

Data624 - Project1

Amit Kapoor

3/28/2021

Contents

Overview	1
Part A - ATM Forecast	1
Exploratory Analysis	1
Data Cleaning	5
Time Series	6
Forecast May, 2010	29
Part B - Forecasting Power	29
Exploratory Analysis	30
Data Cleaning	30
Time Series	31
Forecast 2014	41
Part C - Waterflow Pipe	42
Exploratory Analysis	42
Data Cleaning	45
Time Series	46
Forecast a week forward	57

Overview

This project includes 3 time series dataset and requires to select best forecasting model for all 3 datasets.

- Part A - ATM Forecast
- Part B - Forecasting Power
- Part C - Waterflow Pipe

Part A - ATM Forecast

The dataset contains cash withdrawals from 4 different ATM machines from May 2009 to Apr 2010. The variable 'Cash' is provided in hundreds of dollars and data is in a single file. Before starting our analysis we will first download the excel from github and then read it through read_excel.

Exploratory Analysis

```
temp.file <- tempfile(fileext = ".xlsx")
download.file(url="https://github.com/amit-kapoor/data624/blob/main/Project1/ATM624Data.xlsx?raw=true",
             destfile = temp.file,
```

```

mode = "wb",
quiet = TRUE)
atm.data <- read_excel(temp.file, skip=0, col_types = c("date","text","numeric"))

glimpse(atm.data)

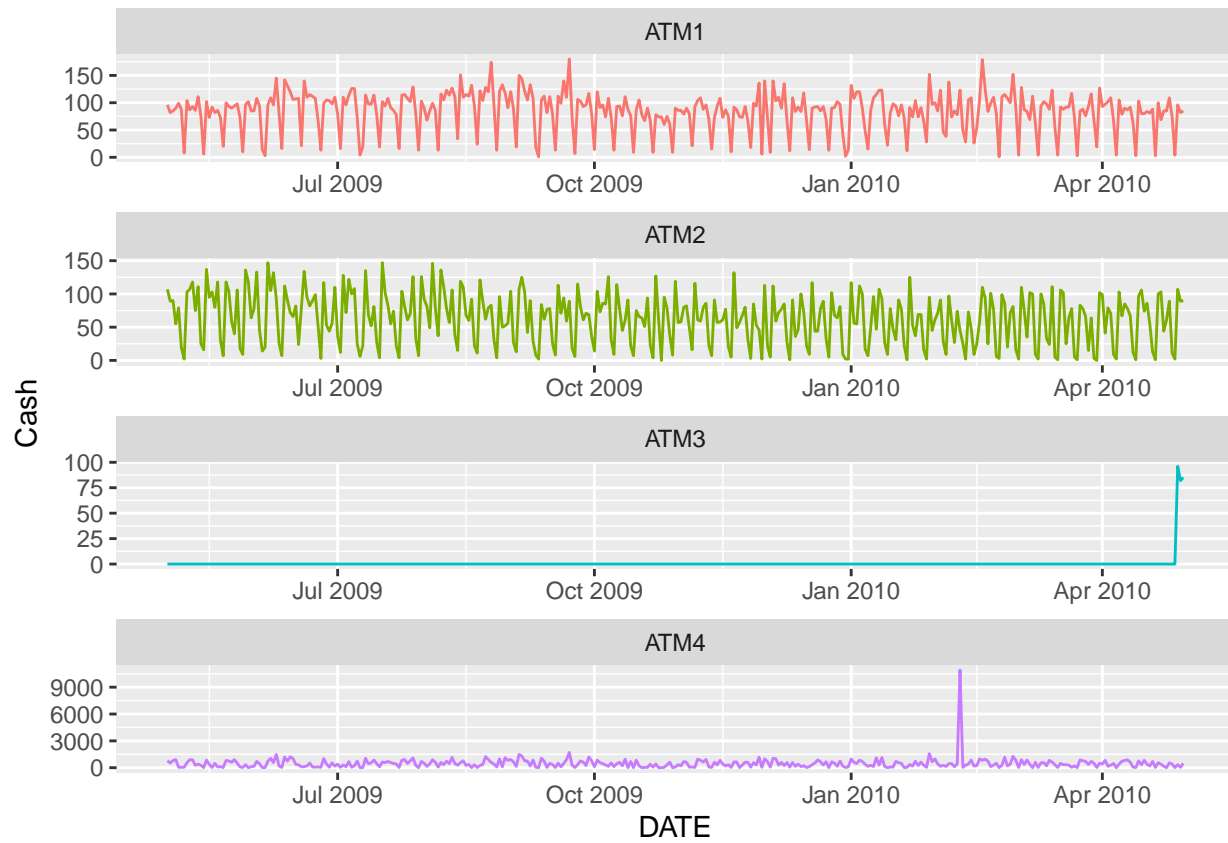
## Rows: 1,474
## Columns: 3
## $ DATE <dtm> 2009-05-01, 2009-05-01, 2009-05-02, 2009-05-02, 2009-05-03, 2009-~
## $ ATM <chr> "ATM1", "ATM2", "ATM1", "ATM2", "ATM1", "ATM2", "ATM1", "ATM2", "~
## $ Cash <dbl> 96, 107, 82, 89, 85, 90, 90, 55, 99, 79, 88, 19, 8, 2, 104, 103, ~

# rows missing values
atm.data[!complete.cases(atm.data),]

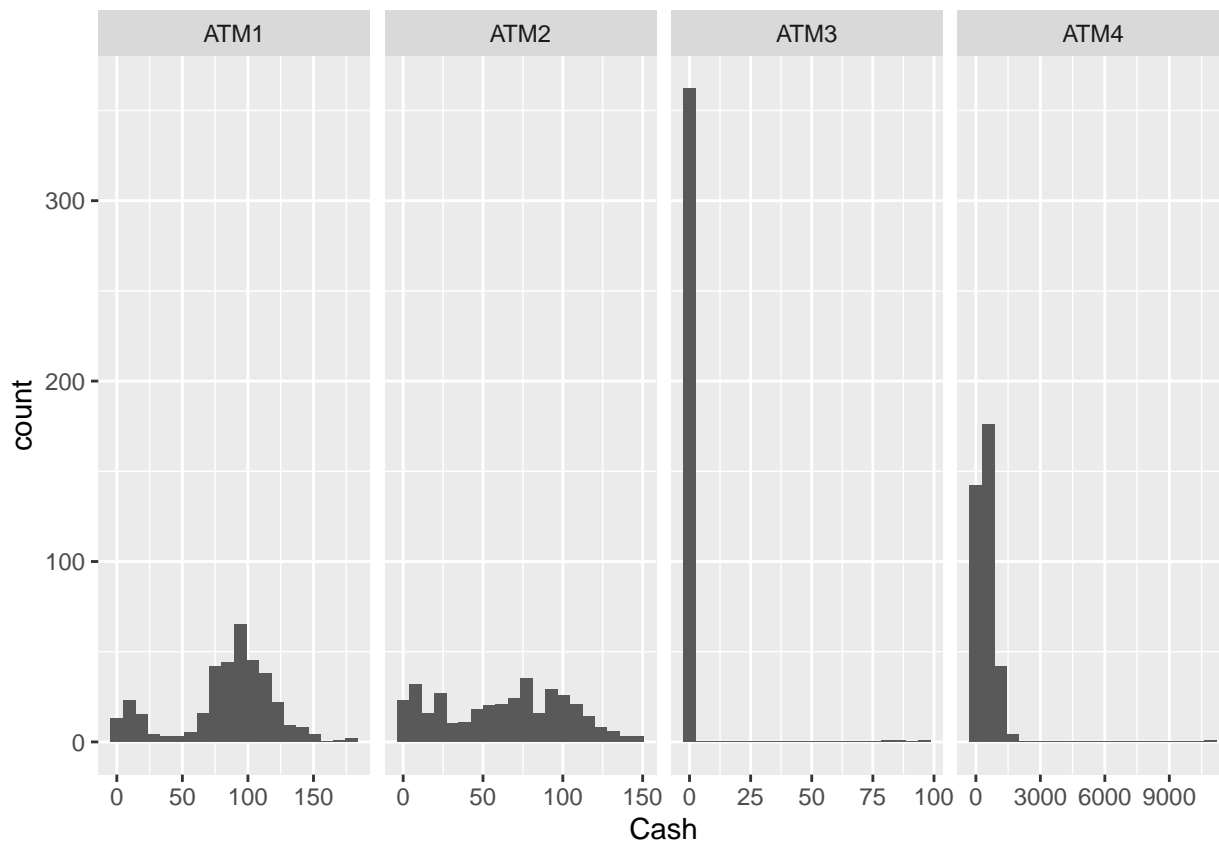
## # A tibble: 19 x 3
##   DATE                ATM    Cash
##   <dtm>              <chr> <dbl>
## 1 2009-06-13 00:00:00 ATM1     NA
## 2 2009-06-16 00:00:00 ATM1     NA
## 3 2009-06-18 00:00:00 ATM2     NA
## 4 2009-06-22 00:00:00 ATM1     NA
## 5 2009-06-24 00:00:00 ATM2     NA
## 6 2010-05-01 00:00:00 <NA>    NA
## 7 2010-05-02 00:00:00 <NA>    NA
## 8 2010-05-03 00:00:00 <NA>    NA
## 9 2010-05-04 00:00:00 <NA>    NA
## 10 2010-05-05 00:00:00 <NA>    NA
## 11 2010-05-06 00:00:00 <NA>    NA
## 12 2010-05-07 00:00:00 <NA>    NA
## 13 2010-05-08 00:00:00 <NA>    NA
## 14 2010-05-09 00:00:00 <NA>    NA
## 15 2010-05-10 00:00:00 <NA>    NA
## 16 2010-05-11 00:00:00 <NA>    NA
## 17 2010-05-12 00:00:00 <NA>    NA
## 18 2010-05-13 00:00:00 <NA>    NA
## 19 2010-05-14 00:00:00 <NA>    NA

ggplot(atm.data[complete.cases(atm.data),] , aes(x=DATE, y=Cash, col=ATM )) +
  geom_line(show.legend = FALSE) +
  facet_wrap(~ATM, ncol=1, scales = "free")

```



```
ggplot(atm.data[complete.cases(atm.data),] , aes(x=Cash )) +
  geom_histogram(bins=20) +
  facet_grid(cols=vars(ATM), scales = "free")
```



```
# consider complete cases
atm.comp <- atm.data[complete.cases(atm.data),]
# pivot wider with cols from 4 ATMs and their values as Cash
atm.comp <- atm.comp %>% pivot_wider(names_from = ATM, values_from = Cash)
head(atm.comp)
```

```
## # A tibble: 6 x 5
##   DATE                ATM1  ATM2  ATM3  ATM4
##   <dtm>              <dbl> <dbl> <dbl> <dbl>
## 1 2009-05-01 00:00:00    96   107    0 777.
## 2 2009-05-02 00:00:00    82    89    0 524.
## 3 2009-05-03 00:00:00    85    90    0 793.
## 4 2009-05-04 00:00:00    90    55    0 908.
## 5 2009-05-05 00:00:00    99    79    0 52.8
## 6 2009-05-06 00:00:00    88    19    0 52.2
```

```
# summary
atm.comp %>% select(-DATE) %>% summary()
```

```
##           ATM1           ATM2           ATM3           ATM4
##  Min.   : 1.00   Min.   : 0.00   Min.   : 0.0000   Min.   : 1.563
## 1st Qu.: 73.00   1st Qu.: 25.50   1st Qu.: 0.0000   1st Qu.: 124.334
## Median : 91.00   Median : 67.00   Median : 0.0000   Median : 403.839
## Mean   : 83.89   Mean   : 62.58   Mean   : 0.7206   Mean   : 474.043
## 3rd Qu.:108.00   3rd Qu.: 93.00   3rd Qu.: 0.0000   3rd Qu.: 704.507
## Max.   :180.00   Max.   :147.00   Max.   :96.0000   Max.   :10919.762
## NA's   :3       NA's   :2
```

Per above exploratory analysis, all ATMs show different patterns. We would perform forecasting for each

ATM separately.

- ATM1 and ATM2 shows similar pattern (approx.) throughout the time. ATM1 and ATM2 have 3 and 2 missing entries respectively.
- ATM3 appears to become online in last 3 days only and rest of days appears inactive. So the data available for this ATM is very limited.
- ATM4 requires replacement for outlier and we can assume that one day spike of cash withdrawal is unique. It has an outlier showing withdrawal amount 10920.

Data Cleaning

For this part we will first apply `ts()` function to get required time series. Next step is to apply `tsclean` function that will handle missing data along with outliers. To estimate missing values and outlier replacements, this function uses linear interpolation on the (possibly seasonally adjusted) series. Once we get the clean data we will use `pivot_longer` to get the dataframe in its original form.

```
atm.ts <- ts(atm.comp %>% select(-DATE))
head(atm.ts)
```

```
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
##      ATM1 ATM2 ATM3      ATM4
## 1    96  107    0 776.99342
## 2    82   89    0 524.41796
## 3    85   90    0 792.81136
## 4    90   55    0 908.23846
## 5    99   79    0 52.83210
## 6    88   19    0 52.20845
```

```
# apply tsclean
atm.ts.cln <- sapply(X=atm.ts, tsclean)
atm.ts.cln %>% summary()
```

```
##      ATM1      ATM2      ATM3      ATM4
## Min.   : 1.00   Min.   : 0.00   Min.   : 0.0000   Min.   : 1.563
## 1st Qu.: 73.00  1st Qu.: 26.00  1st Qu.: 0.0000  1st Qu.: 124.334
## Median : 91.00  Median : 67.00  Median : 0.0000  Median : 402.770
## Mean   : 84.15  Mean   : 62.59  Mean   : 0.7206  Mean   : 444.757
## 3rd Qu.:108.00  3rd Qu.: 93.00  3rd Qu.: 0.0000  3rd Qu.: 704.192
## Max.   :180.00  Max.   :147.00  Max.   :96.0000  Max.   :1712.075
```

If we compare this summary with previous one of original data, ATM1 and ATM2 has no more NAs and ATM4 outlier value (10919.762) is handled and now the max value is 1712.075.

```
# convert into data frame, pivot longer , arrange by ATM and bind with dates
atm.new <- as.data.frame(atm.ts.cln) %>%
  pivot_longer(everything(), names_to = "ATM", values_to = "Cash") %>%
  arrange(ATM)

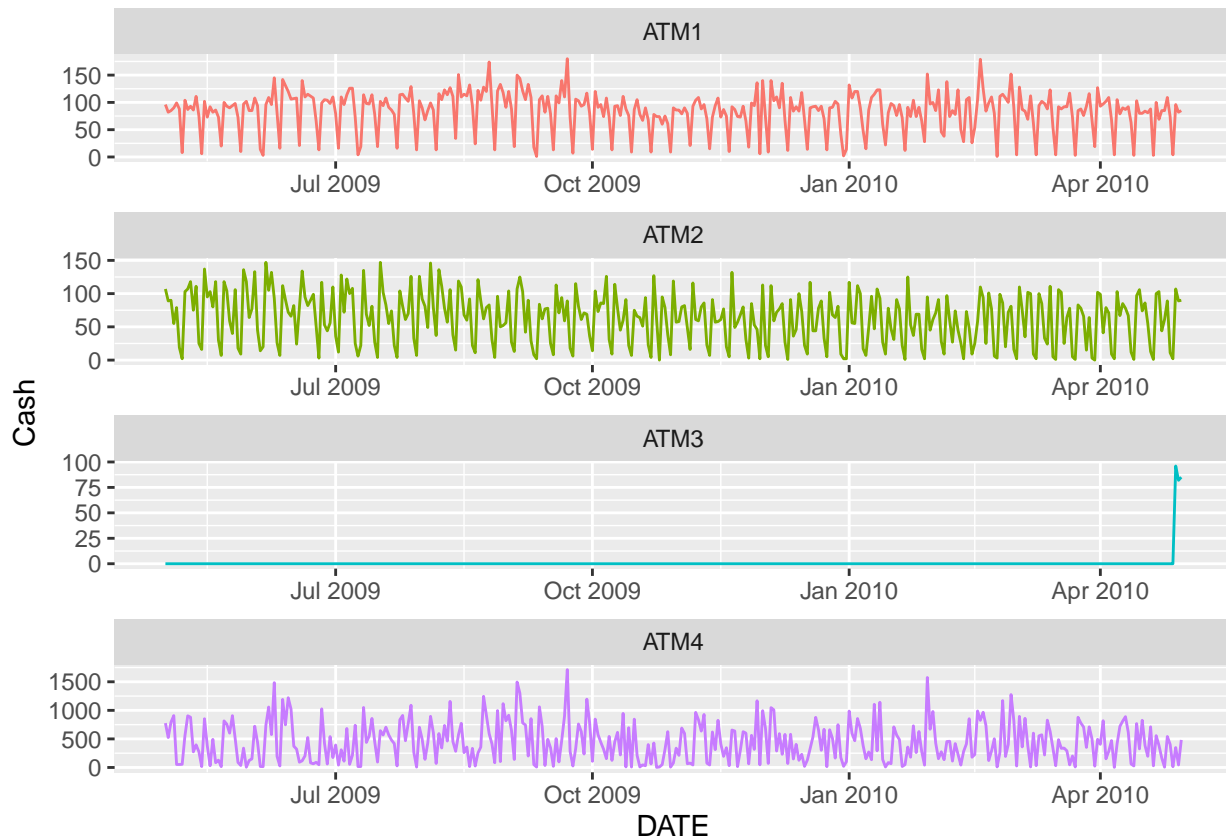
atm.new <- cbind(
  DATE = seq(as.Date("2009-05-1"), as.Date("2010-04-30"), length.out=365),
  atm.new)

head(atm.new)
```

```
##      DATE  ATM Cash
## 1 2009-05-01 ATM1   96
```

```
## 2 2009-05-02 ATM1    82
## 3 2009-05-03 ATM1    85
## 4 2009-05-04 ATM1    90
## 5 2009-05-05 ATM1    99
## 6 2009-05-06 ATM1    88
```

```
ggplot(atm.new , aes(x=DATE, y=Cash, col=ATM )) +
  geom_line(show.legend = FALSE) +
  facet_wrap(~ATM, ncol=1, scales = "free")
```



Though above plot doesn't show much differences for ATM1,2,3 but tsclean handled the ATM4 data very well after replacing the outlier.

