Basic Non Seasonal ARIMA

Dataset used The data I have used is the dataset “UKNonDurables” which is publicly available. The dataset consists of quarterly consumption of Non Durables in UK from 1st quarter of 1955 to 4th quarter of 1988. The dataset consists of 136 total observations. The documentation is found in the link https://vincentarelbundock.github.io/Rdatasets/doc/AER/UKNonDurables.html (https://vincentarelbundock.github.io/Rdatasets/doc/AER/UKNonDurables.html).

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.6.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.6.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(moments)
```

```
#Reading the data
data = read.table('UKNonDurables.csv',sep=',',header=T,stringsAsFactors = F)
head(data)
```

	X <int>	time <dbl>	value <int>
1	1	1955.00	24030
2	2	1955.25	25620

	X <int>	time <dbl>	value <int>
3	3	1955.50	26209
4	4	1955.75	27167
5	5	1956.00	24620
6	6	1956.25	25972

6 rows

Step 1: Preparing the dataset

```
#Only keeping two columns : Time and Value
data = data[, c('time','value')]
head(data)
```

	time <dbl>	value <int>
1	1955.00	24030
2	1955.25	25620
3	1955.50	26209
4	1955.75	27167
5	1956.00	24620
6	1956.25	25972

6 rows

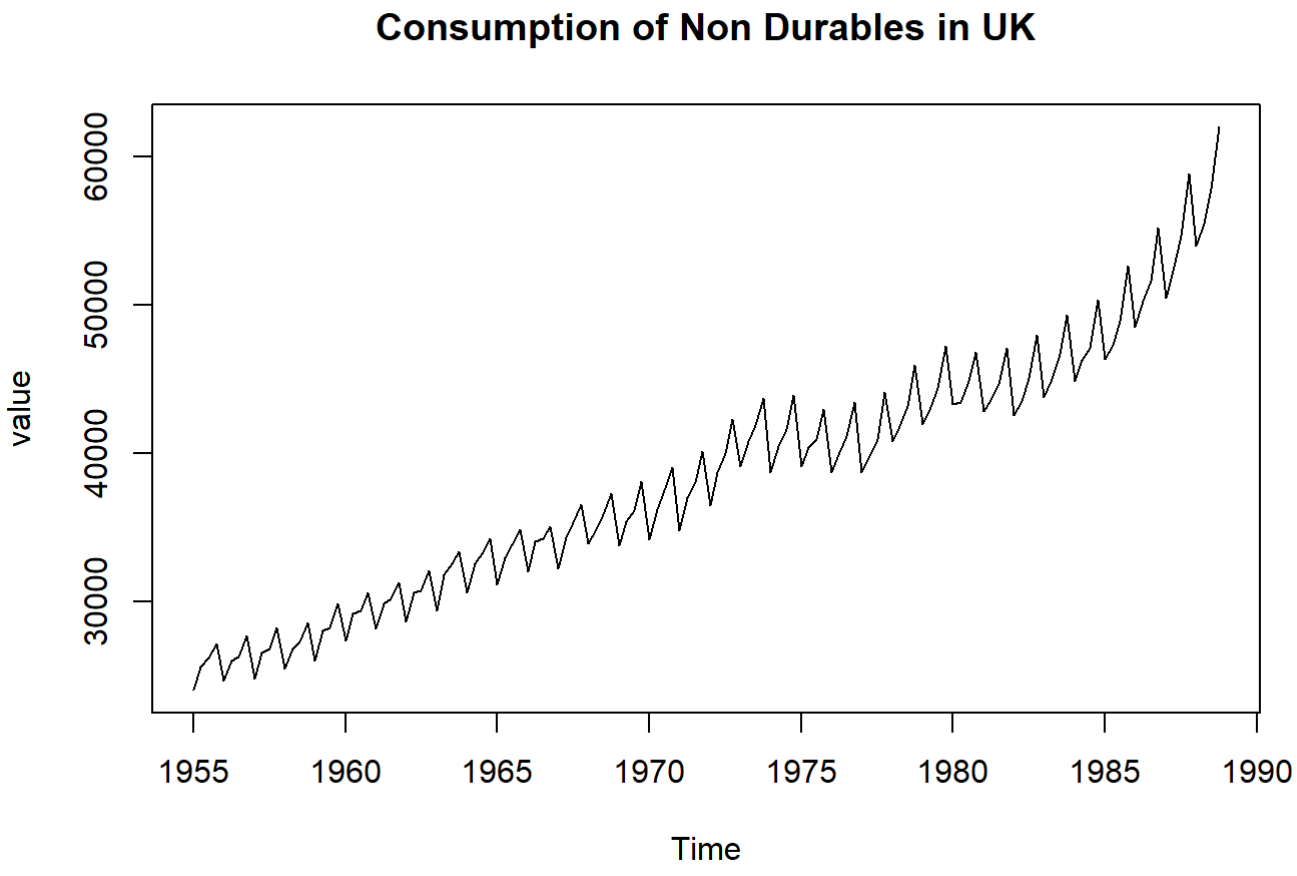
```
#Ordering data based on time
data = data[order(data$time),]

