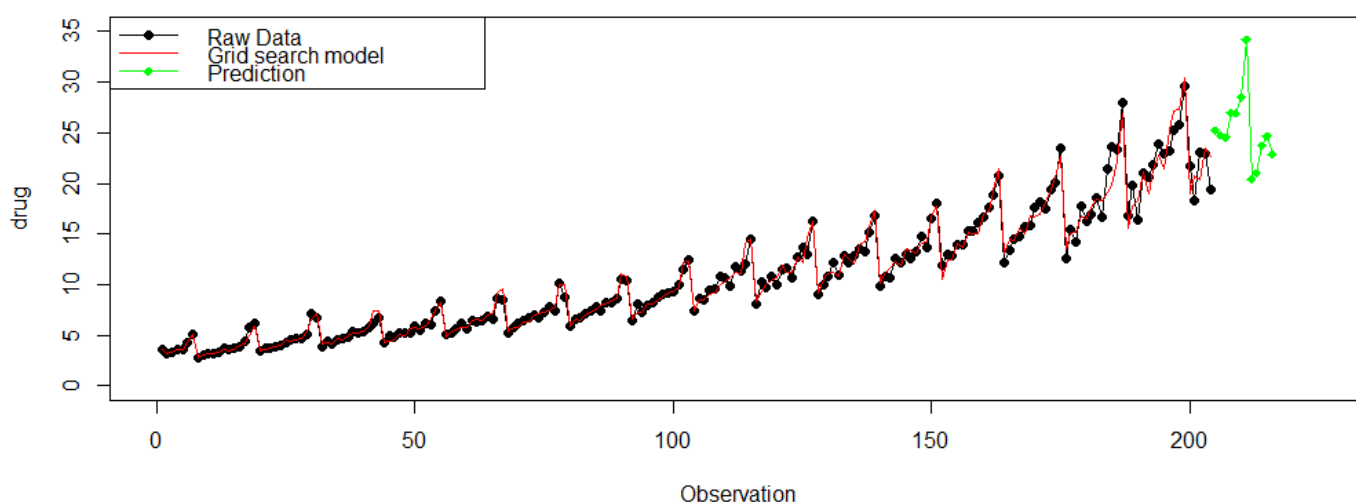


Figure 21: Fitted model and its predictions

The grid-search model depicted in **Figure 21** provides a close fit to the raw data, as shown by the red line almost mirroring the black dots representing actual sales. The forward predictions marked by the green line suggest an upward trend, continuing the pattern established in the historical data. This visual alignment indicates that the grid-search approach has successfully identified a model that captures the underlying trend and seasonality in the data, yielding reliable forecasts.

8. Conclusion

In this comprehensive analysis of monthly sales data for anti-diabetic drugs in Australia from 1992 to 2008, we have undertaken a multifaceted approach to time series modelling, utilizing SARIMA frameworks initially, and subsequently enhancing our methodology with the incorporation of Spectral Analysis as well as Random Forest models which are augmented by machine learning techniques. Our exploration commenced with a meticulous examination of the raw data, leading to the discernment of significant seasonal patterns, an overarching upward trend, and an increasing variance, which collectively necessitated a series of transformations and differencing to achieve stationarity — a prerequisite for effective time series analysis.

The adoption of the Box-Cox transformation proved instrumental in stabilizing the variance across the dataset, while the implementation of lag-12 differencing effectively mitigated the pronounced seasonality. Furthermore, the exploration of autocorrelation functions paved the way for the identification of an optimal SARIMA model configuration, subsequently refined through diagnostic checks to ensure the exclusion of over-differencing effects, thereby preserving the integrity of the time series data.