Time Series

Function to plot forecast for various models.

```
# function to plot forecast(s)
atm.forecast <- function(timeseries) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)
  # models for forecast
  hw.model <- timeseries %>% hw(h=31, seasonal = "additive", lambda = lambda, damped = TRUE)
  ets.model <- timeseries %>% ets(lambda = lambda)
  arima.model <- timeseries %>% auto.arima(lambda = lambda)
  # forecast
  atm.hw.fcst <- forecast(hw.model, h=31)
  atm.ets.fcst <- forecast(ets.model, h=31)
  atm.arima.fcst <- forecast(arima.model, h=31)
```

```

# plot forecasts
p1 <- autoplot(timeseries) +
  autolayer(atm.hw.fcst, PI=FALSE, series="Holt-Winters") +
  autolayer(atm.ets.fcst, PI=FALSE, series="ETS") +
  autolayer(atm.arima.fcst, PI=FALSE, series="ARIMA") +
  theme(legend.position = "top") +
  ylab("Cash Withdrawl")
# zoom in plot
p2 <- p1 +
  labs(title = "Zoom in ") +
  xlim(c(51,56))

grid.arrange(p1,p2,ncol=1)
}

```

Function to calculate RMSEs for various models.

```

model_accuracy <- function(timeseries, atm_num) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)

  # split the data to train and test
  train <- window(timeseries, end=c(40, 3))
  test <- window(timeseries, start=c(40, 4))

  # models for forecast
  hw.model <- train %>% hw(h=length(train), seasonal = "additive", lambda = lambda, damped = TRUE)
  ets.model <- train %>% ets(model='ANA', lambda = lambda)

  # Arima model
  if (atm_num == 1) {
    # for ATM1
    arima.model <- train %>% Arima(order=c(0,0,2),
                                   seasonal = c(0,1,1),
                                   lambda = lambda)
  } else if (atm_num == 2) {
    # for ATM2
    arima.model <- train %>% Arima(order=c(3,0,3),
                                   seasonal = c(0,1,1),
                                   include.drift = TRUE,
                                   lambda = lambda,
                                   biasadj = TRUE)
  } else {
    # for ATM4
    arima.model <- train %>% Arima(order=c(0,0,1),
                                   seasonal = c(2,0,0),
                                   lambda = lambda)
  }

  # forecast
  hw.frct = forecast(hw.model, h = length(test))$mean
  ets.frct = forecast(ets.model, h = length(test))$mean
  arima.frct = forecast(arima.model, h = length(test))$mean
}

```

```

# dataframe having rmse
rmse = data.frame(RMSE=cbind(accuracy(hw.frct, test)[,2],
                             accuracy(ets.frct, test)[,2],
                             accuracy(arima.frct, test)[,2]))

names(rmse) = c("Holt-Winters", "ETS", "ARIMA")
# display rmse
rmse
}

```

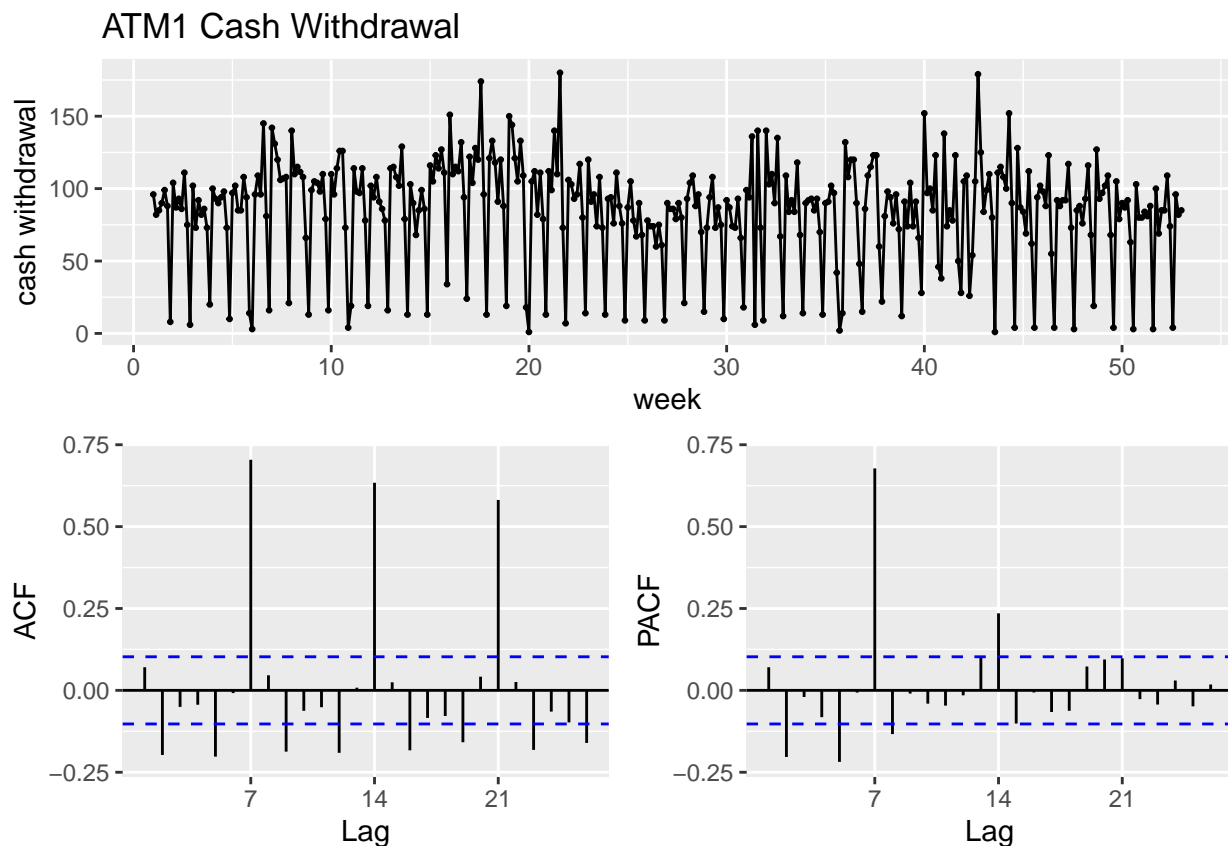
ATM1

Seeing the time series plot, it is clear that there is a seasonality in the data. We can see increasing and decreasing activities over the weeks in below plot. From the ACF plot, we can see a slight decrease in every 7th lag due to trend. PACF plot shows some significant lags at the beginning.

```

atm1.ts <- atm.new %>% filter(ATM=="ATM1") %>% select(Cash) %>% ts(frequency = 7)
ggtsdisplay(atm1.ts, main="ATM1 Cash Withdrawal", ylab="cash withdrawal", xlab="week")

```



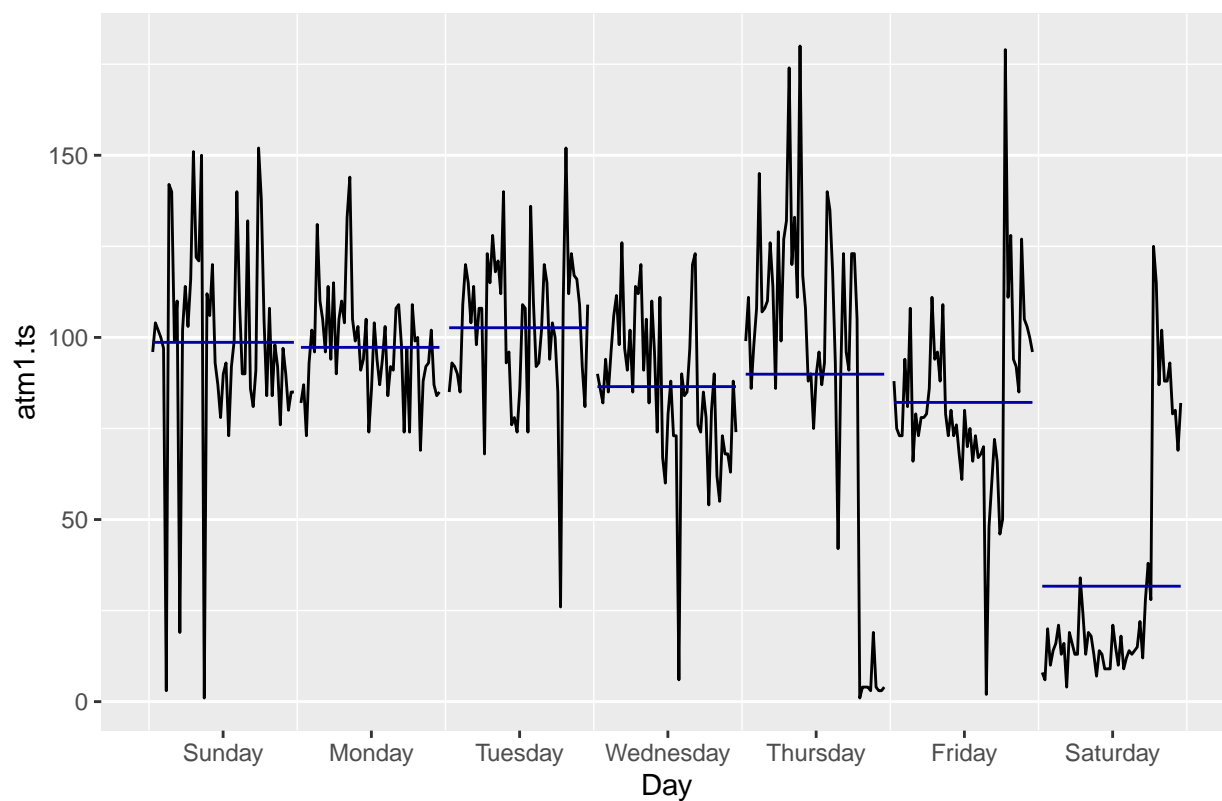
From the above plots it is evident that the time series is non stationary, showing seasonality and will require differencing to make it stationary.

```

ggsubseriesplot(atm1.ts, main="ATM1 Cash Withdrawal")

```


ATM1 Cash Withdrawal

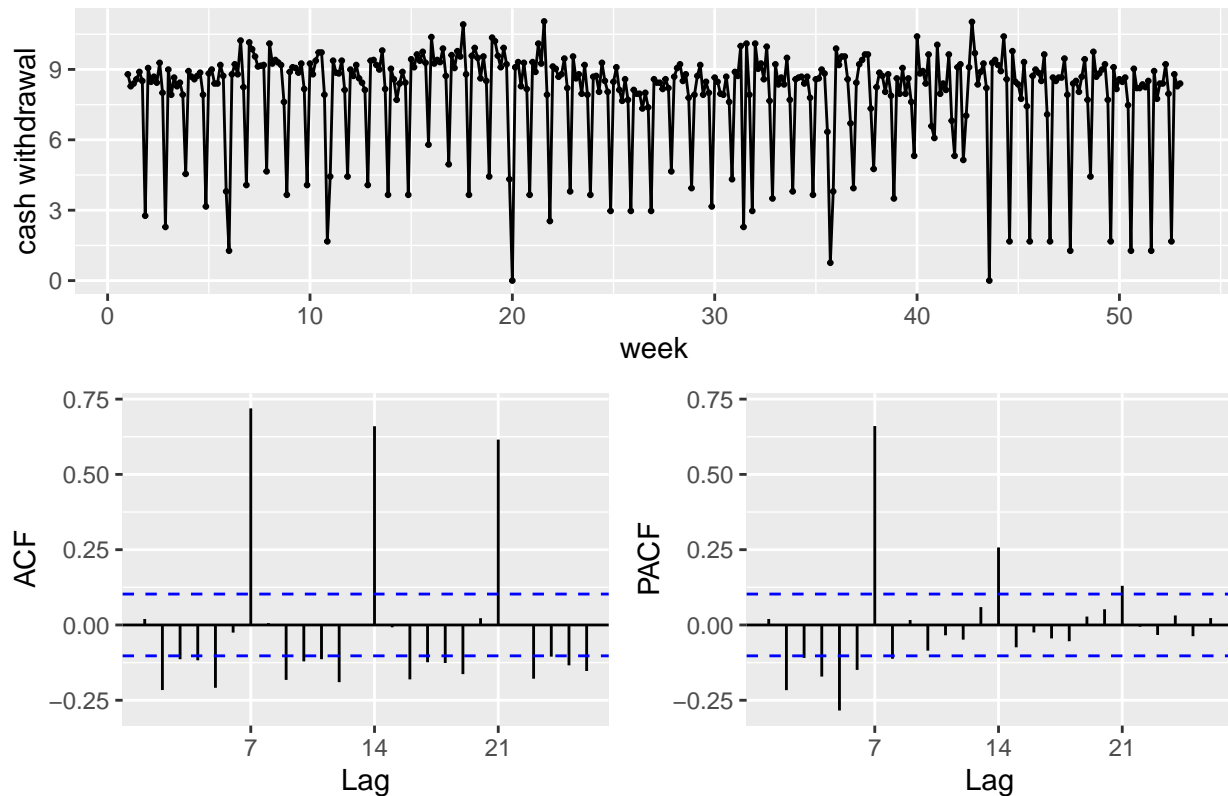


From the subseries plot, it is apparent that Tuesdays having highest mean of cash withdrawal while Saturdays being the lowest.

Next step is to apply BoxCox transformation. With λ being 0.26, the resulting transformation does handle the variability in time series as shown in below transformed plot.

```
atm1.lambda <- BoxCox.lambda(atm1.ts)
atm1.ts.bc <- BoxCox(atm1.ts, atm1.lambda )
ggtsdisplay(atm1.ts.bc, main=paste("ATM1 Cash Withdrawal",round(atm1.lambda, 3)), ylab="cash withdrawal")
```

ATM1 Cash Withdrawal 0.262



Next we will see the number of differences required for a stationary series and the number of differences required for a seasonally stationary series.

```
# Number of differences required for a stationary series
ndiffs(atm1.ts.bc)
```

```
## [1] 0
```

```
# Number of differences required for a seasonally stationary series
nsdiffs(atm1.ts.bc)
```

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

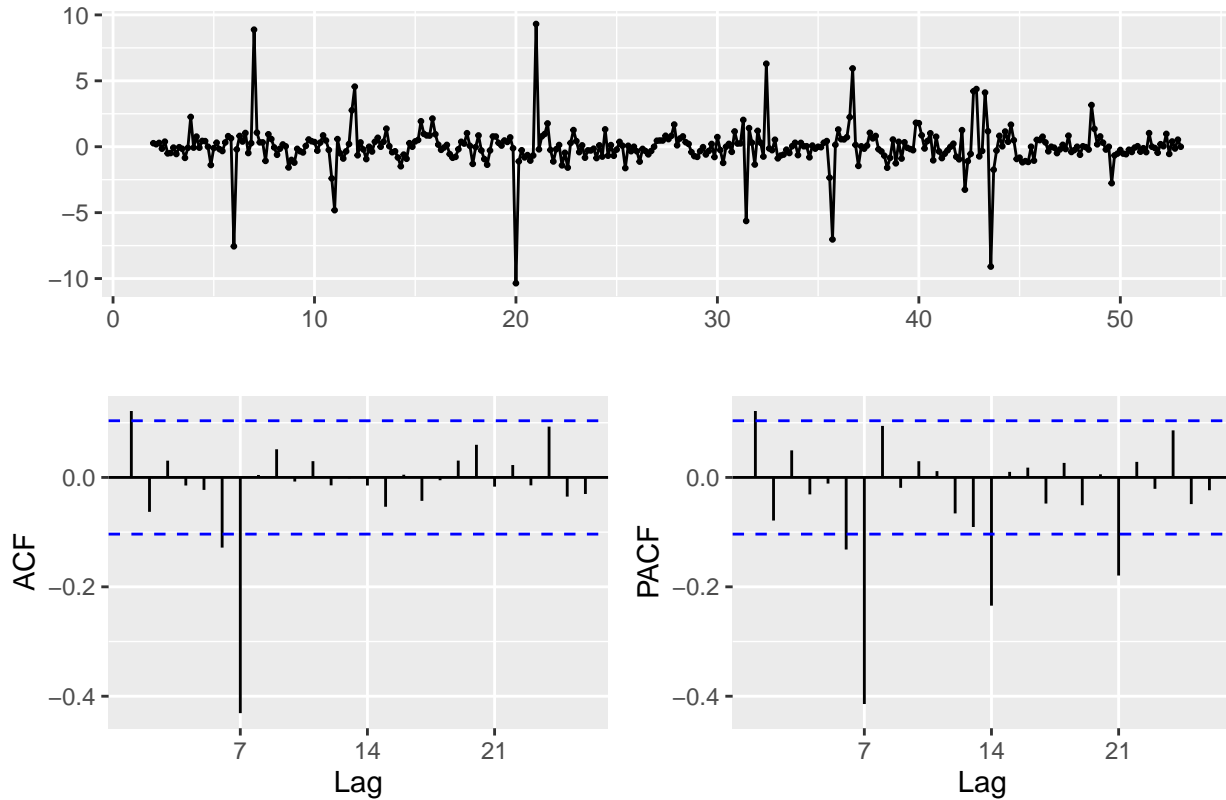
It shows number of differences required for a seasonality stationary series is 1. Next step is to check kpss summary.

```
atm1.ts.bc %>% diff(lag=7) %>% ur.kpss() %>% summary()
```

```
##
## #####
## # KPSS Unit Root Test #
## #####
##
## Test is of type: mu with 5 lags.
##
## Value of test-statistic is: 0.0153
##
## Critical value for a significance level of:
##          10pct  5pct  2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary.

```
atm1.ts.bc %>% diff(lag=7) %>% ggtsdisplay()
```



The data is non-stationary with seasonality so there will be a seasonal difference of 1. Finally, the differencing of the data has now made it stationary. From the ACF plot, it is apparent now that there is a significant spike at lag 7 but none beyond lag 7.

Lets start with Holt-Winter's additive model with damped trend since the seasonal variations are roughly constant through out the series.