#Check for missing data
paste("Number of Missing values is", sum(is.na(data)))
```

```
## [1] "Number of Missing values is 0"
```

```
#So there are no missing values in the dataset

#Cleaning data
start_date = data$time[1]
value = ts(data$value, start = start_date,freq =4)
value = tsclean(value)
data$value = value

#Dividing data into train and test sets where test set contains the last 6 observations
tm = data$time

period<- 6
n = length(value)
m = n - period


#Split data into train data and test data with test data having last 6 observations

train_lim = length(data$value) - period
data['log_value'] = log(data$value)

train_data = data[1:train_lim,]
test_data = data[(train_lim+1):length(data$value),]
```

Step 2 : Plot of the total data

```
#Time series plot of the whole data
ts.plot(value,main = "Consumption of Non Durables in UK")
```



Step 3: Descriptive statistics of the data

```
value = ts(train_data$value, start = start_date, frequency = 4)
descriptive_stats = list('Mean' = mean(value),'Median' = median(value),'Minimum' = min(valu
e),
                         'Maximum' = max(value), 'Standard Deviation' = sd(value) )
descriptive_stats$skewness = skewness(value)
descriptive_stats$kurtosis = kurtosis(value)

f <-function(x,mean1,sd1)
{
  #print (paste (mean1,sd1))
  count = 0
  for (i in c(1:length(x)))
  {
    if ((x[i]>=(mean1-3*sd1)) && (x[i]<=(mean1+3*sd1)))
      count = count+1
  }

  return ((count*100)/(length(x)))
}
descriptive_stats$perct_within_3sd = f(value,descriptive_stats$Mean,descriptive_stats$`Standa
rd Deviation`)

str(descriptive_stats)
```

```
## List of 8
##  $ Mean              : num 37513
##  $ Median            : num 37747
##  $ Minimum           : num 24030
##  $ Maximum           : num 55152
##  $ Standard Deviation: num 7534
##  $ skewness          : num 0.0895
##  $ kurtosis          : num 2.05
##  $ perct_within_3sd  : num 100
```

So, the descriptive statistics of the train dataset can be seen from above. Note, that the data has a wide span of values and a high standard deviation of 7534.

Step 4: Testing for stationarity/Coming up with Transformations to make it stationary First, let's check for the stationarity of the data using the three tests(ADF, KPSS and PP).

```
adf.test(value)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  value
## Dickey-Fuller = -3.1302, Lag order = 5, p-value = 0.1066
## alternative hypothesis: stationary
```

```
kpss.test(value)
```

```
## Warning in kpss.test(value): p-value smaller than printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  value
## KPSS Level = 2.6633, Truncation lag parameter = 4, p-value = 0.01
```