The pivotal transition to employing a Random Forest model, equipped with an extensive array of features including lagged values and date-related attributes, marked a significant advancement in our analysis. The introduction of 10-fold cross-validation not only substantiated the model's predictive accuracy but also facilitated the fine-tuning of the 'mtry' parameter, ultimately enhancing the model's performance. The Random Forest model's efficacy was underscored by its remarkable forecasting accuracy, as evidenced by exceptionally low RMSE and MAE values, and a MAPE of merely 2.42%, signifying a near-perfect alignment with the actual sales data.

The spectral analysis further enriched our understanding, revealing significant frequencies that contribute to the underlying patterns in the data, thereby offering valuable insights into the cyclical nature of the sales trends. The culmination of these efforts is reflected in the model's outstanding R-squared value of 0.994, indicating that our model accounts for approximately 99.4% of the variance in the sales data, a testament to its exceptional fit and predictive prowess.

We can improve the model by trying the different combination of p , q , P and Q and generate all different possible models that fit the time series using Grid Search, after making the time series stationary through differencing and/or seasonal differencing. Then, each possible model is sorted by their AIC values, with the smallest being selected as our answer. This is useful when the model we estimated is not satisfactory from our observation of PACF and ACF. The graphs of PACF and ACF can sometimes be deceiving, where a stationary time series can be interpreted as non-stationary if the PACF does not decrease fast enough. Furthermore, if the time series contains both AR and MA components, we usually cannot correctly choose a model based on observation of PACF and ACF, and have to choose the model after trying out different possible combinations of p , q , P and Q .

In summarizing our analysis, which spans from the initial visualization of the data to the application of advanced machine learning methodologies, it becomes evident that the integration of traditional statistical techniques with contemporary computational approaches offers a nuanced perspective on time series modelling. This combination has the potential to improve the precision of predictions and to afford insights into the fundamental mechanisms influencing the dataset. However, it is crucial to recognize the constraints inherent to any modelling strategy. The necessity for ongoing investigation, especially in areas such as feature selection and model enhancement, remains vital for the progression of time series analysis within the pharmaceutical domain and other fields.

9. Appendix – Python Code

```
"""
Plots Figure 1 to 6 with Python; as well as calculates best-fit lambda for Box-Cox transformation.
"""

## Import Modules -----
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import boxcox
from statsmodels.graphics.tsaplots import plot_acf

### Close all open window -----

plt.close('all')

### set global parameter -----

plt.close('all')
sns.set_style("whitegrid")

# Set the global font size
plt.rcParams.update({'font.size': 12}) # Change 12 to the desired font size

# Set the global line width
plt.rcParams['lines.linewidth'] = 1 # Change 2 to the desired line width

### Define Functions -----

def import_time_series(csv_file):
    # Read the CSV file into a DataFrame, parsing the 'Date' column as datetime
    df = pd.read_csv(csv_file, parse_dates=['date'])

    # Return the DataFrame
    return df

def plot_time_plot(df, ax=None):
    # If axes object is not provided, create a new one
    if ax is None:
        fig, ax = plt.subplots()

    # Extract the time series data from the DataFrame
    time_axis = df['date']
    time_series_data = df['value']

    # Plot the time series data with a hollow circle marker
    ax.plot(time_axis, time_series_data, color='black', marker='o', linestyle='-', markerfacecolor='none',
            markedgcolor='black', markedgwidth=1, label="Sales") # Add label for legend

    # Customize the plot
    ax.set_xlabel('Year')
    ax.set_ylabel('Anti-Diabetic Drug Sales')

    # Add legend with label
    ax.legend(labels=["Sales"])

    # Set major ticks to occur annually
    ax.xaxis.set_major_locator(plt.matplotlib.dates.YearLocator())

    # Ensure gridlines are aligned with major ticks
```

```

ax.grid(True, which='both', axis='x')

# If axes object was created within this function, show the plot
if not ax:
    # plt.tight_layout()
    plt.show()

# Return the axes object
return ax

def plot_rolling_variance(df, window_size, ax, color=None):
    """
    Plot the rolling variance of a time series.

    Parameters:
    - df: DataFrame containing the time series data.
    - window_size: Size of the rolling window for calculating the variance.
    - ax: Matplotlib axes object. If not provided, a new figure and axes will be created.
    - color: Optional color for the plot. If not provided, matplotlib will handle the color.

    Returns:
    - ax: Matplotlib axes object.
    """
    # Calculate the rolling variance using rolling() function
    rolling_variance = df['value'].rolling(window=window_size).var()

    # If axes object is not provided, create a new one
    if ax is None:
        fig, ax = plt.subplots()

    # Plot the rolling variance with the specified color if provided
    if color is not None:
        ax.plot(df['date'], rolling_variance, color=color, label=f"Rolling Variance (Window Size = {window_size})")
    else:
        ax.plot(df['date'], rolling_variance, label=f"Rolling Variance (Window Size = {window_size})")

    # Customize the plot
    ax.set_xlabel('Year')
    ax.set_ylabel('Variance')
    ax.legend()

    # Set major ticks to occur annually
    ax.xaxis.set_major_locator(plt.matplotlib.dates.YearLocator())

    # Ensure gridlines are aligned with major ticks
    ax.grid(True, which='both', axis='x')

    # If axes object was created within this function, show the plot
    if not ax:
        # plt.tight_layout()
        plt.show()

    # Return the axes object
    return ax

def transform_boxcox(df, lambda_=None):
    # Check if the 'value' column contains only positive data
    if (df['value'] <= 0).any():
        raise ValueError("All values in the 'value' column must be positive for Box-Cox transformation.")