```
# Holt Winters with damped True
atm1.ts %>% hw(h=31, seasonal = "additive", lambda = atm1.lambda, damped = TRUE)
```

##	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## 53.14286	86.726308	48.2873323	144.09156	34.0075240	183.86219
## 53.28571	99.656005	56.7502143	162.78119	40.5780934	206.17461
## 53.42857	74.268913	40.2785592	125.84028	27.8645027	161.94499
## 53.57143	3.946722	0.9101988	11.36403	0.3067520	18.00566
## 53.71429	99.554782	56.6834213	162.63577	40.5259535	206.00148
## 53.85714	78.851329	43.2063605	132.58498	30.1007501	170.06058
## 54.00000	85.114307	47.2424187	141.74438	33.2015587	181.05113
## 54.14286	86.658670	45.6127105	150.10813	30.9111908	195.01621
## 54.28571	99.582554	53.7351794	169.36386	37.0454815	218.30796
## 54.42857	74.210981	37.9429783	131.29091	25.1978202	172.11308
## 54.57143	3.940224	0.7732156	12.30036	0.2189239	20.04980
## 54.71429	99.485702	53.6737241	169.22060	36.9987686	218.13522
## 54.85714	78.794338	40.7477446	138.25412	27.2771480	180.60622
## 55.00000	85.055212	44.6156330	147.70043	30.1637147	192.09415

```
## 55.14286      86.599982 43.2340302 155.85781 28.2323452 205.80698
## 55.28571      99.518822 51.0490613 175.64880 33.9793617 230.03192
## 55.42857      74.160715 35.8698562 136.50504 22.8992462 181.96358
## 55.57143        3.934588  0.6604831  13.21942  0.1555673  22.09545
## 55.71429      99.425760 50.9921256 175.50745 33.9371713 229.85958
## 55.85714      78.744887 38.5634686 143.67495 24.8394878 190.81691
## 56.00000      85.003935 42.2795812 153.39265 27.5361909 202.77913
## 56.14286      86.549058 41.0923732 161.39633 25.8838357 216.31745
## 56.28571      99.463519 48.6265521 181.69785 31.2829118 241.43875
## 56.42857      74.117099 34.0067798 141.53228 20.8914243 191.57013
## 56.57143        3.929701  0.5664429  14.12646  0.1096150  24.14933
## 56.71429      99.373745 48.5734860 181.55814 31.2445441 241.26666
## 56.85714      78.701976 36.5988213 148.89936 22.7069013 200.76969
## 57.00000      84.959439 40.1763734 158.87600 25.2333014 213.18774
## 57.14286      86.504867 39.1457044 166.76293 23.8038208 226.60572
## 57.28571      99.415528 46.4210490 187.55457 28.8873888 252.59311
## 57.42857      74.079250 32.3163685 146.40758 19.1195026 200.98424
```

Next is to apply exponential smoothing method on this time series. It shows that the ETS(A, N, A) model best fits for the transformed ATM4, i.e. exponential smoothing with additive error, no trend component and additive seasonality.

```
atm1.ts %>% ets(lambda = atm1.lambda )
```

```
## ETS(A,N,A)
##
## Call:
## ets(y = ., lambda = atm1.lambda)
##
## Box-Cox transformation: lambda= 0.2616
##
## Smoothing parameters:
##   alpha = 1e-04
##   gamma = 0.3513
##
## Initial states:
##   l = 7.9717
##   s = -4.5094 0.5635 1.0854 0.5711 0.9551 0.5582
##       0.7761
##
## sigma: 1.343
##
##      AIC      AICc      BIC
## 2379.653 2380.275 2418.652
```

Next we will find out the appropriate ARIMA model for this time series. The suggested model seems ARIMA(0,0,2)(0,1,1)[7].

```
atm1.fit3 <- atm1.ts %>% auto.arima(lambda = atm1.lambda )
atm1.fit3
```

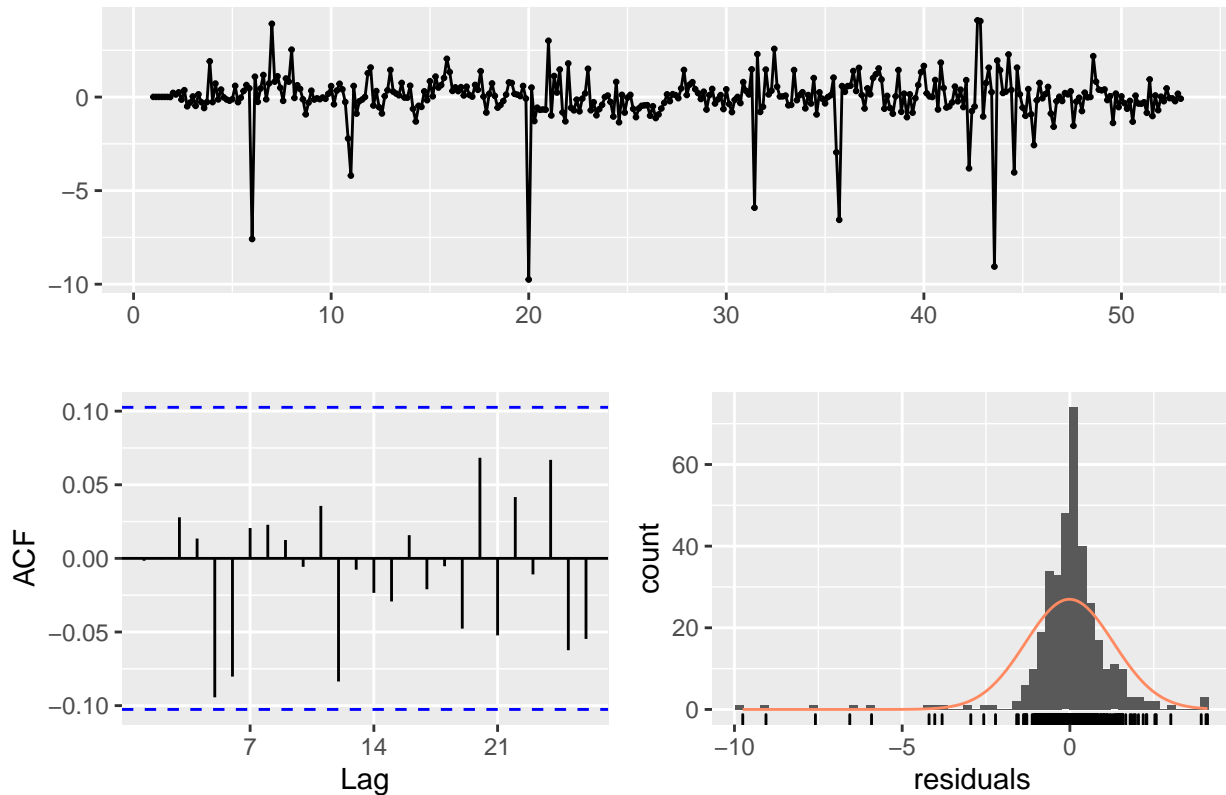
```
## Series: .
## ARIMA(0,0,2)(0,1,1)[7]
## Box Cox transformation: lambda= 0.2615708
##
## Coefficients:
##      ma1      ma2      sma1
```

```
##      0.1126  -0.1094  -0.6418
## s.e.  0.0524   0.0520   0.0432
##
## sigma^2 estimated as 1.764:  log likelihood=-609.99
## AIC=1227.98   AICc=1228.09   BIC=1243.5
```

Next is to see residuals time series plot which shows residuals are being near normal with mean of the residuals being near to zero. Also there is no significant autocorrelation that confirms that forecasts are good.

```
checkresiduals(atm1.fit3)
```

Residuals from ARIMA(0,0,2)(0,1,1)[7]

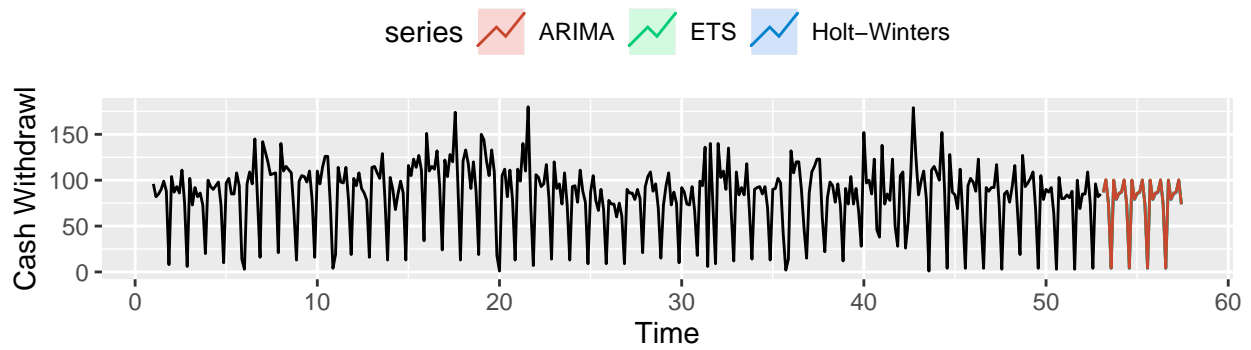


```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,2)(0,1,1)[7]
## Q* = 9.8626, df = 11, p-value = 0.5428
##
## Model df: 3.    Total lags used: 14
```

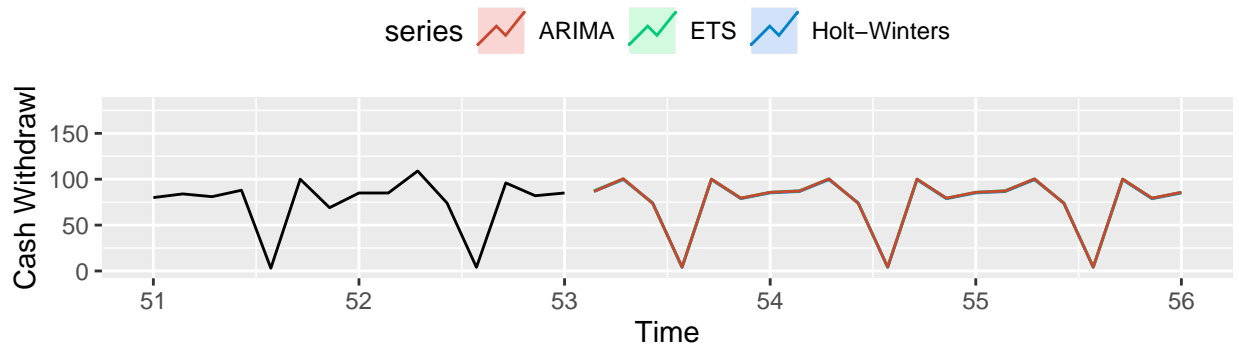
Let's plot the forecast for all the considered models above which will shows a nice visual comparison. it will also show a zoomed in plot to have a clearer view.

```
atm.forecast(atm1.ts)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



Zoom in



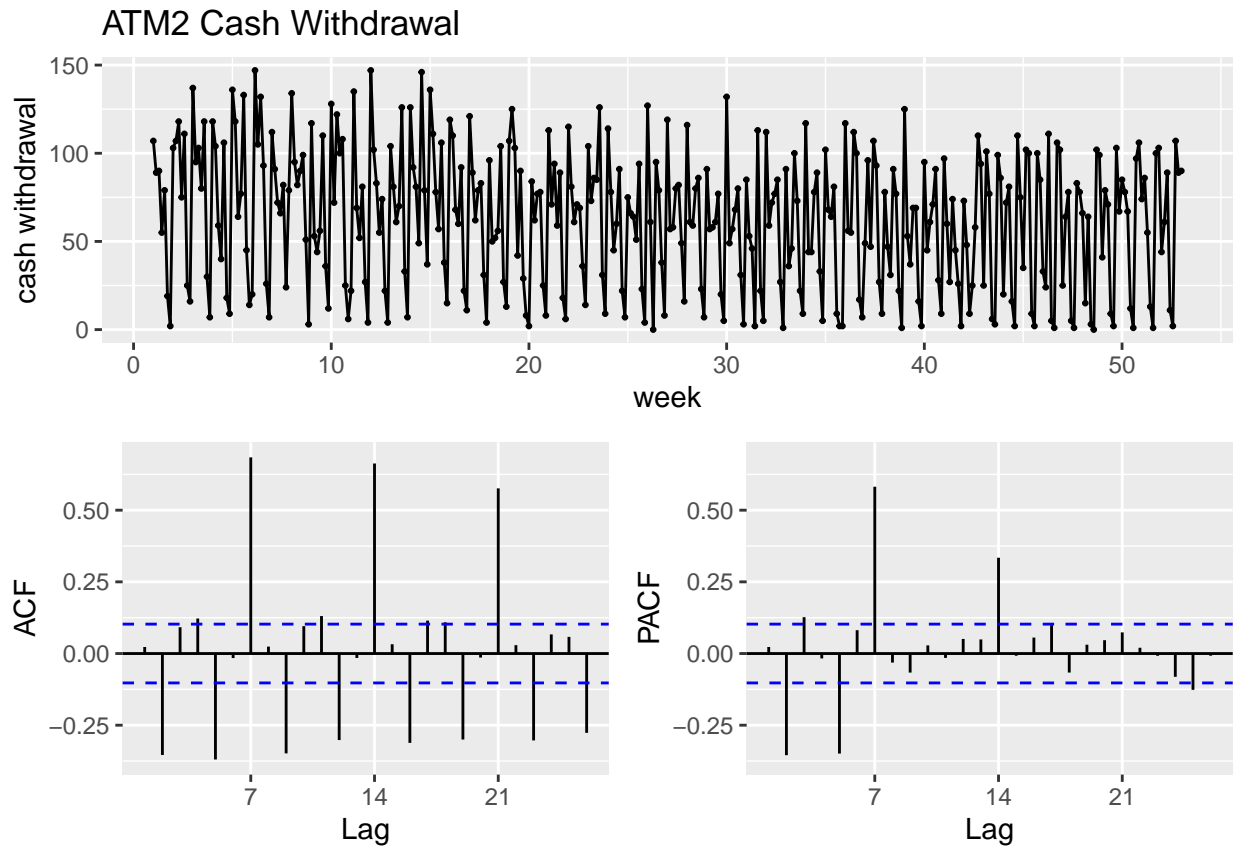
```
model_accuracy(atm1.ts,1)
```

```
## Holt-Winters      ETS      ARIMA
## 1      49.35115 49.22521 49.18074
```

ATM2

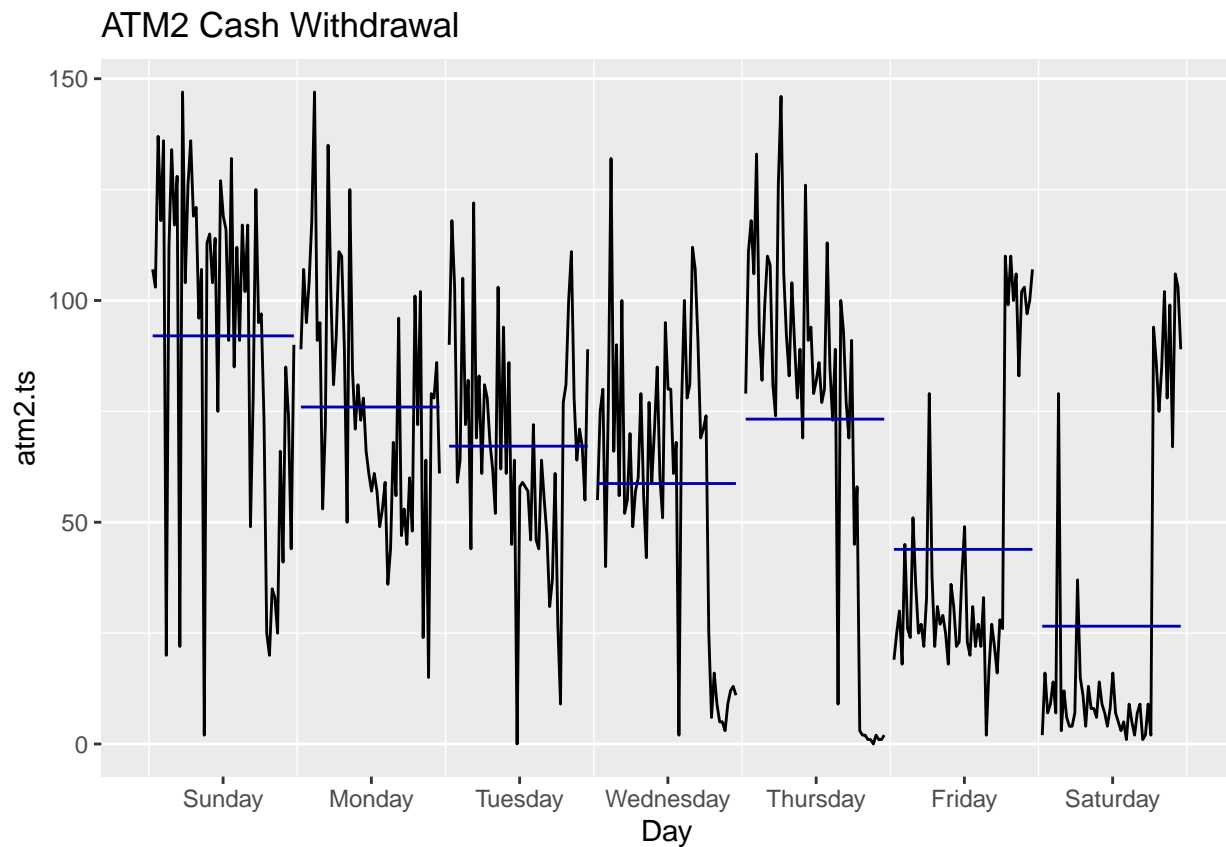
From the time series plot, it is apparent that there is a seasonality in the data but don't see a trend over the period. ACF shows significant lags at 7, 14, and 21 confirming seasonality. From the PACF, there are few significant lags at the beginning but others within critical limit. Overall, it is non-stationary, having seasonality and would require differencing for it to become stationary.

```
atm2.ts <- atm.new %>% filter(ATM=="ATM2") %>% select(Cash) %>% ts(frequency = 7)
ggtsdisplay(atm2.ts, main="ATM2 Cash Withdrawal", ylab="cash withdrawal", xlab="week")
```



From the subseries plot, it is clear that Sunday is having highest mean for cash withdrawal while Saturday has the lowest.