```
pp.test(value)
```

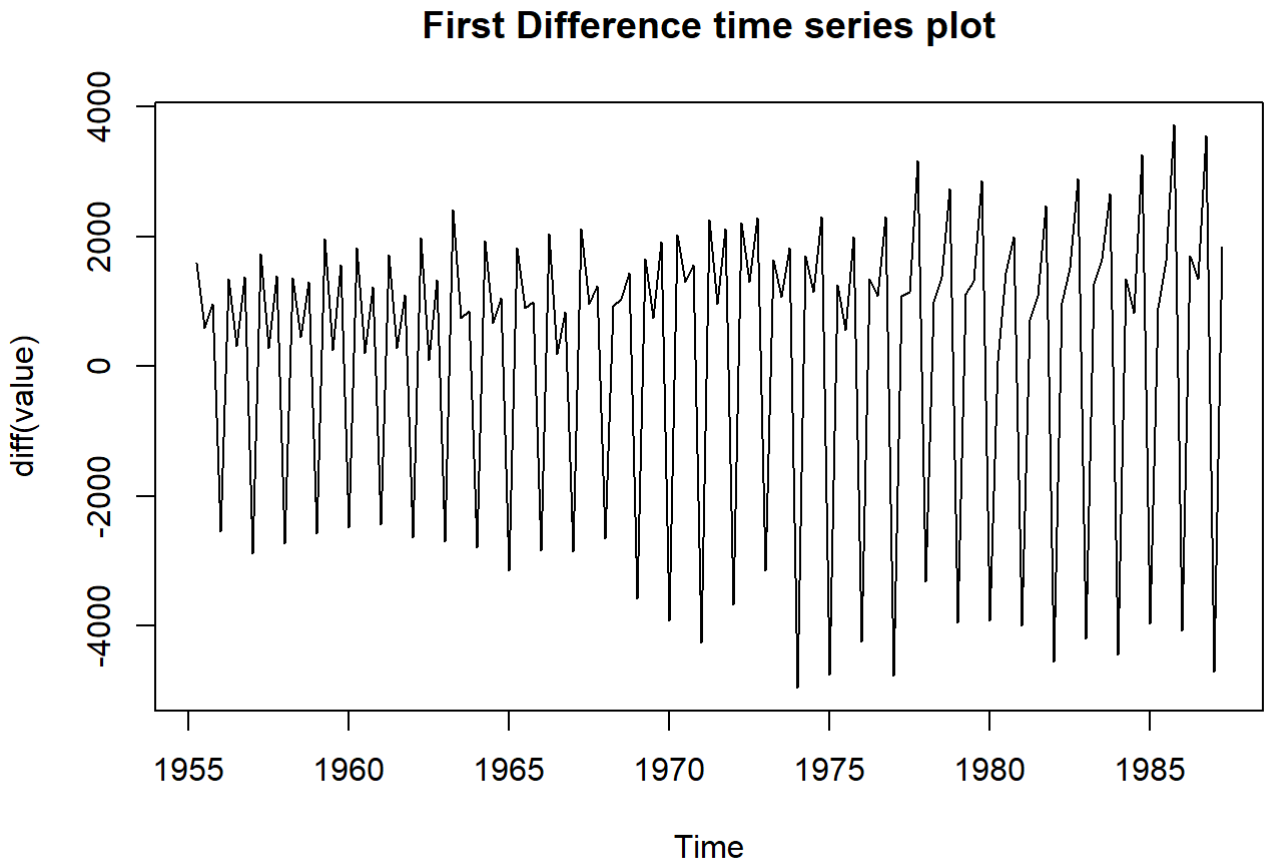
```
## Warning in pp.test(value): p-value smaller than printed p-value
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  value
## Dickey-Fuller Z(alpha) = -139.63, Truncation lag parameter = 4, p-value
## = 0.01
## alternative hypothesis: stationary
```

Although, the PP test says that the data is stationary with a 5% level of significance, we cannot say that the data is stationary as the other two tests reject stationarity.

Step 5: Coming up with the transformation of data We have already seen that the values of the data span a wide range. So, a log transformation may be a good idea. Anyways we plot the time series of first difference of the data as follows:

```
ts.plot(diff(value),main = "First Difference time series plot")
```



We see that the variance does not seem constant. We also had seen that the range of the data is very large.So, we take the log transform of the data and proceed.

```
value = log(value)

adf = adf.test(value)
print (adf)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  value
## Dickey-Fuller = -2.8073, Lag order = 5, p-value = 0.2409
## alternative hypothesis: stationary
```

```
kpss = kpss.test(value)
```

```
## Warning in kpss.test(value): p-value smaller than printed p-value
```

```
print (kpss)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  value
## KPSS Level = 2.6535, Truncation lag parameter = 4, p-value = 0.01
```