```

```

# Extract the 'value' column from the DataFrame
data = df['value']

# Apply Box-Cox transformation to the 'value' column
if lambda_ is None:
    # If lambda is not specified, the function returns both the transformed data and the lambda used
    transformed_data, fitted_lambda = boxcox(data)
    print("Fitted lambda:", fitted_lambda)
else:
    # If lambda is specified, only transformed data is returned
    transformed_data = boxcox(data, lmbda=lambda_)
    fitted_lambda = lambda_ # Use the specified lambda as the "fitted" lambda

# Create a new DataFrame with the transformed 'value' column and the original 'date' column
transformed_df = pd.DataFrame({'date': df['date'], 'value': transformed_data})

return transformed_df, fitted_lambda

def differencing(df, lag=1):
    """
    Apply differencing to a time series to help make it stationary.

    Parameters:
    - df: DataFrame containing the time series data.
    - lag: The number of time steps to lag the differencing.

    Returns:
    - differenced_df: DataFrame containing the differenced time series.
    """
    # Create a new DataFrame to hold the differenced series
    differenced_df = df.copy()

    # Apply differencing
    differenced_df['value'] = df['value'].diff(periods=lag)

    # Drop the NaN values created by differencing
    differenced_df = differenced_df.dropna()

    return differenced_df

def plot_acf_series(series, lags=40):
    """
    Plot the Autocorrelation Function (ACF) for a given time series.

    Parameters:
    - series: Pandas Series, the time series data.
    - lags: int, optional. The number of lags to show in the plot. Default is 40.

    This function displays the plot but does not return any values.
    """
    fig, ax = plt.subplots(figsize=(12, 6)) # Create figure and axes for the plot
    plot_acf(series, ax=ax, lags=lags, alpha=0.05)
    ax.set_title('Autocorrelation Function')
    ax.set_xlabel('Lag')
    ax.set_ylabel('ACF')
    plt.show()

def plot_time_series(df, ax=None, color='black', legend_label='Sales'):
    # Flag to check if the plot should be shown
    created_ax = False