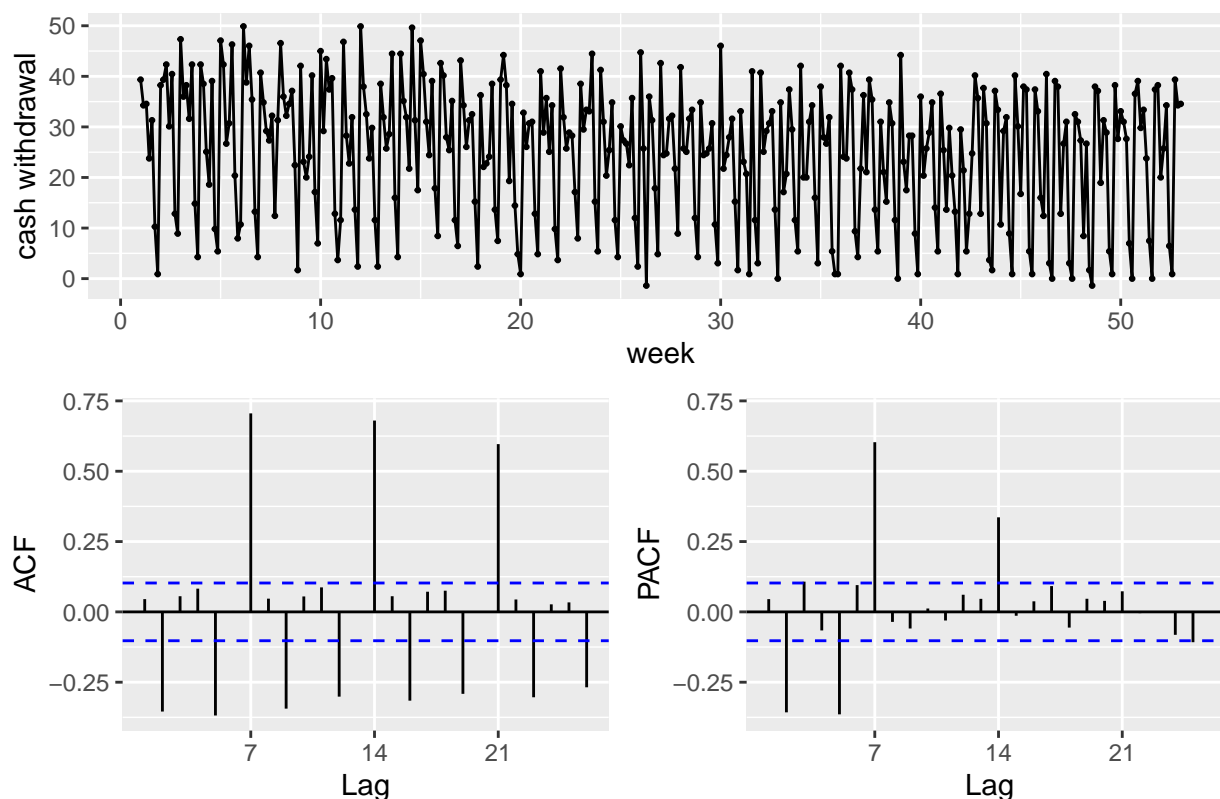
```
ggsubseriesplot(atm2.ts, main="ATM2 Cash Withdrawal")
```



Next step is to apply BoxCox transformation. With λ being 0.72, the resulting transformation does handle the variability in time series as shown in below transformed plot.

```
atm2.lambda <- BoxCox.lambda(atm2.ts)
atm2.ts.bc <- BoxCox(atm2.ts, atm2.lambda )
ggtsdisplay(atm2.ts.bc, main=paste("ATM2 Cash Withdrawal",round(atm2.lambda, 3)), ylab="cash withdrawal")
```


ATM2 Cash Withdrawal 0.724



Number of differences required for a stationary series

```
ndiffs(atm2.ts.bc)
```

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

Number of differences required for a seasonally stationary series

```
nsdiffs(atm2.ts.bc)
```

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

It shows number of differences required is 1 for boxcox transformed data.

```
atm2.ts.bc %>% diff(lag=7) %>% ur.kpss() %>% summary()
```

```
##
```

```
## #####
```

```
## # KPSS Unit Root Test #
```

```
## #####
```

```
##
```

```
## Test is of type: mu with 5 lags.
```

```
##
```

```
## Value of test-statistic is: 0.0162
```

```
##
```

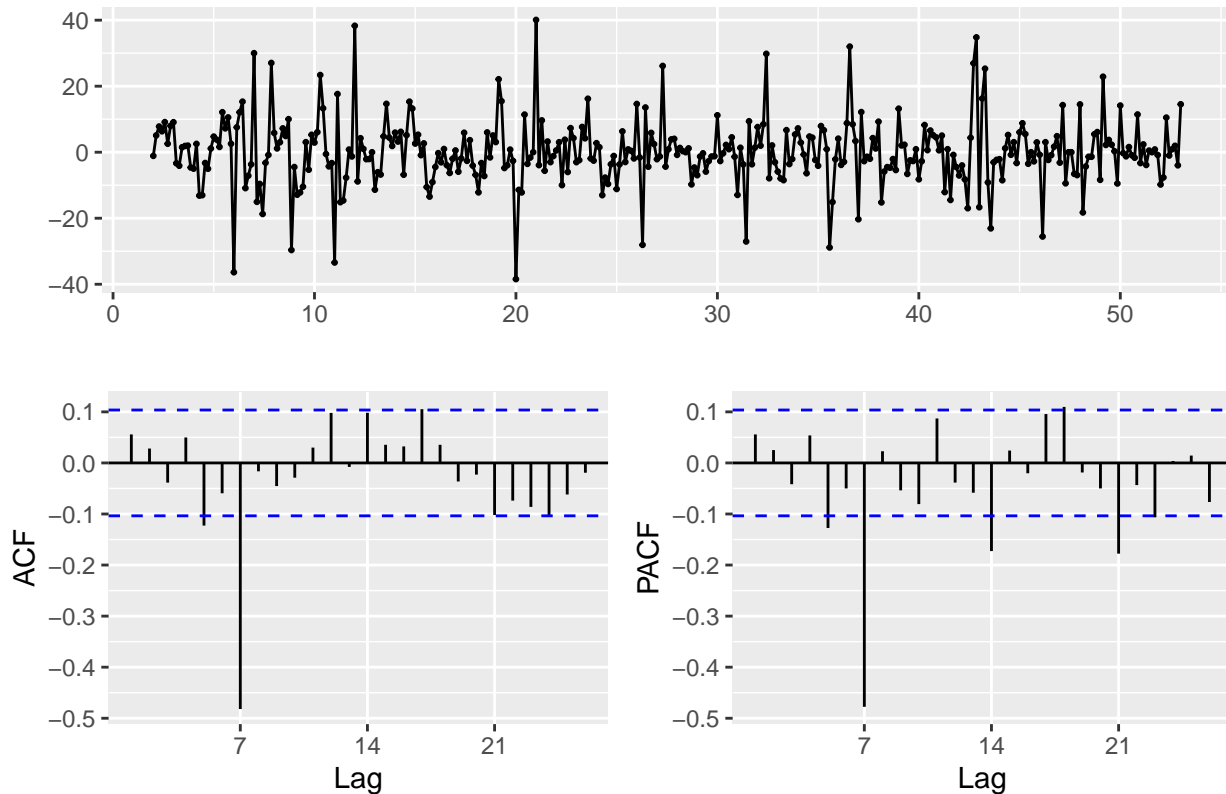
```
## Critical value for a significance level of:
```

```
##          10pct  5pct 2.5pct  1pct
```

```
## critical values 0.347 0.463 0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary

```
atm2.ts.bc %>% diff(lag=7) %>% ggtsdisplay()
```



First we will start with Holt-Winters damped method. Damping is possible with both additive and multiplicative Holt-Winters' methods. This method often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend.

```
# Holt Winters
```

```
atm2.ts %>% hw(h=31, seasonal = "additive", lambda = atm2.lambda, damped = TRUE)
```

##	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## 53.14286	67.727881	35.291267	105.26894	20.74561	126.87010
## 53.28571	74.012766	40.580383	112.34286	25.35441	134.30920
## 53.42857	10.844773	-3.254323	36.70434	-13.45333	53.31462
## 53.57143	1.648706	-13.353074	22.08418	-26.56518	36.83677
## 53.71429	101.948220	64.792300	143.32926	47.14368	166.72907
## 53.85714	92.500300	56.498440	132.92025	39.58380	155.86508
## 54.00000	68.866332	36.243721	106.55382	21.56968	128.22256
## 54.14286	67.775348	33.216555	108.21505	18.01659	131.58961
## 54.28571	74.059485	38.420870	115.33960	22.46113	139.10021
## 54.42857	10.871202	-4.387783	38.91404	-15.98503	57.04338
## 54.57143	1.663821	-14.982817	24.01057	-29.59257	40.21221
## 54.71429	101.993433	62.324951	146.52593	43.68529	171.80421
## 54.85714	92.542577	54.122362	136.04975	36.29148	160.84582
## 55.00000	68.903765	34.144880	109.49813	18.80244	132.94355
## 55.14286	67.811143	31.293904	110.99079	15.54866	136.05209
## 55.28571	74.094716	36.416629	118.16249	19.82956	143.62898
## 55.42857	10.891142	-5.535746	41.01325	-18.47284	60.59806
## 55.57143	1.675242	-16.563883	25.85263	-32.52085	43.44674
## 55.71429	102.027528	60.025873	149.53496	40.49965	176.59647

```
## 55.85714      92.574457  51.911131 138.99687  33.26820 165.55113
## 56.00000      68.931993  32.200914 112.27407  16.29845 137.40928
## 56.14286      67.838136  29.500528 113.62437  13.30674 140.29932
## 56.28571      74.121282  34.544212 120.84032  17.42301 147.93815
## 56.42857      10.906182  -6.688629  43.01959 -20.91749  64.00567
## 56.57143       1.683868 -18.101954  27.62307 -35.36252  46.56110
## 56.71429     102.053237  57.869192 152.38739  37.54566 181.15191
## 56.85714      92.598496  49.839436 141.79169  30.47392 170.02576
## 57.00000      68.953279  30.388346 114.90931  14.02175 141.66104
## 57.14286      67.858490  27.819168 116.13717  11.26493 144.36295
## 57.28571      74.141315  32.785857 123.39486  15.21412 152.06003
## 57.42857      10.917527  -7.840726  44.94651 -23.32042  67.28683
```

Next is to apply exponential smoothing method on this time series. It shows that the ETS(A, N, A) model best fits for the transformed ATM4, i.e. exponential smoothing with additive error, no trend component and additive seasonality.

```
# ETS
atm2.ts %>% ets(lambda = atm2.lambda)

## ETS(A,N,A)
##
## Call:
## ets(y = ., lambda = atm2.lambda)
##
## Box-Cox transformation: lambda= 0.7243
##
## Smoothing parameters:
##   alpha = 1e-04
##   gamma = 0.3852
##
## Initial states:
##   l = 26.7912
##   s = -17.8422 -13.3191 10.8227 1.8426 4.2781 5.7994
##       8.4185
##
## sigma: 8.5054
##
##      AIC      AICc      BIC
## 3727.060 3727.682 3766.059
```

We will now find out the appropriate ARIMA model for this time series. The suggested model seems ARIMA(3,0,3)(0,1,1)[7] with drift.

```
atm2.fit3 <- atm2.ts %>% auto.arima(lambda = atm2.lambda )
atm2.fit3

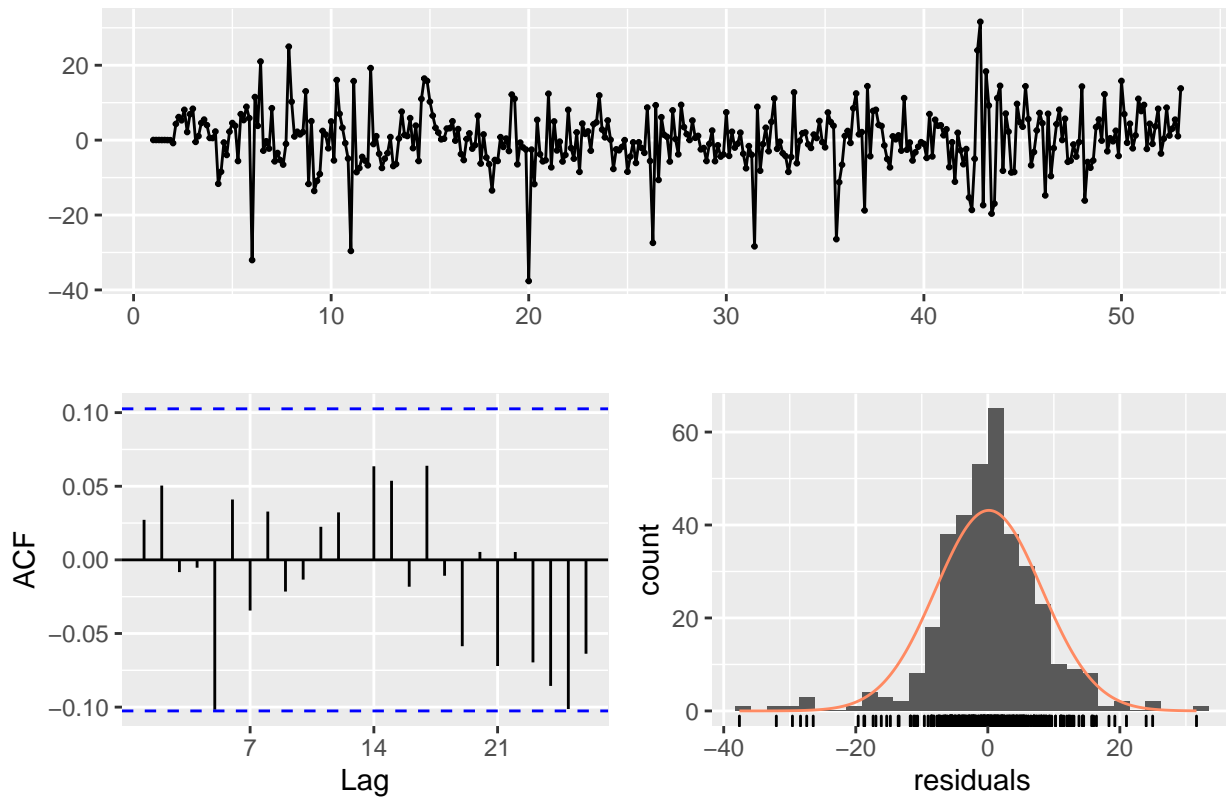
## Series: .
## ARIMA(3,0,3)(0,1,1)[7] with drift
## Box Cox transformation: lambda= 0.7242585
##
## Coefficients:
##      ar1      ar2      ar3      ma1      ma2      ma3      sma1      drift
##      0.4902 -0.4948  0.8326 -0.4823  0.3203 -0.7837 -0.7153 -0.0203
## s.e.  0.0863  0.0743  0.0614  0.1060  0.0941  0.0621  0.0453  0.0072
##
## sigma^2 estimated as 67.52: log likelihood=-1260.59
```

```
## AIC=2539.18   AICc=2539.69   BIC=2574.1
```

Next is to see residuals time series plot which shows residuals are being near normal with mean of the residuals being near to zero. Also there is no significant autocorrelation that confirms that forecasts are good.

```
checkresiduals(atm2.fit3)
```

Residuals from ARIMA(3,0,3)(0,1,1)[7] with drift

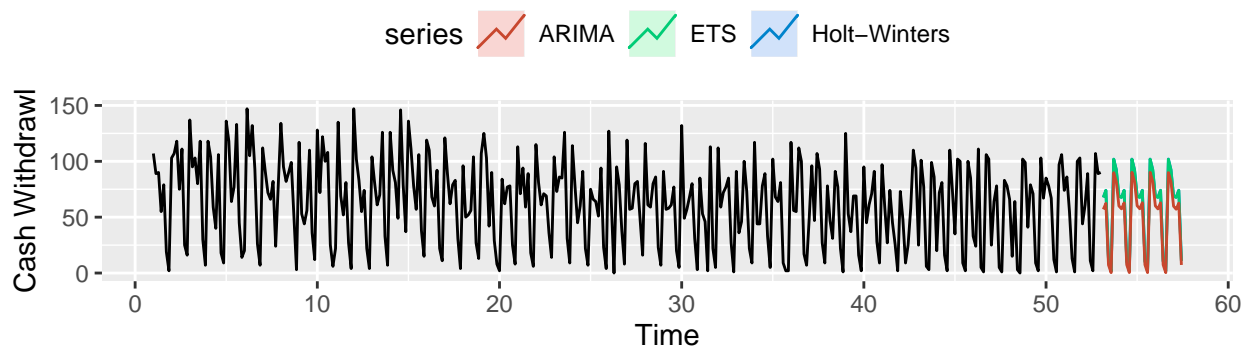


```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(3,0,3)(0,1,1)[7] with drift
## Q* = 8.944, df = 6, p-value = 0.1768
##
## Model df: 8.   Total lags used: 14
```

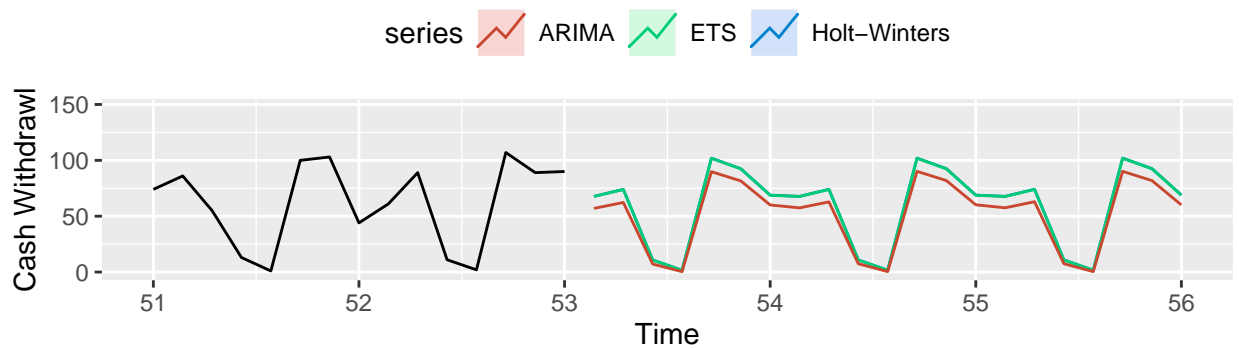
Next step is to plot the forecast for all the considered models above which will shows a nice visual comparison. it will also show a zoomed in plot to have a clearer view.