```
pp = pp.test(value)
```

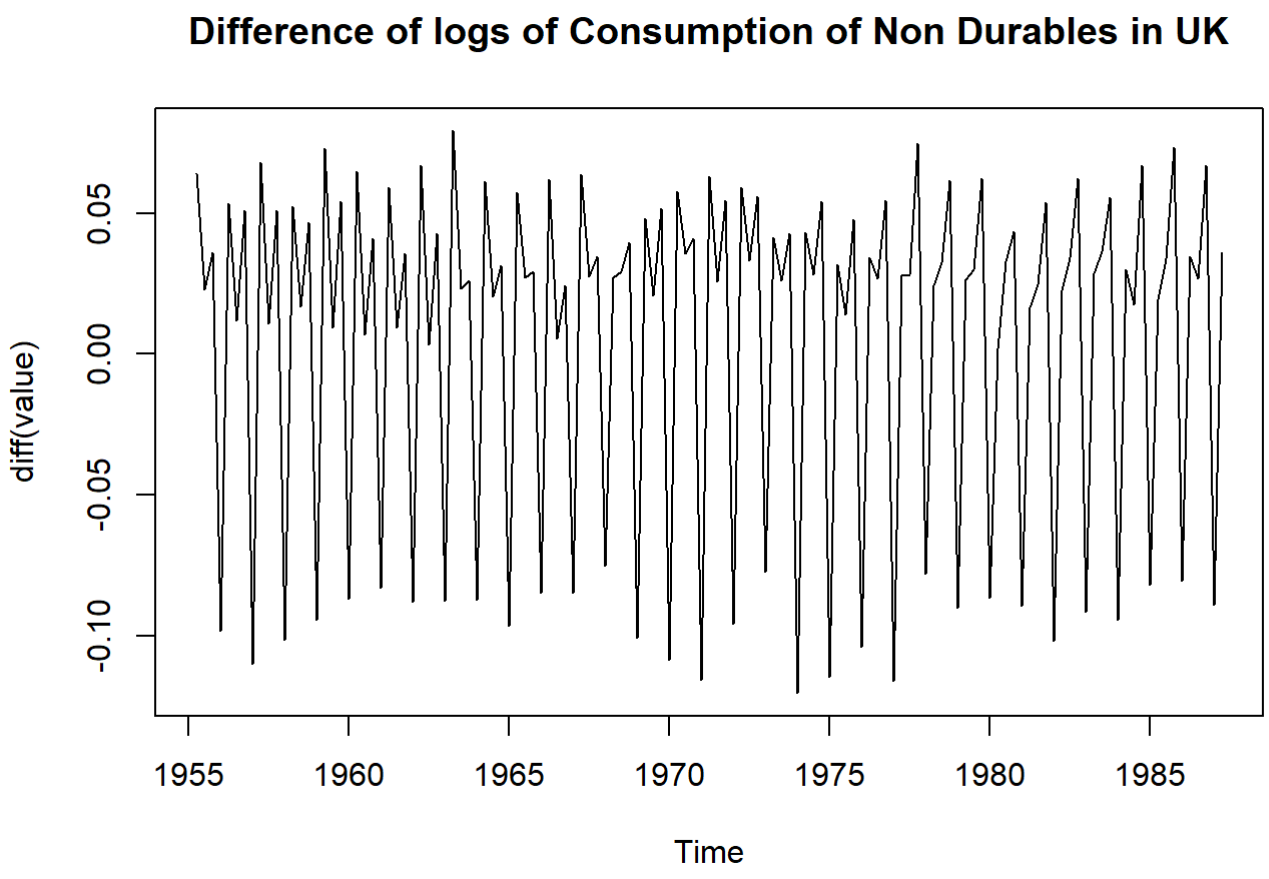
```
## Warning in pp.test(value): p-value smaller than printed p-value
```

```
print (pp)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  value
## Dickey-Fuller Z(alpha) = -136.44, Truncation lag parameter = 4, p-value
## = 0.01
## alternative hypothesis: stationary
```

The log of the data is not stationary but let's see the first difference of the log of the values.

```
ts.plot(diff(value),main = "Difference of logs of Consumption of Non Durables in UK")
```



This look's much better than the first difference of the values. Let's do the stationarity tests on these.

```
adf = adf.test(diff(value))
```

```
## Warning in adf.test(diff(value)): p-value smaller than printed p-value
```

```
print (adf)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(value)
## Dickey-Fuller = -4.1495, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
kpss = kpss.test(diff(value))
```

```
## Warning in kpss.test(diff(value)): p-value greater than printed p-value
```

```
print (kpss)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  diff(value)
## KPSS Level = 0.039654, Truncation lag parameter = 4, p-value = 0.1
```

```
pp = pp.test(diff(value))
```

```
## Warning in pp.test(diff(value)): p-value smaller than printed p-value
```

```
print (pp)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  diff(value)
## Dickey-Fuller Z(alpha) = -151.78, Truncation lag parameter = 4, p-value
## = 0.01
## alternative hypothesis: stationary
```

As can be seen from the tests, the first difference of the log transform of the data is stationary. So, we get the difference order as 1. We also confirm this with the function “ndiffs” which give the difference order to make it stationary for a given test(KPSS by default).

Step 6: Fitting ARIMA models a) AIC model

```
#Fitting model which has minimum AIC
d = 1
model_fit = auto.arima(value,seasonal = F,d=d,max.p = 5, max.q = 5, ic = "aic")
model_fit
```