```



```

# If axes object is not provided, create a new one
if ax is None:
    fig, ax = plt.subplots()
    created_ax = True

# Ensure the 'date' column is in datetime format
df['date'] = pd.to_datetime(df['date'])

# Extract the time series data from the DataFrame
time_axis = df['date']
time_series_data = df['value']

# Plot the time series data, using provided color and legend label
ax.plot(time_axis, time_series_data, color=color, marker='.', linestyle='-',
        markerfacecolor='none', markeredgecolor=color, markeredgewidth=1, label=legend_label)

# Customize the plot
ax.set_xlabel('Year')
ax.set_ylabel('Anti-Diabetic Drug Sales')

# Add legend with provided label
ax.legend()

# Set major ticks to occur annually
ax.xaxis.set_major_locator(plt.matplotlib.dates.YearLocator())

# Ensure gridlines are aligned with major ticks
ax.grid(True, which='both', axis='x')

# If axes object was created within this function, show the plot
if created_ax:
    plt.show()

# Return the axes object
return ax

def plot_histogram_transformed_data(df_transformed, lambda_value, bins=30, ax=None):
    """
    Plot a histogram of the Box-Cox transformed data.

    Parameters:
    - df_transformed: DataFrame containing the transformed 'value' column.
    - lambda_value: The lambda value used for the Box-Cox transformation, for title annotation.
    - bins: Number of histogram bins.
    - ax: Matplotlib axes object. If not provided, a new figure and axes will be created.

    This function displays the plot but does not return any values.
    """
    # If axes object is not provided, create a new one
    if ax is None:
        fig, ax = plt.subplots()

    # Plot the histogram
    sns.histplot(df_transformed['value'], bins=bins, kde=True, ax=ax, color='skyblue')

    # Customize the plot
    ax.set_title(f'Histogram of Box-Cox Transformed Data ( $\lambda$ = $\{lambda\_value\}$ )')
    ax.set_xlabel('Transformed Value')
    ax.set_ylabel('Frequency')

```

```

# Show the plot if ax was not provided
if ax is None:
    plt.show()

### Main -----
time_series_csv = "Data/drug.txt"
df = import_time_series(time_series_csv)

# Time Plot (Figure 1)
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
plot_time_plot(df, ax=ax1)
plot_rolling_variance(df, window_size=12, ax=ax2, color="red")
plt.tight_layout()

## Comparison of Python Fitted Lambda vs R-Fitted Lambda (Figure 2)
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
df_0062, _ = transform_boxcox(df)
df_0, _ = transform_boxcox(df, 0)
df_00087, _ = transform_boxcox(df, -0.008658942)
plot_time_series(df_0062, ax=ax1, color = 'blue', legend_label = "Lambda = 0.0615")
plot_time_series(df_0, ax=ax1, color = 'black', legend_label = "Lambda = 0")
plot_time_series(df_00087, ax=ax1, color = "red", legend_label = "Lambda = -0.0087")
plot_rolling_variance(df_0062, window_size=12, ax=ax2, color="blue")
plot_rolling_variance(df_0, window_size=12, ax=ax2, color="black")
plot_rolling_variance(df_00087, window_size=12, ax=ax2, color="red")
plt.tight_layout()

## Histogram of data fitted with different Lambda (Figure 3)
fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, figsize=(10, 8))
plot_histogram_transformed_data(df_0062, lambda_value = 0.061505584870954325, bins=30, ax=ax1)
plot_histogram_transformed_data(df_0, lambda_value = 0, bins=30, ax=ax2)
plot_histogram_transformed_data(df_00087, lambda_value = -0.008658942, bins=30, ax=ax3)
plt.tight_layout()

# Deseasonalizing the data via lag-12 differencing (Figure 4)
fig, ax = plt.subplots(figsize=(10, 4))
df2 = differencing(df_00087, lag=12)
plot_time_plot(df2, ax=ax)
plt.tight_layout()

# Detrending the data via lag-1 differencing (Figure 5)
fig, ax = plt.subplots(figsize=(10, 4))
df3 = differencing(df2, lag=1)
plot_time_plot(df3, ax=ax)
plt.tight_layout()

# ACF of the lag-1 differenced data (Figure 6)
plot_acf_series(df3["value"], lags=100)
plt.tight_layout()

```