```
atm.forecast(atm2.ts)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



Zoom in

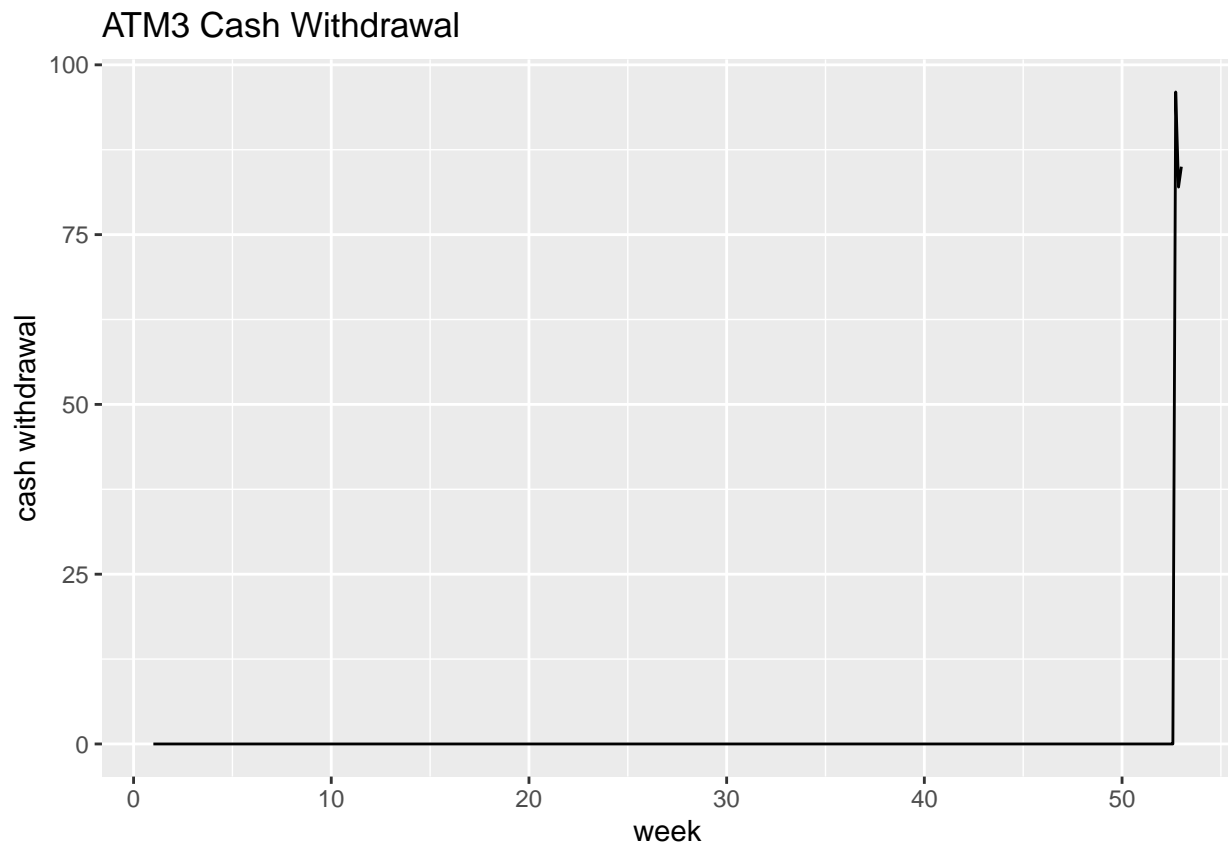


```
model_accuracy(atm2.ts,2)
```

```
##      Holt-Winters      ETS      ARIMA
## 1         57.20467 57.58101 56.58658
```

ATM3

```
atm3.ts <- atm.new %>% filter(ATM=="ATM3") %>% select(Cash) %>% ts(frequency = 7)
autoplot(atm3.ts, main="ATM3 Cash Withdrawal", ylab="cash withdrawal", xlab="week")
```

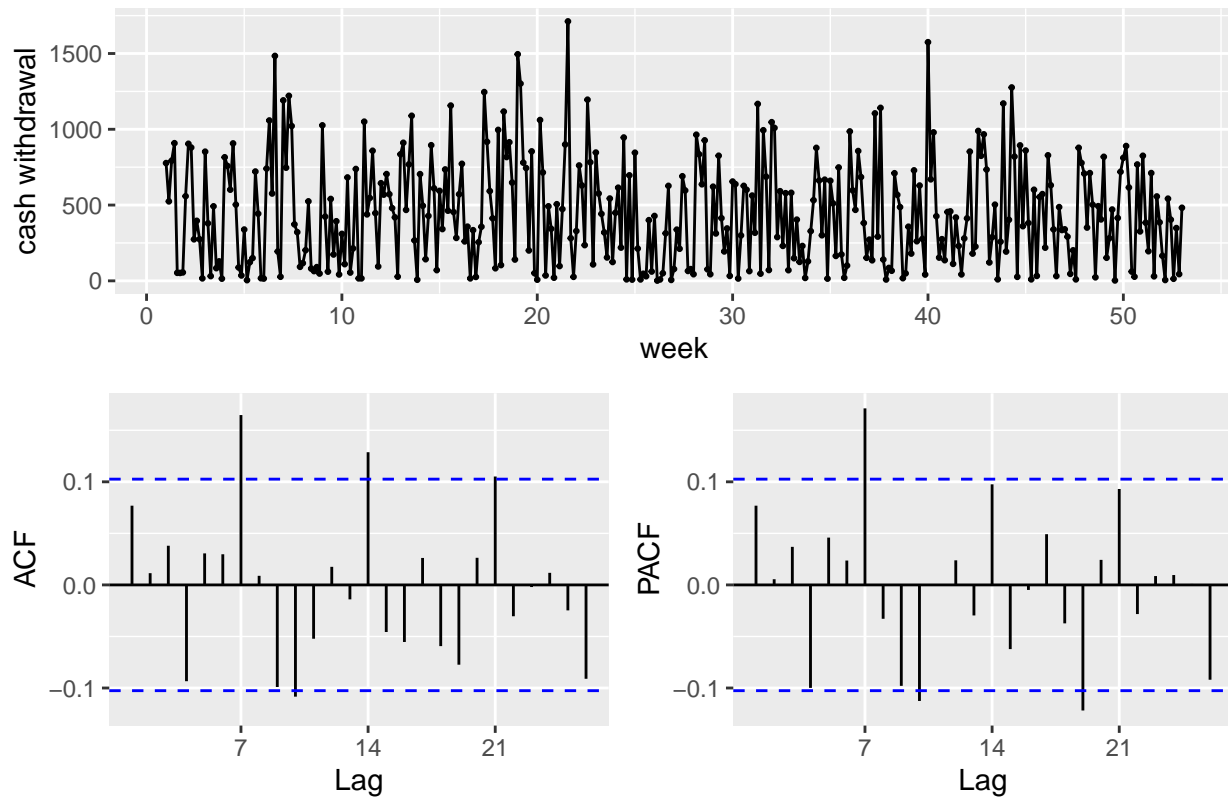


ATM4

Seeing the time series plot, it is apparent that there is seasonality in this series. ACF shows a decrease in every 7th lag. From the PACF, there are few significant lags at the beginning but others within critical limit. Overall, it is non stationary, having seasonality and might require differencing for it to become stationary.

```
atm4.ts <- atm.new %>% filter(ATM=="ATM4") %>% select(Cash) %>% ts(frequency = 7)
ggtsdisplay(atm4.ts, main="ATM4 Cash Withdrawal", ylab="cash withdrawal", xlab="week")
```

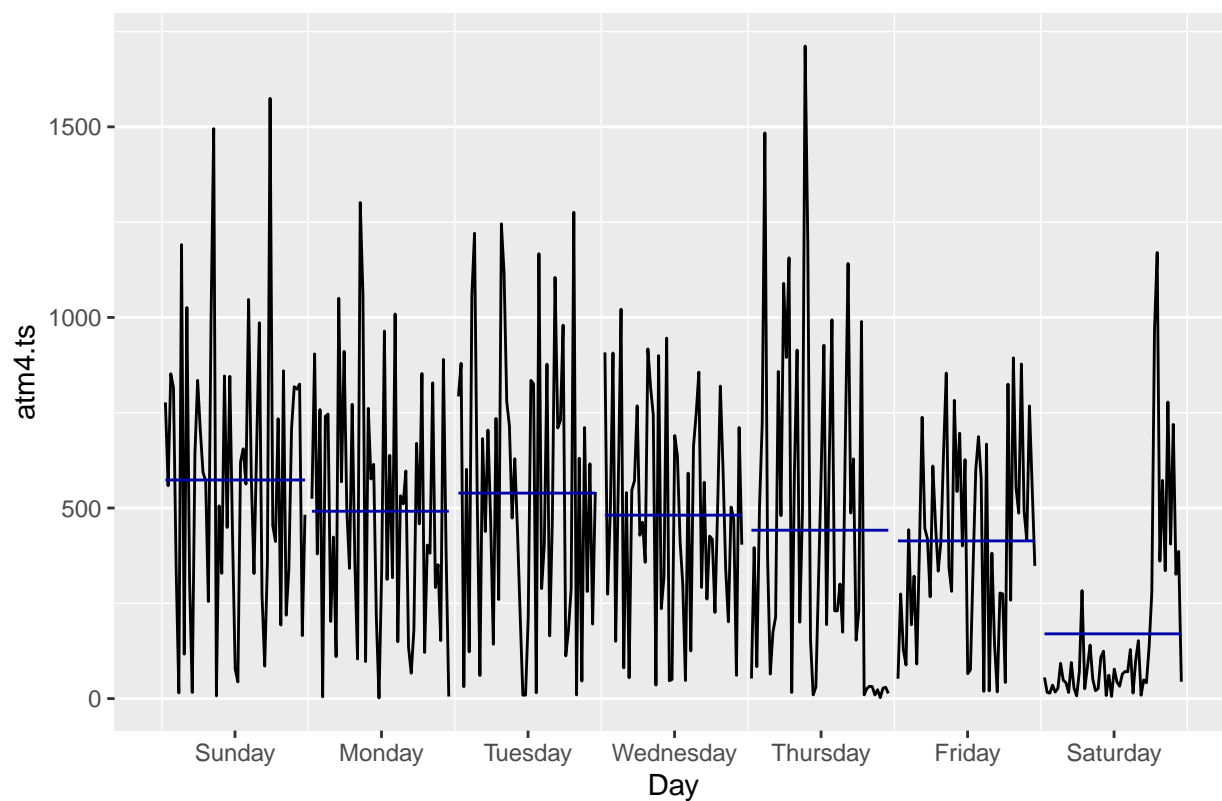
ATM4 Cash Withdrawal



From the subseries plot, it is clear that Sunday is having highest mean for cash withdrawal while Saturday has the lowest.

```
ggsubseriesplot(atm4.ts, main="ATM4 Cash Withdrawal")
```

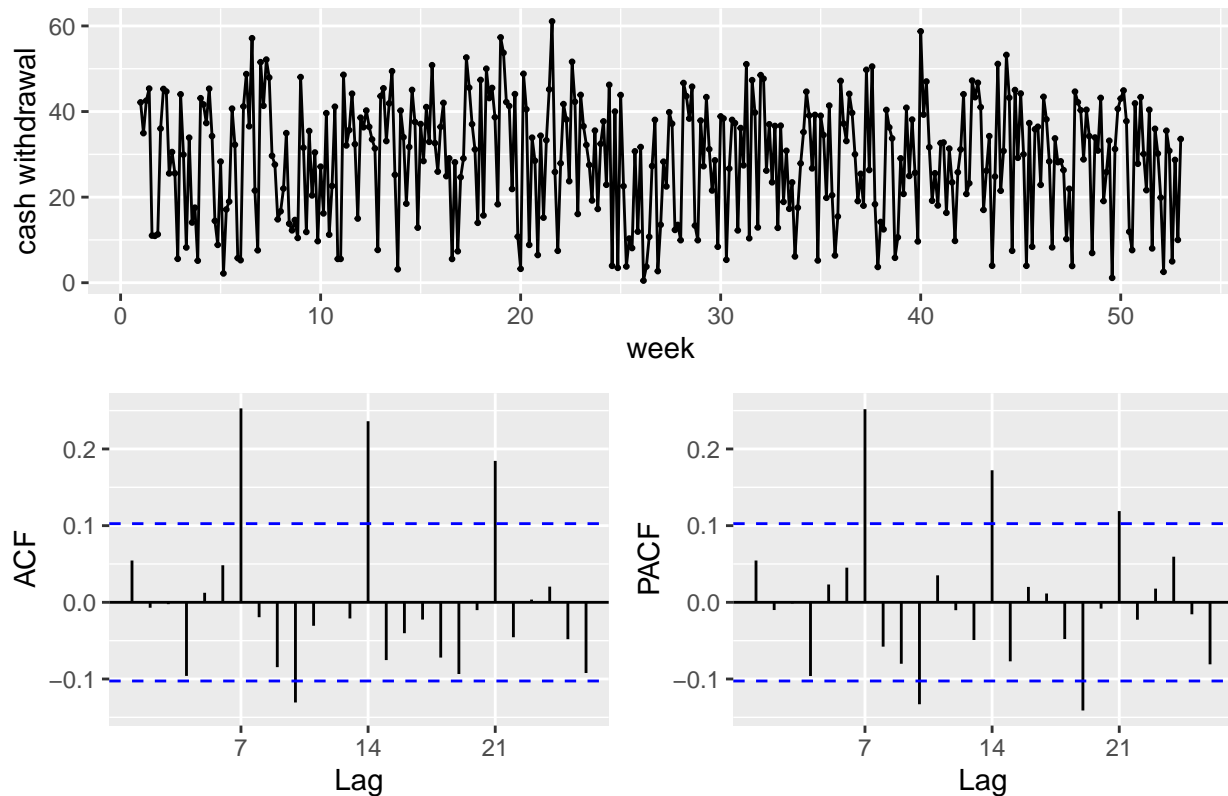
ATM4 Cash Withdrawal



Next step is to apply BoxCox transformation. With λ being 0.45, the resulting transformation does handle the variability in time series as shown in below transformed plot.

```
atm4.lambda <- BoxCox.lambda(atm4.ts)
atm4.ts.bc <- BoxCox(atm4.ts, atm4.lambda )
ggtsdisplay(atm4.ts.bc, main=paste("ATM4 Cash Withdrawal",round(atm4.lambda, 3)), ylab="cash withdrawal")
```


ATM4 Cash Withdrawal 0.45



```
# Number of differences required for a stationary series
ndiffs(atm4.ts.bc)
```

```
## [1] 0
```

```
# Number of differences required for a seasonally stationary series
nsdiffs(atm4.ts.bc)
```

```
## [1] 0
```

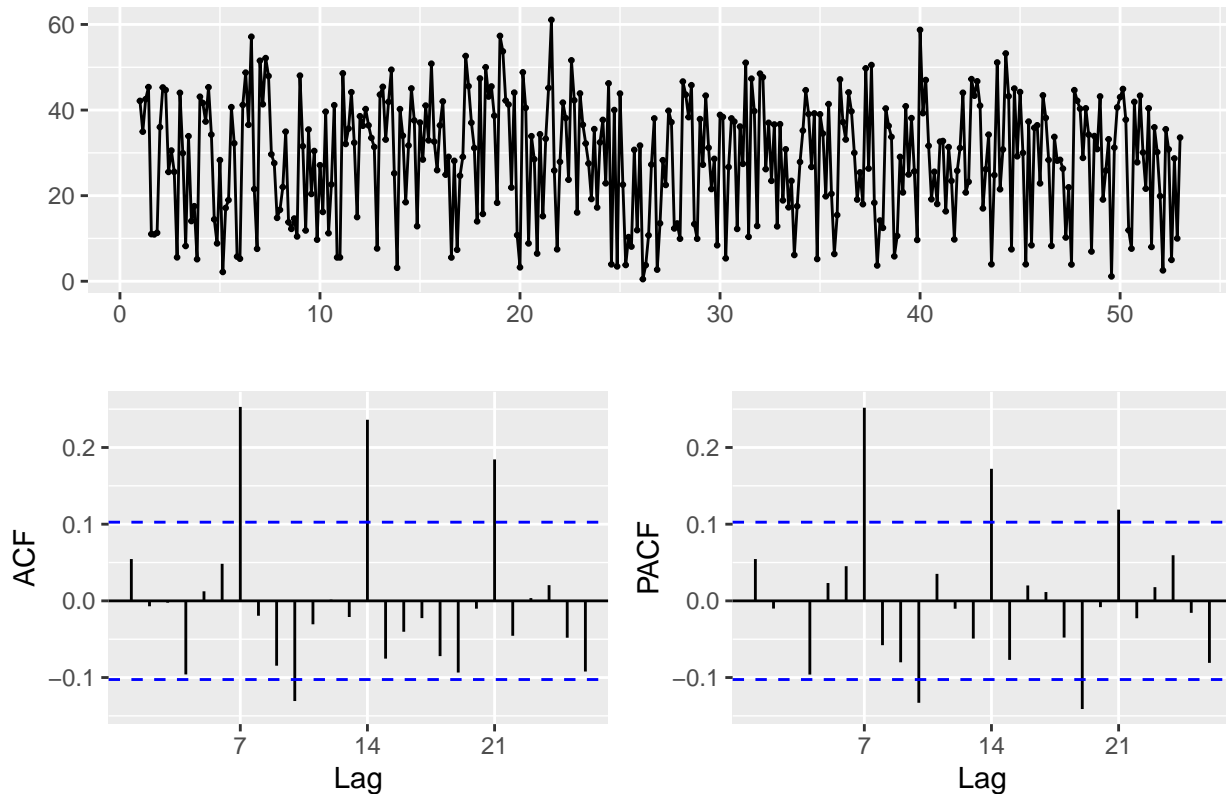
It shows number of differences required is 0 for boxcox transformed data.

```
atm4.ts.bc %>% ur.kpss() %>% summary()
```

```
##
## #####
## # KPSS Unit Root Test #
## #####
##
## Test is of type: mu with 5 lags.
##
## Value of test-statistic is: 0.0792
##
## Critical value for a significance level of:
##          10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463 0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary.