```
## Series: value
## ARIMA(2,1,2) with drift
##
## Coefficients:
##           ar1      ar2      ma1     ma2   drift
##       -0.0399  -0.6266  -1.1314  0.7224  0.0057
## s.e.   0.0822   0.0823   0.0903  0.0583  0.0010
##
## sigma^2 estimated as 0.001157:  log likelihood=254
## AIC=-495.99   AICc=-495.3   BIC=-478.83
```

b)BIC model

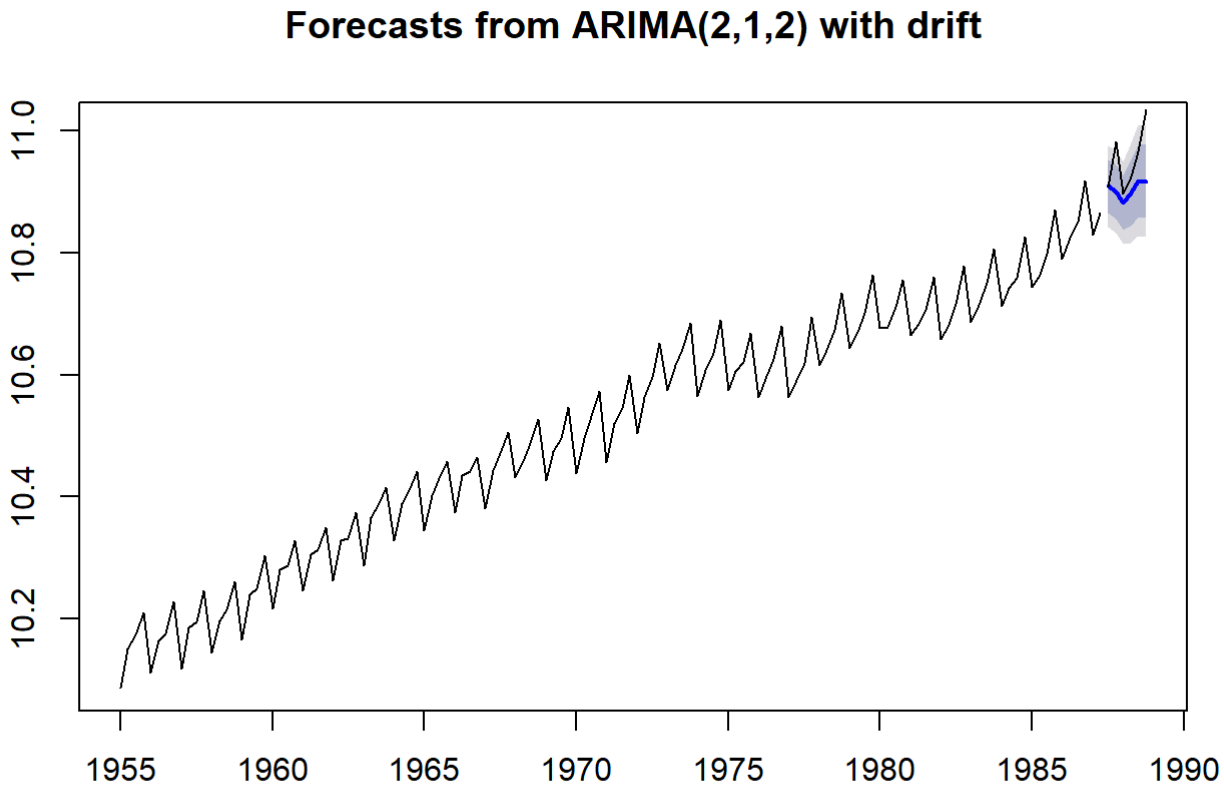
```
model_fit = auto.arima(value,seasonal = F,d=d,max.p = 5, max.q = 5, ic = "bic")
model_fit
```

```
## Series: value
## ARIMA(2,1,2) with drift
##
## Coefficients:
##           ar1      ar2      ma1     ma2   drift
##       -0.0399  -0.6266  -1.1314  0.7224  0.0057
## s.e.   0.0822   0.0823   0.0903  0.0583  0.0010
##
## sigma^2 estimated as 0.001157:  log likelihood=254
## AIC=-495.99   AICc=-495.3   BIC=-478.83
```

So, the same model is selected for both AIC and BIC. We are going to proceed with this model.

Step 7: Forecasting and evaluating the model

```
plot(forecast(model_fit,h=period),type='l')
lines(test_data$time, test_data$log_value )
```



As can be seen from the graph above, the blue line is the forecasted value of the log of the consumption and the black line is the log of the actual value of consumption. The model seems to capture the trend well. However, we have to take the exponential of the forecasted value to get the actual forecast.

We do that conversion and print the mse as follows:

```
forecasted_values = forecast(model_fit,h = period)
forecasted_values = as.numeric(forecasted_values$mean)

forecasted_values = lapply(forecasted_values,exp)
forecasted_values = as.numeric(forecasted_values)

true_values = data$value[(m+1):(m+period)]

mse = mean((true_values-forecasted_values)^2)

mse
```

```
## [1] 13175977
```

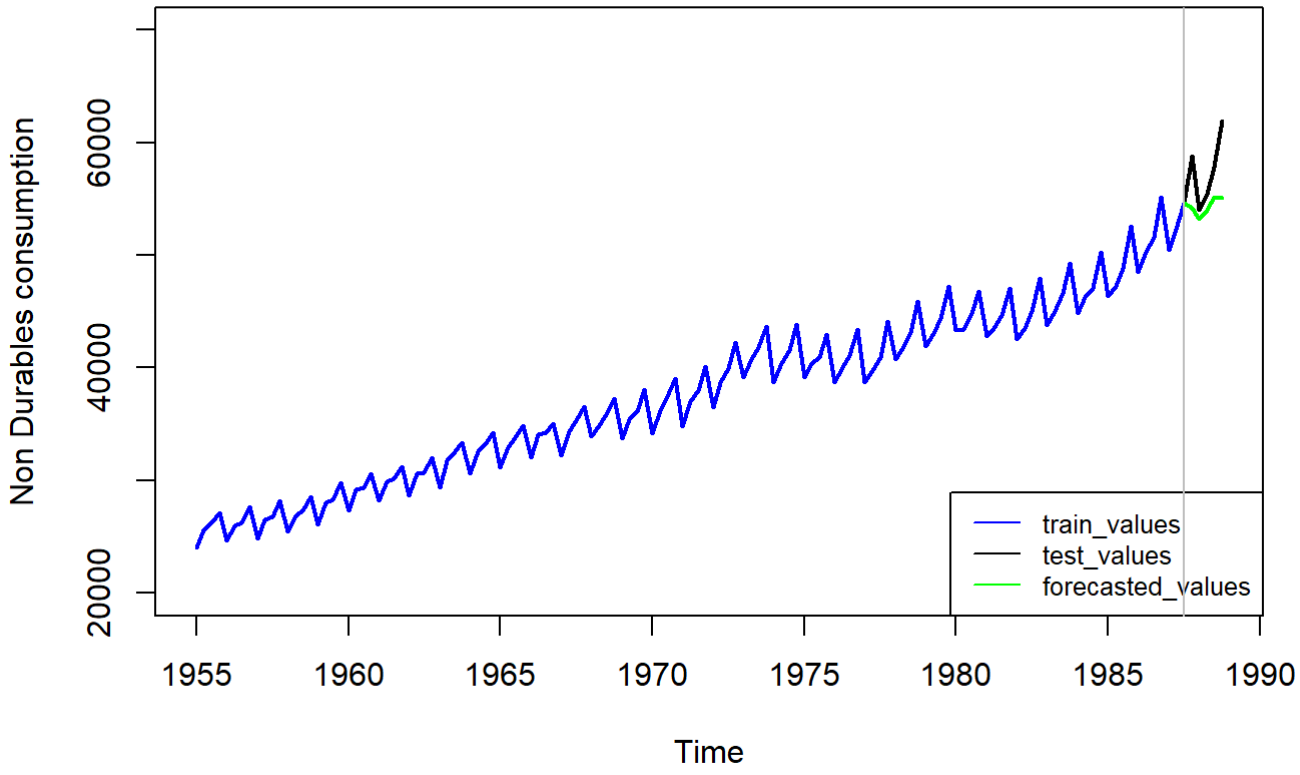
The value looks quite big. But as we saw from the descriptive stats of the data, the training data itself spans over a wide range and has a variance of 56761482 as shown below:

```
descriptive_stats$`Standard Deviation`^2
```

```
## [1] 56761482
```

So, this may be a reason for such huge mse value as the model otherwise seems to capture the trend pretty well as can be shown in the graph below:

```
plot(NULL,xlim = c(tm[1],tm[n]),ylim = c(20000,70000),ylab = 'Non Durables consumption',xlab=
'Time')
points(data$time, data$value,type='l',lwd=2,col='blue')
points(test_data$time, test_data$value,type='l',lwd=2,col='black')
points(test_data$time, forecasted_values,type='l',lwd=2,col='green')
legend("bottomright",
       legend =  c("train_values","test_values","forecasted_values"),
       col = c("blue","black","green"),lty = 1,
       cex = 0.8)

abline(v=tm[m+1],col='grey')
```



The comparison of true value and forecasted value is shown below:

```
data.frame("True_value"=true_values, "Forecasted_values" = forecasted_values)
```

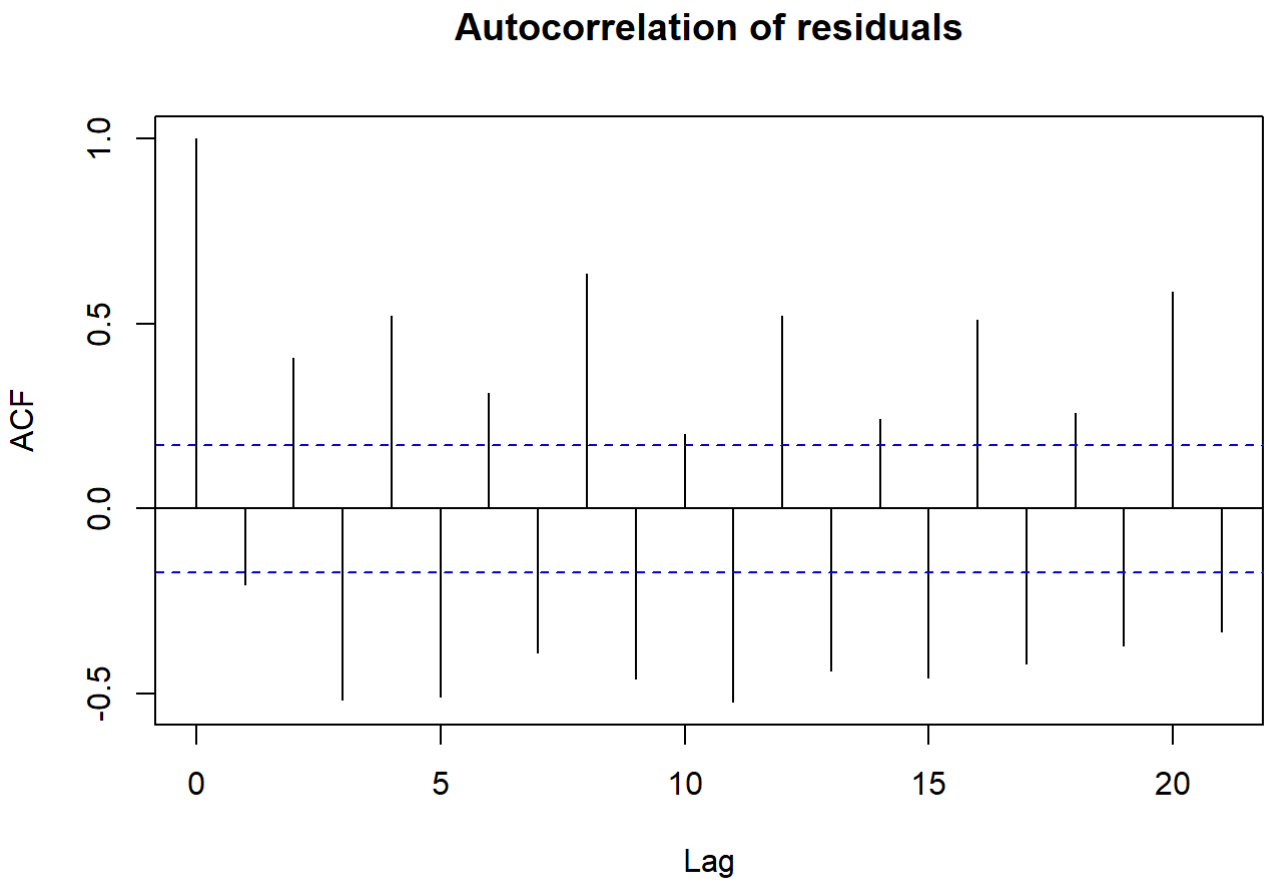
True_value <dbl>	Forecasted_values <dbl>
54633	54663.91
58802	54156.47
53990	53196.82
55477	54058.95
57850	55155.39
61978	55080.68

6 rows

The model seems to be pretty close to the true values initially but fails to capture the increased spike in the last two quarters leading to a high mse.

##Model Diagnostics 1) Checking autocorrelation between residuals by plotting the autocorrelation plot of the residuals.

```
acf(ts(model_fit$residuals),main = "Autocorrelation of residuals")
```



From the ACF plot of residuals, it is obvious that the residuals are correlated with each other. Hence, the major of assumption of independece of white noise if violated. We also perform the statistical tests: Box Pierce and Ljung Box tests to confirm this as follows:

```
# Box-Pierce Test
Box.test(ts(model_fit$residuals),lag = 10, type = "Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  ts(model_fit$residuals)
## X-squared = 248.74, df = 10, p-value < 2.2e-16
```