```
atm4.ts.bc %>% ggtsdisplay()
```



First we will start with Holt-Winters damped method. Damping is possible with both additive and multiplicative Holt-Winters' methods. This method often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend.

```
# Holt Winters
```

```
atm4.ts %>% hw(h=31, seasonal = "additive", lambda = atm4.lambda, damped = TRUE)
```

##	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## 53.14286	326.46664	5.361266e+01	872.7889	4.7560920	1283.0394
## 53.28571	390.55947	7.881312e+01	980.9502	12.8286778	1416.0583
## 53.42857	397.88339	8.186526e+01	993.0862	13.9675943	1430.9036
## 53.57143	88.16707	-1.188133e-04	412.7690	-21.7513686	696.1136
## 53.71429	437.83425	9.906165e+01	1058.5849	20.8852913	1510.7692
## 53.85714	284.50971	3.881453e+01	799.7425	1.5164332	1192.4004
## 54.00000	507.20922	1.308726e+02	1169.8559	35.4549744	1645.5454
## 54.14286	324.77262	5.208909e+01	874.0891	4.2406561	1287.4075
## 54.28571	388.90207	7.701404e+01	982.6924	11.9597845	1421.1069
## 54.42857	396.39921	8.010639e+01	995.1580	13.0852412	1436.3713
## 54.57143	87.59346	-4.150601e-03	414.2213	-22.8793652	700.0263
## 54.71429	436.60517	9.725297e+01	1061.2815	19.8415430	1517.0757
## 54.85714	283.65049	3.777331e+01	802.2506	1.2832703	1198.1842
## 55.00000	506.16225	1.288966e+02	1173.1625	34.1181908	1652.7103
## 55.14286	324.04660	5.092781e+01	877.1018	3.8375566	1293.9333
## 55.28571	388.19148	7.560926e+01	986.0591	11.2521458	1428.1862
## 55.42857	395.76275	7.870397e+01	998.6853	12.3531475	1443.6612
## 55.57143	87.34775	-1.273091e-02	416.4752	-23.8878631	705.0385
## 55.71429	436.07791	9.575384e+01	1065.1735	18.9437425	1524.8790

```
## 55.85714      283.28192  3.689963e+01  805.6726   1.0925953 1205.1449
## 56.00000      505.71298  1.272021e+02 1177.4724  32.9319224 1661.1294
## 56.14286      323.73508  4.992740e+01  880.8442   3.4901477 1301.3790
## 56.28571      387.88653  7.438035e+01  990.1166  10.6232674 1436.1287
## 56.42857      395.48959  7.746167e+01 1002.8304  11.6956591 1451.7237
## 56.57143       87.24235 -2.513721e-02  419.0707 -24.8511354  710.5204
## 56.71429      435.85159  9.439585e+01 1069.5705  18.1202396 1533.3133
## 56.85714      283.12372  3.610421e+01  809.4805   0.9275239 1212.6021
## 57.00000      505.52010  1.256379e+02 1182.2034  31.8238709 1670.0733
## 57.14286      323.60135  4.900310e+01  884.8928   3.1755185 1309.2095
## 57.28571      387.75561  7.323505e+01  994.4625  10.0390607 1444.4301
## 57.42857      395.37231  7.629643e+01 1007.2323  11.0813715 1460.1059
```

Next is to apply exponential smoothing method on this time series. It shows that the ETS(A, N, A) model best fits for the transformed ATM4, i.e. exponential smoothing with additive error, no trend component and additive seasonality.

```
# ETS
atm4.ts %>% ets(lambda = atm4.lambda)

## ETS(A,N,A)
##
## Call:
## ets(y = ., lambda = atm4.lambda)
##
## Box-Cox transformation: lambda= 0.4498
##
## Smoothing parameters:
##   alpha = 1e-04
##   gamma = 0.1035
##
## Initial states:
##   l = 28.6369
##   s = -18.6503 -3.3529 1.6831 4.7437 5.4471 4.9022
##       5.2271
##
## sigma: 12.9202
##
##      AIC      AICc      BIC
## 4032.268 4032.890 4071.267
```

Next we will find out the appropriate ARIMA model for this time series. The suggested model seems ARIMA(0,0,1)(2,0,0)[7] with non-zero mean.

```
# Arima
atm4.fit3 <- atm4.ts %>% auto.arima(lambda = atm4.lambda)
atm4.fit3

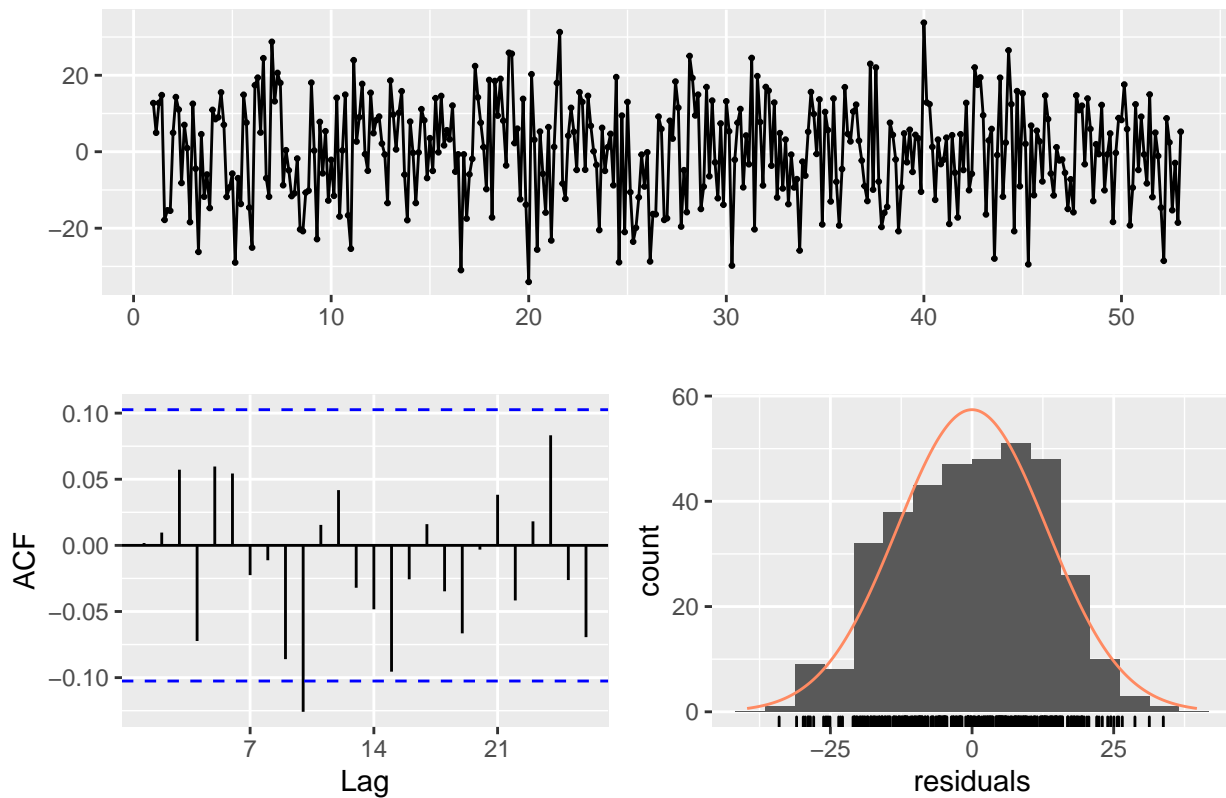
## Series: .
## ARIMA(0,0,1)(2,0,0)[7] with non-zero mean
## Box Cox transformation: lambda= 0.449771
##
## Coefficients:
##      ma1      sar1      sar2      mean
##    0.0790  0.2078  0.2023  28.6364
## s.e.  0.0527  0.0516  0.0525  1.2405
##
```

```
## sigma^2 estimated as 176.5: log likelihood=-1460.57
## AIC=2931.14 AICc=2931.3 BIC=2950.64
```

Next is to see residuals time series plot which shows residuals are being near normal with mean of the residuals being near to zero. Also there is no significant autocorrelation that confirms that forecasts are good.

```
checkresiduals(atm4.fit3)
```

Residuals from ARIMA(0,0,1)(2,0,0)[7] with non-zero mean

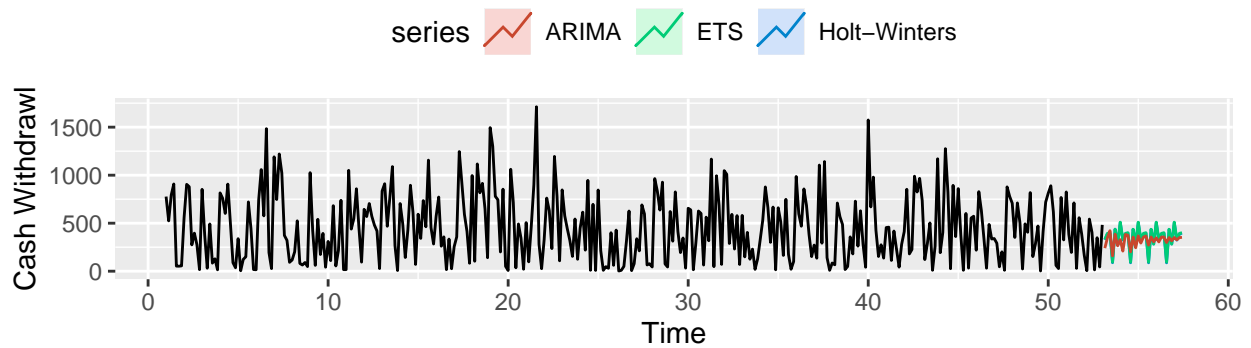


```
##
## Ljung-Box test
##
## data: Residuals from ARIMA(0,0,1)(2,0,0)[7] with non-zero mean
## Q* = 16.645, df = 10, p-value = 0.0826
##
## Model df: 4. Total lags used: 14
```

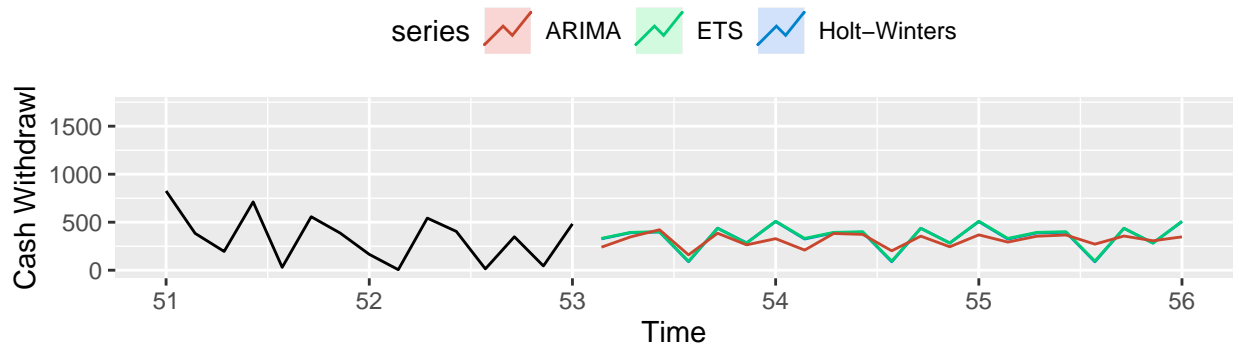
Next is to plot the forecast for all the considered models above which will shows a nice visual comparison. it will also show a zoomed in plot to have a clearer view.

```
atm.forecast(atm4.ts)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



Zoom in



```
model_accuracy(atm4.ts,4)
```

```
## Holt-Winters ETS ARIMA
## 1 360.3953 360.7951 315.7226
```

Forecast May, 2010

Part B - Forecasting Power

The dataset contains residential power usage for January 1998 until December 2013. Its monthly data from 1998 and power consumed is in KWH column. This dataset contains a total 192 records.

```
download.file(
  url="https://github.com/amit-kapoor/data624/blob/main/Project1/ResidentialCustomerForecastLoad-624.xlsx",
  destfile = temp.file,
  mode = "wb",
  quiet = TRUE)
power.data <- read_excel(temp.file, skip=0, col_types = c("numeric","text","numeric"))
head(power.data)
```

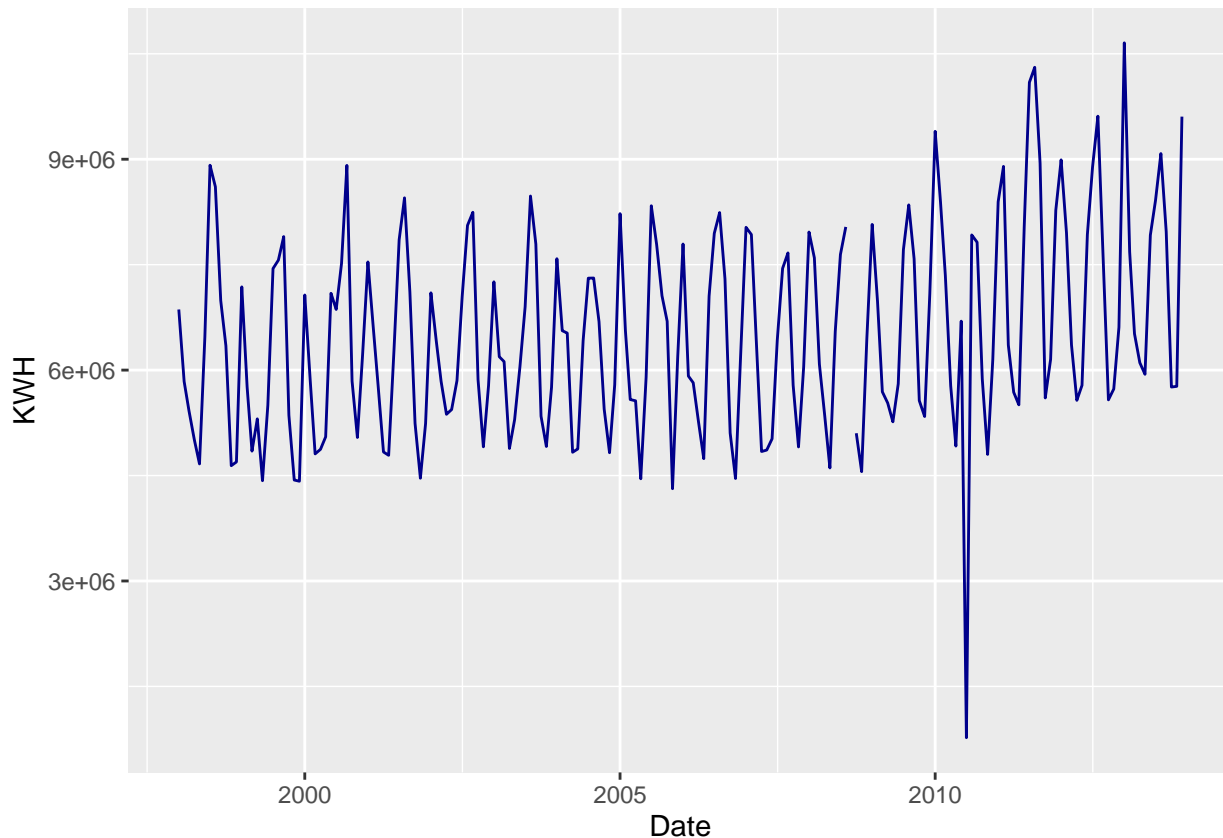
```
## # A tibble: 6 x 3
## CaseSequence `YYYY-MM` KWH
##      <dbl> <chr>      <dbl>
## 1      733 1998-Jan 6862583
## 2      734 1998-Feb 5838198
## 3      735 1998-Mar 5420658
## 4      736 1998-Apr 5010364
## 5      737 1998-May 4665377
## 6      738 1998-Jun 6467147
```

Exploratory Analysis

Seeing the plot closely, it is apparent that there is an outlier and a missing entry too. We will use the `tsclean` function to take care of missing entry and outlier in the data. Other than this data seems to be good shape.

```
power.data$`YYYY-MMM` <- paste0(power.data$`YYYY-MMM`, "-01")
power.data$Date <- lubridate::ymd(power.data$`YYYY-MMM`)

ggplot(power.data, aes(x=Date, y=KWH )) +
  geom_line(color="darkblue")
```



Data Cleaning

We will first create the time series of given data and then perform `tsclean` function.

```
power.ts <- ts(power.data$KWH, start=c(1998, 1), frequency = 12)
head(power.ts)
```

```
##           Jan      Feb      Mar      Apr      May      Jun
## 1998 6862583 5838198 5420658 5010364 4665377 6467147
```

```
power.ts %>% summary()
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.      NA's
##  770523 5429912 6283324 6502475 7620524 10655730         1
```

```
power.ts <- tsclean(power.ts)
power.ts %>% summary()
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

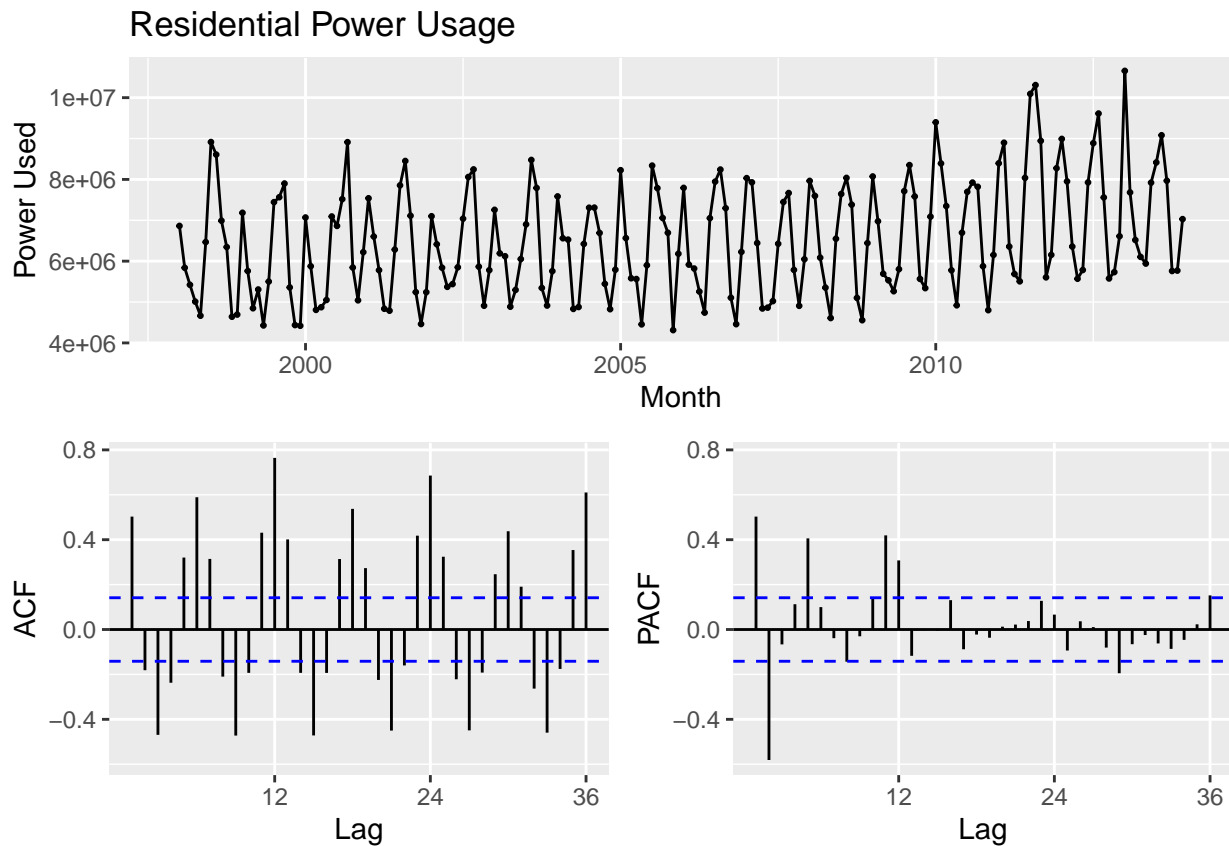
```
## 4313019 5443502 6351262 6529701 7608792 10655730
```

It is apparent that `tsclean` did take care of NA's and outlier in the data.

Time Series

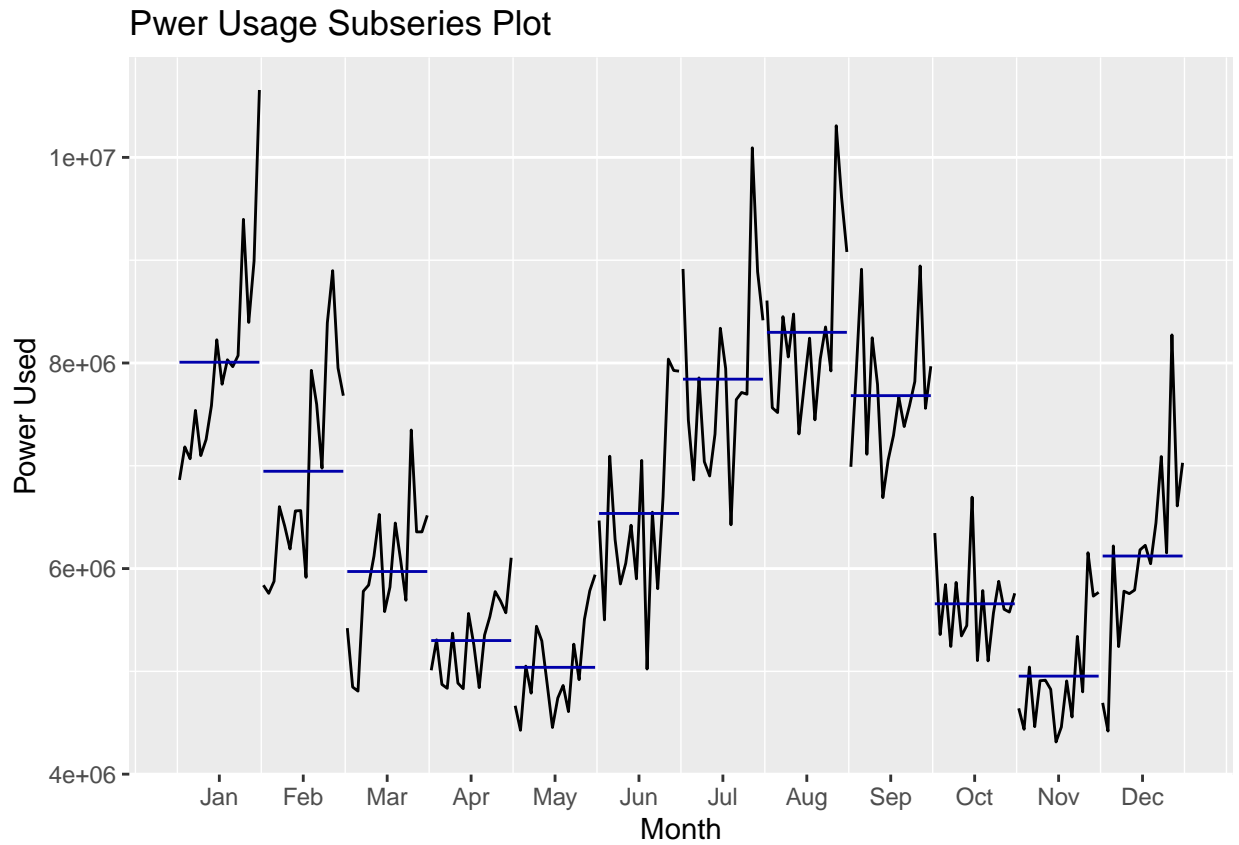
So far we have analyzed the data and perform data cleaning to handle missing and outlier data. In this section we will delve into the time series and see the models that perform best for prediction.

```
ggtsdisplay(power.ts, main="Residential Power Usage", ylab="Power Used", xlab="Month")
```

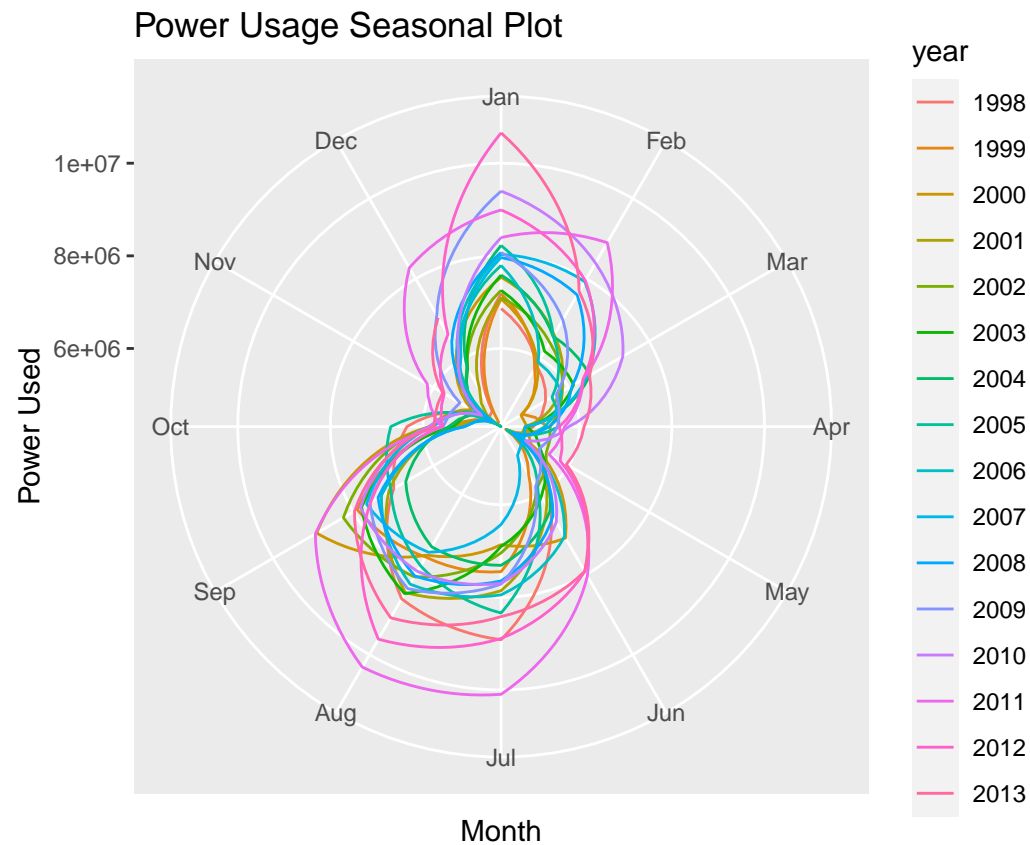


From the above time series plot, it is evident that seasonality exists in the data. We don't see a trend in the data. ACF plot shows the auto correlation and PACF shows few significant lags in the beginning. Overall, it shows seasonality and non-stationary data. It could require differencing to make it stationary which will be confirmed in further steps.

```
ggsubseriesplot(power.ts, main="Power Usage Subseries Plot", ylab="Power Used")
```

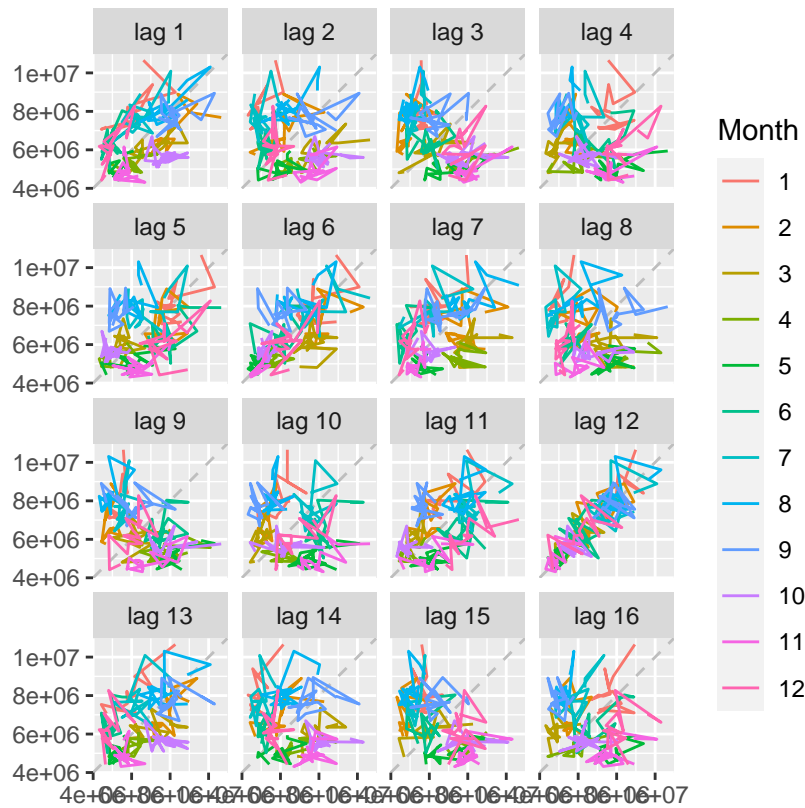


```
ggseasonplot(power.ts, polar=TRUE, main="Power Usage Seasonal Plot", ylab="Power Used")
```

The seasonal plots above shows a decline in power usage from Jan to May, increase till Aug and then decline in Nov. Aug is the month of most power consumption.

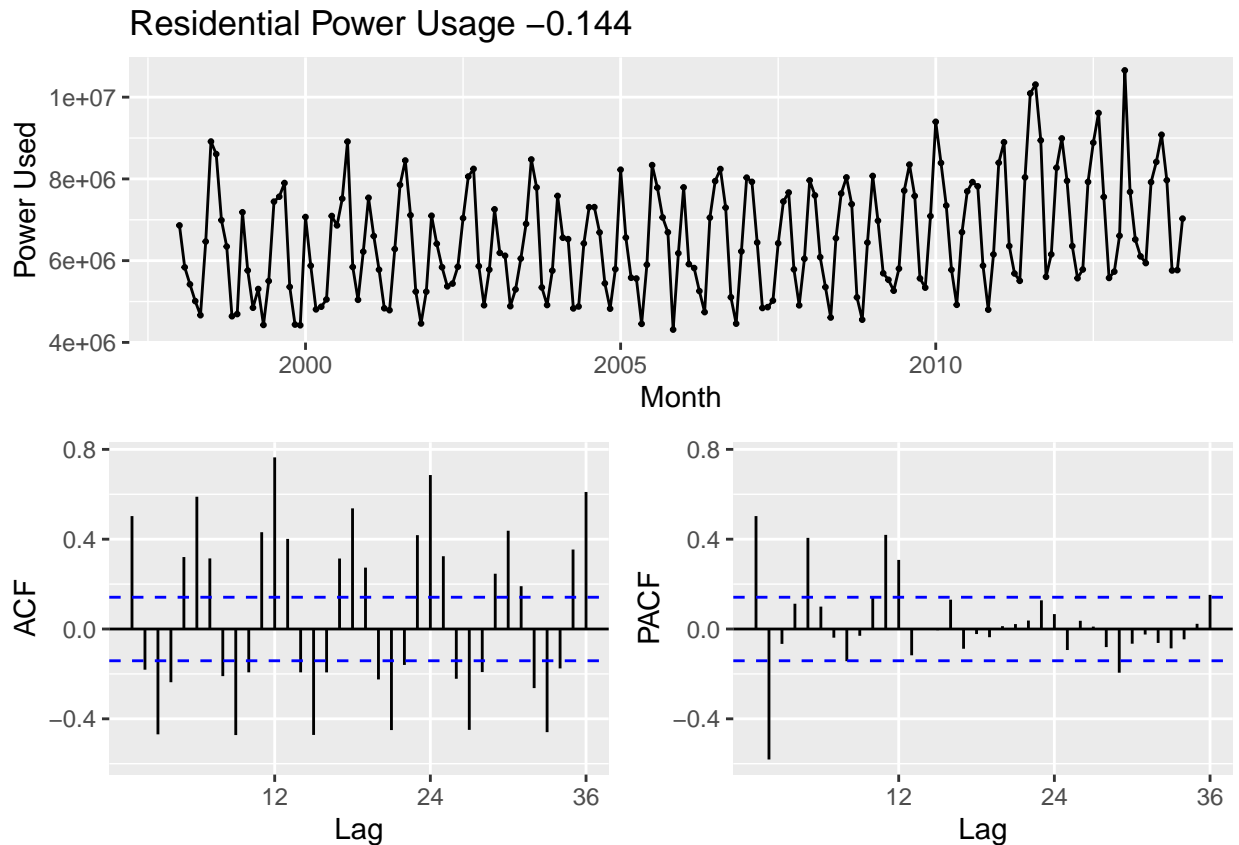
```
gglagplot(power.ts )
```



In the above lagplot, colors show different month. The lines connect points in chronological order. The relationship is strongly positive at lag 12, reflecting the strong seasonality in the data.

Next step is to apply Box-Cox transformation and check the transformed data.

```
powerts.lambda <- BoxCox.lambda(power.ts)
power.ts.bc <- BoxCox(power.ts, powerts.lambda )
ggtsdisplay(power.ts, main=paste("Residential Power Usage",round(powerts.lambda, 3)), ylab="Power Used"
```



The Box-Cox transformation above did handle the variation in the data with λ as 0.144 and appears stable now. Next we see that Number of differences required for a stationary and seasonally stationary series are 1.

```
# Number of differences required for a stationary series
ndiffs(power.ts.bc)
```

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

```
# Number of differences required for a seasonally stationary series
nsdiffs(power.ts.bc)
```

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

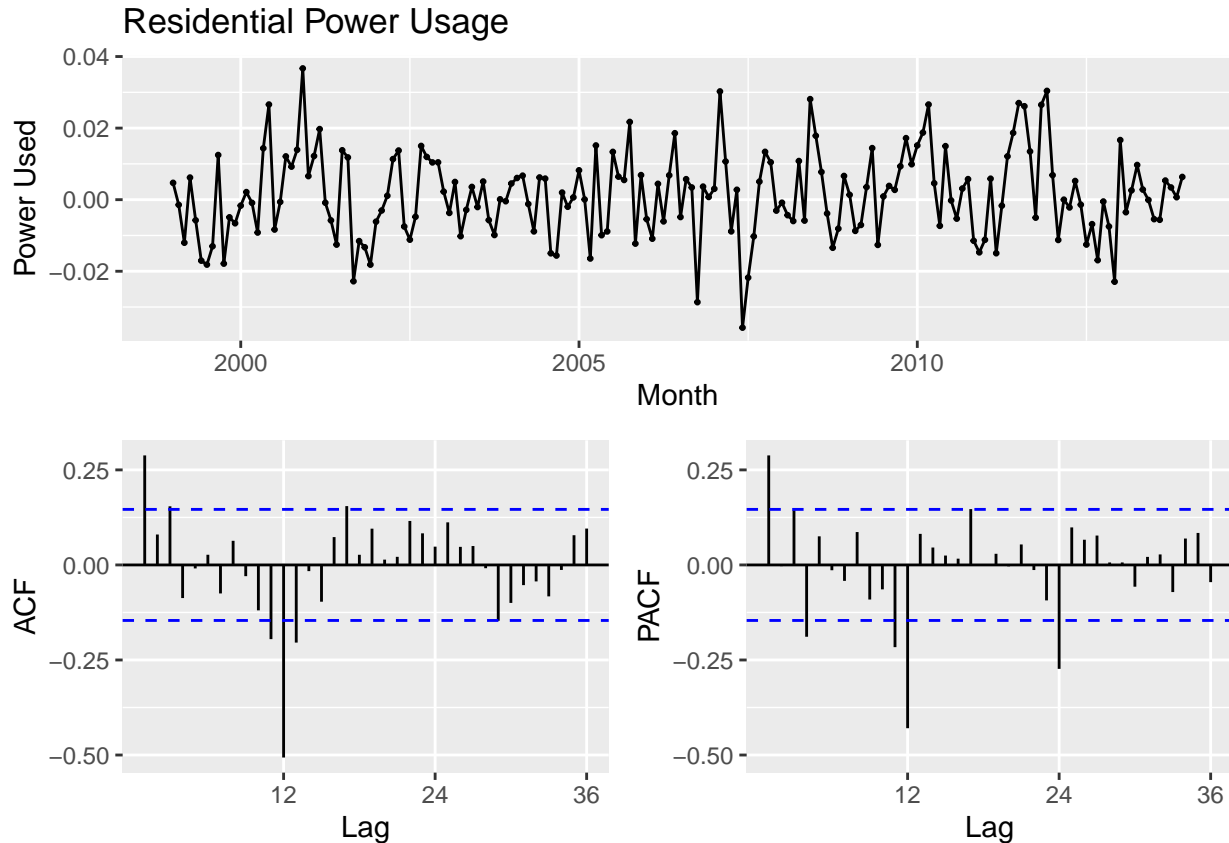
It shows number of differences required is 1 for boxcox transformed data.