```
Box.test(ts(model_fit$residuals),lag = 10, type = "Ljung-Box")
```

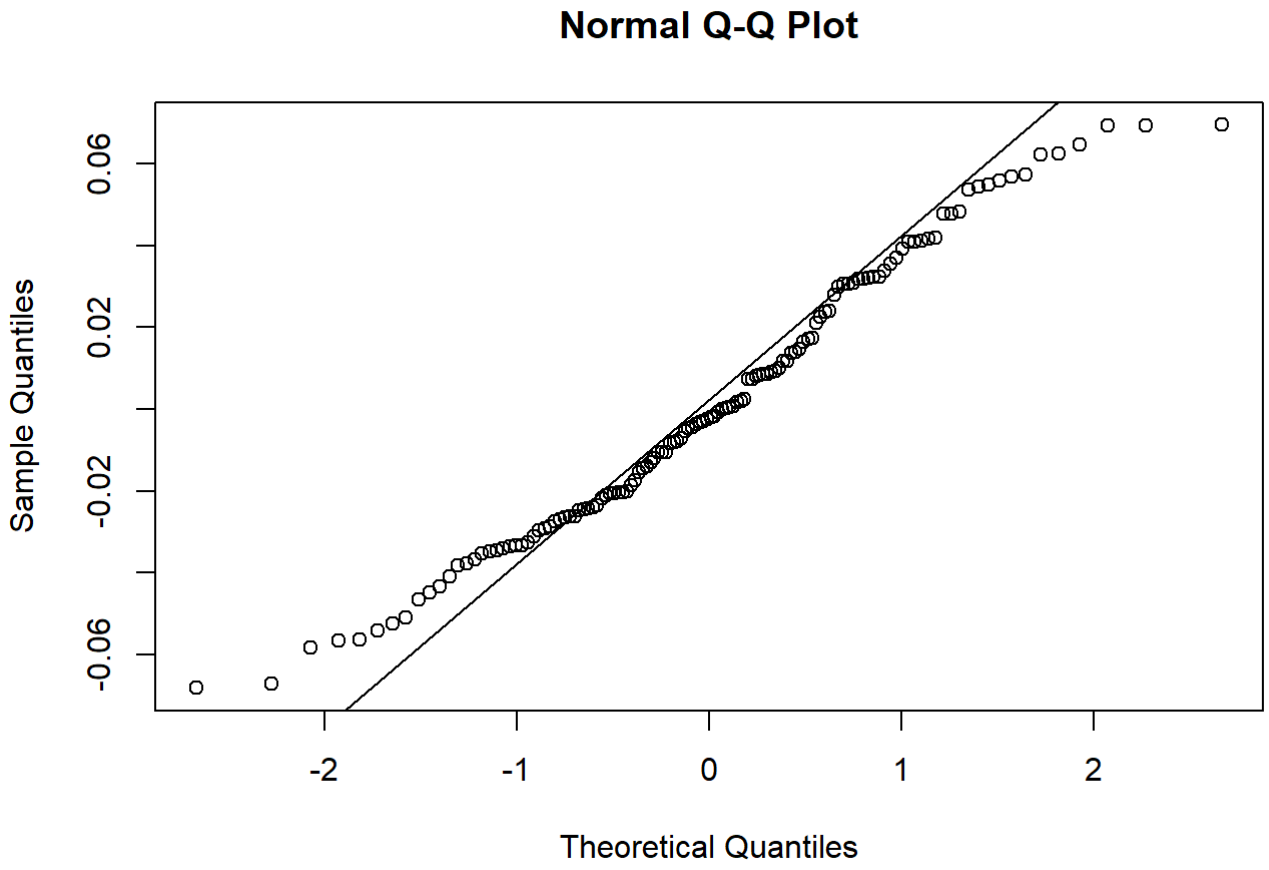
```
##
##  Box-Ljung test
##
## data:  ts(model_fit$residuals)
## X-squared = 264.09, df = 10, p-value < 2.2e-16
```

```
# Ljung-Box Test
```

So, for both the Ljung-Box and Box-Pierce tests, the null hypothesis of independence is rejected. Hence, the residuals are correlated and some significant lags are not being considered in the model. This may further indicate that the model should consider seasonality as the data is quarterly data.

2. Checking Normality of residuals

```
qqnorm(model_fit$residuals)
qqline(model_fit$residuals)
```



It is hard to say from the normal qq plot of the residuals whether the assumption of white noise being sampled from a gaussian distribution is valid or not. We test the normality of the residuals with the Shapiro-Wilk test, Kolmogorov-Smirnov test and the Jarque Bera test.

```
shapiro.test(model_fit$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_fit$residuals
## W = 0.97877, p-value = 0.03919
```

```
ks.test(model_fit$residuals,'pnorm',mean =0, sd = sqrt(model_fit$sigma2))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  model_fit$residuals
## D = 0.065197, p-value = 0.6384
## alternative hypothesis: two-sided
```

```
jarque.bera.test(model_fit$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  model_fit$residuals
## X-squared = 3.8761, df = 2, p-value = 0.144
```

We see from above that for a 5% level of significance, the null hypothesis of normality is rejected for Shapiro Wilk Test but for the Kolmogorov-Smirnov and Jarque Bera Tests, we cannot reject the null hypothesis of Normality. Hence, by vote of majority we are going to say that our assumption of normality of error terms cannot be rejected.

Conclusion We saw from the model diagnostics that a major issue was there was correlation in the residuals which indicate vital lags are not being considered in the model building. This could be due to seasonality in the data or Non-Seasonal ARIMA not being the correct model for our purpose. We will explore considering seasonality and other models in the next parts.