```
power.ts.bc %>% diff(lag=12) %>% ur.kpss() %>% summary()
```

```
##
## #####
## # KPSS Unit Root Test #
## #####
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.1049
##
## Critical value for a significance level of:
##      10pct  5pct  2.5pct  1pct
## critical values 0.347 0.463 0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary.

```
power.ts.bc %>%
  diff(lag=12) %>%
  ggtsdisplay(main="Residential Power Usage", ylab="Power Used", xlab="Month")
```



Now we will apply four models in this time series: Holt Winters additive with damped True, Holt Winters multiplicative with damped True, exponential smoothing and arima. First we will start with Holt-Winters damped method. Damping is possible with both additive and multiplicative Holt-Winters' methods. This method often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend.

```
# Holt Winters additive with damped True
power.ts %>% hw(h=31, seasonal = "additive", lambda = powerts.lambda, damped = TRUE)
```

##	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## Jan 2014	9107297	8076875	10290936	7585779	10988269
## Feb 2014	7770646	6904466	8763438	6490967	9347335
## Mar 2014	6660179	5928390	7497188	5578499	7988664
## Apr 2014	5969397	5319026	6712381	5007781	7148238
## May 2014	5625182	5013393	6323915	4720557	6733732
## Jun 2014	7275964	6451637	8223053	6058824	8781111
## Jul 2014	8859946	7822939	10057288	7330606	10765524
## Aug 2014	9322020	8216844	10600620	7692933	11358099
## Sep 2014	8668636	7644836	9852384	7159284	10553344
## Oct 2014	6321057	5601315	7148606	5258532	7636511
## Nov 2014	5469798	4855427	6174761	4562382	6589737
## Dec 2014	6775208	5987467	7683813	5613190	8220830

```
## Jan 2015      9113992 8005052 10402262 7480984 11167935
## Feb 2015      7776112 6844659 8855485 6403639 9495747
## Mar 2015      6664665 5878334 7573701 5505361 8111924
## Apr 2015      5973271 5274971 6779388 4943398 7256143
## May 2015      5628725 4972370 6386125 4660623 6833931
## Jun 2015      7280622 6397096 8306589 5979421 8916161
## Jul 2015      8865659 7755195 10161867 7232276 10935129
## Aug 2015      9327948 8145442 10711046 7589442 11537448
## Sep 2015      8673977 7579322 9953345 7064335 10717313
## Oct 2015      6324700 5555572 7218128 5192069 7749092
## Nov 2015      5472821 4816718 6233216 4506100 6684315
## Dec 2015      6778989 5938447 7758475 5542135 8342027
## Jan 2016      9119188 7936962 10507290 7382692 11339177
## Feb 2016      7780354 6787930 8942347 6321671 9637208
## Mar 2016      6668146 5830831 7645926 5436663 8229419
## Apr 2016      5976278 5233146 6842661 4882887 7359020
## May 2016      5631476 4933411 6444895 4604266 6929491
## Jun 2016      7284237 6345305 8385579 5904772 9045098
## Jul 2016      8870093 7690866 10260839 7139837 11097210
```

```
# Holt Winters multiplicative with damped True
power.ts %>% hw(h=31, seasonal = "multiplicative", damped = TRUE)
```

```
##      Point Forecast    Lo 80      Hi 80    Lo 95      Hi 95
## Jan 2014      9017833 7957065 10078601 7395529 10640137
## Feb 2014      7828457 6875211 8781704 6370593 9286322
## Mar 2014      6739385 5891755 7587016 5443046 8035725
## Apr 2014      5958146 5185614 6730678 4776661 7139631
## May 2014      5658721 4903631 6413811 4503910 6813531
## Jun 2014      7362538 6353007 8372069 5818594 8906483
## Jul 2014      8756962 7524819 9989104 6872562 10641361
## Aug 2014      9316480 7972982 10659977 7261778 11371181
## Sep 2014      8596291 7327223 9865359 6655419 10537163
## Oct 2014      6299672 5348552 7250791 4845060 7754283
## Nov 2014      5499685 4651304 6348065 4202198 6797171
## Dec 2014      6805669 5733940 7877398 5166601 8444737
## Jan 2015      9020418 7571435 10469400 6804390 11236445
## Feb 2015      7830653 6548528 9112778 5869812 9791493
## Mar 2015      6741234 5616962 7865507 5021809 8460660
## Apr 2015      5959745 4947971 6971519 4412371 7507120
## May 2015      5660207 4682622 6637793 4165119 7155295
## Jun 2015      7364430 6071164 8657697 5386550 9342311
## Jul 2015      8759163 7195971 10322356 6368467 11149860
## Aug 2015      9318772 7629505 11008038 6735261 11902282
## Sep 2015      8598360 7015852 10180868 6178123 11018597
## Oct 2015      6301155 5124217 7478094 4501183 8101127
## Nov 2015      5500952 4458640 6543264 3906873 7095031
## Dec 2015      6807204 5499264 8115143 4806883 8807524
## Jan 2016      9022407 7265105 10779709 6334846 11709969
## Feb 2016      7832343 6286498 9378187 5468177 10196508
## Mar 2016      6742658 5394587 8090729 4680961 8804355
## Apr 2016      5960977 4754078 7167875 4115184 7806769
## May 2016      5661351 4500929 6821774 3886639 7436064
## Jun 2016      7365887 5837827 8893947 5028921 9702853
## Jul 2016      8760859 6921934 10599783 5948466 11573251
```

Next model to ETS: Exponential Smoothing methods.

```
# exponential smoothing
power.ts %>% ets(lambda = powerts.lambda, biasadj = TRUE)

## ETS(A,Ad,A)
##
## Call:
## ets(y = ., lambda = powerts.lambda, biasadj = TRUE)
##
## Box-Cox transformation: lambda= -0.1443
##
## Smoothing parameters:
##   alpha = 0.118
##   beta  = 1e-04
##   gamma = 1e-04
##   phi   = 0.979
##
## Initial states:
##   l = 6.1998
##   b = 1e-04
##   s = -0.006 -0.0285 -0.0132 0.019 0.0263 0.0212
##         0.0014 -0.0255 -0.0192 -0.0077 0.0081 0.024
##
## sigma: 0.0094
##
##      AIC      AICc      BIC
## -765.9795 -762.0258 -707.3446
```

We can see here that the ets model that best describes the data is **ETS(A,Ad,A)** i.e. exponential smoothing with additive error, additive damped trend and additive seasonality.

Next we will find the best Arima model that fits this time series data.

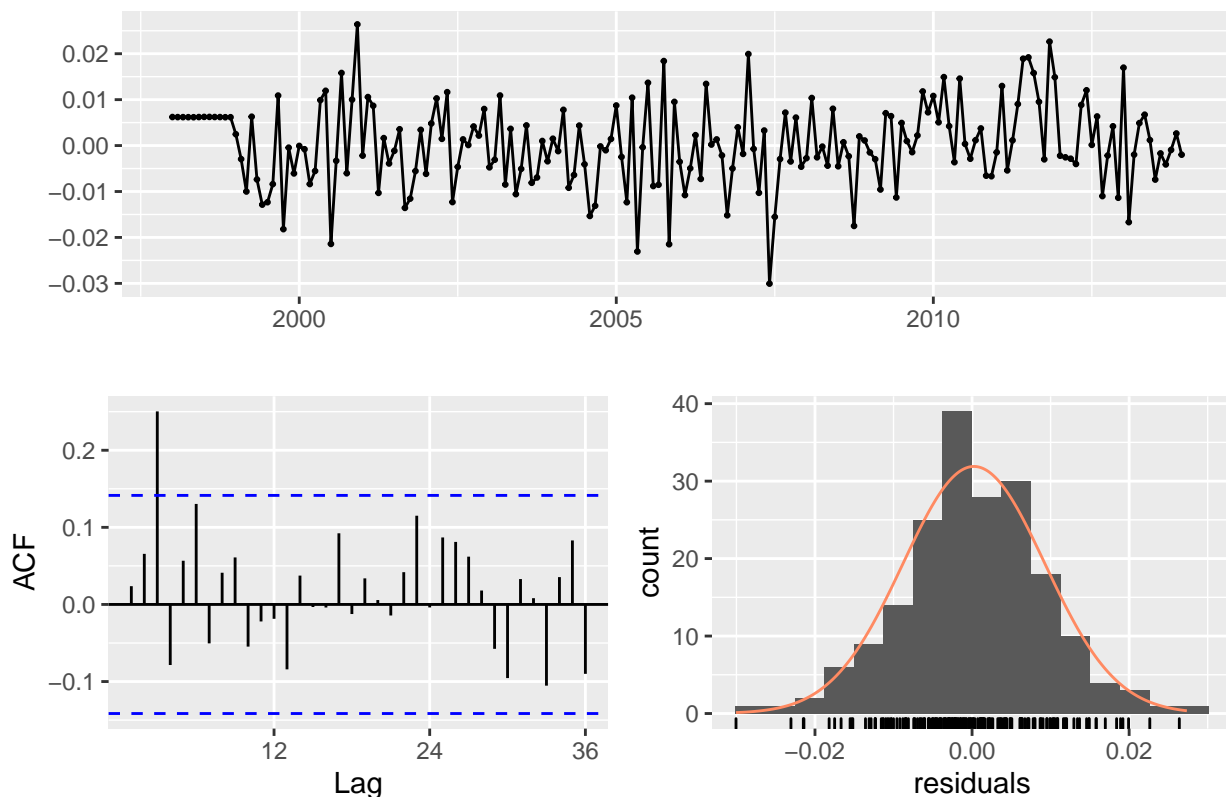
```
power.fit4 <- power.ts %>% auto.arima(lambda = powerts.lambda, biasadj = TRUE)
power.fit4

## Series: .
## ARIMA(0,0,1)(2,1,0)[12] with drift
## Box Cox transformation: lambda= -0.1442665
##
## Coefficients:
##      ma1      sar1      sar2  drift
##      0.2563 -0.7036 -0.3817 1e-04
## s.e.  0.0809  0.0734  0.0748 1e-04
##
## sigma^2 estimated as 8.869e-05: log likelihood=585.32
## AIC=-1160.65 AICc=-1160.3 BIC=-1144.68
```

The best Arima model comes out is **ARIMA(0,0,1)(2,1,0)[12]** with drift.

```
checkresiduals(power.fit4)
```

Residuals from ARIMA(0,0,1)(2,1,0)[12] with drift



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,1)(2,1,0)[12] with drift
## Q* = 28.193, df = 20, p-value = 0.1049
##
## Model df: 4.    Total lags used: 24
```

Next is to plot the forecasts using all of 4 models described above: HW additive, HW multiplicative, ets and arima. For this, we will create a generic function which will accept the time series and plot the forecast for all these 4 models. There is also a zoomed in plot of the forecast for better clarity.

```
# function to plot forecast(s)
power.forecast <- function(timeseries) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)

  # models for forecast
  hwa.model <- timeseries %>% hw(h=12, seasonal = "additive", lambda = lambda, damped = TRUE)
  hwm.model <- timeseries %>% hw(h=12, seasonal = "multiplicative", damped = TRUE)
  ets.model <- timeseries %>% ets(lambda = lambda)
  arima.model <- timeseries %>% auto.arima(lambda = lambda, biasadj = TRUE)

  # forecast
  pow.hwa.fcst <- forecast(hwa.model, h=12)
  pow.hwm.fcst <- forecast(hwm.model, h=12)
  pow.ets.fcst <- forecast(ets.model, h=12)
  pow.arima.fcst <- forecast(arima.model, h=12)
```

```

# plot forecasts
p1 <- autoplot(timeseries) +
  autolayer(pow.hwa.fcst, PI=FALSE, series="Holt-Winters Additive") +
  autolayer(pow.hwm.fcst, PI=FALSE, series="Holt-Winters Multiplicative") +
  autolayer(pow.ets.fcst, PI=FALSE, series="ETS") +
  autolayer(pow.arima.fcst, PI=FALSE, series="ARIMA") +
  theme(legend.position = "top") +
  ylab("Power Used")

# zoom in plot
p2 <- p1 +
  labs(title = "Zoom in ") +
  xlim(c(2012,2015))

grid.arrange(p1,p2,ncol=1)
}

```

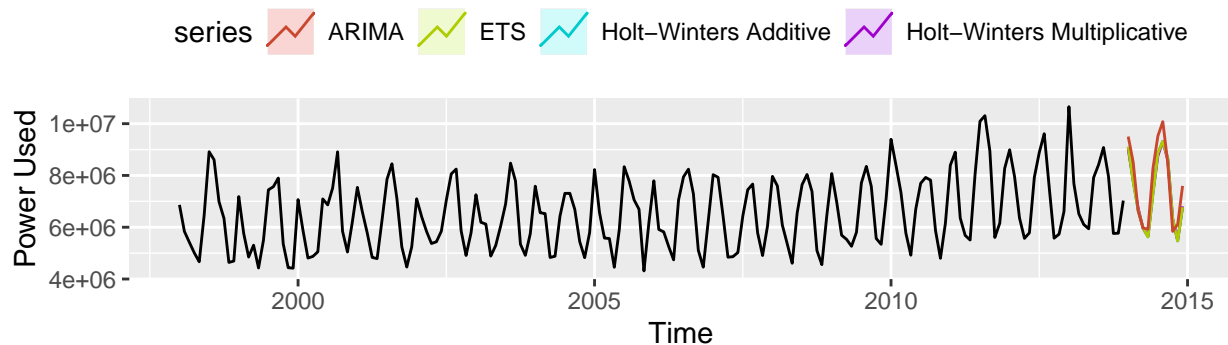
Lets plot the forecast now using the above function for power usage.

```

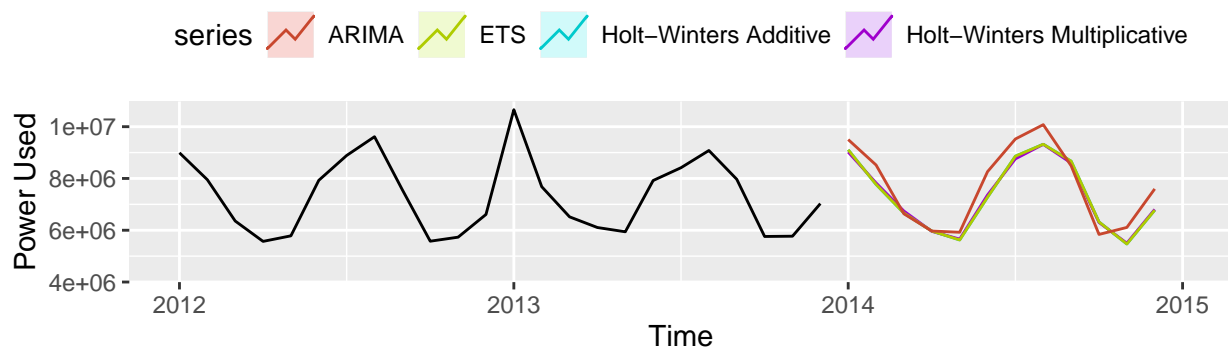
power.arima.fcst <- forecast(power.fit4, h=12)
power.forecast(power.ts)

```

Scale for 'x' is already present. Adding another scale for 'x', which will
replace the existing scale.



Zoom in



Now we will check the accuracy of all 4 models. Again for this purpose, we have created a function that accepts the timeseries and divide the data for training and testing. In this function we will first divide the data for training and testing, train all models with train set and then find out RMSE using test data.


```

powm_accuracy <- function(timeseries) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)

  # split the data to train and test
  train <- window(timeseries, end=c(2009, 12))
  test <- window(timeseries, start=2010)

  # models for forecast
  hwa.model <- train %>% hw(h=length(train), seasonal = "additive", lambda = lambda, damped = TRUE)
  hwm.model <- train %>% hw(h=length(train), seasonal = "multiplicative", damped = TRUE)
  ets.model <- train %>% ets(model="AAA", lambda = lambda, biasadj = TRUE)
  arima.model <- train %>% Arima(order=c(0,0,1),
                                seasonal = c(2,1,0),
                                include.drift = TRUE,
                                lambda = lambda,
                                biasadj = TRUE)

  # forecast
  hwa.frct = forecast(hwa.model, h = length(test))$mean
  hwm.frct = forecast(hwm.model, h = length(test))$mean
  ets.frct = forecast(ets.model, h = length(test))$mean
  arima.frct = forecast(arima.model, h = length(test))$mean

  # dataframe having rmse
  rmse = data.frame(RMSE=cbind(accuracy(hwa.frct, test)[,2],
                                accuracy(hwm.frct, test)[,2],
                                accuracy(ets.frct, test)[,2],
                                accuracy(arima.frct, test)[,2]))
  names(rmse) = c("Holt-Winters Additive", "Holt-Winters Multiplicative", "ETS", "ARIMA")

  # display rmse
  rmse
}

```

```
powm_accuracy(power.ts)
```

```
## Holt-Winters Additive Holt-Winters Multiplicative ETS ARIMA
## 1 1172803 1141901 1151275 950035.4
```

Thus **ARIMA(0,0,1)(2,1,0)[12]** with drift has been the best model to describe the given time series.

Forecast 2014

In this last step we will perform the forecast for 2014.

```

pow.fcst.date <- seq(as.Date('2014-01-01'), as.Date('2014-12-01'), by="month") %>% format("%Y-%b")

write.xlsx(data.frame('DateTime' = pow.fcst.date, 'Waterflow' = power.arima.fcst$mean),
            "Kapoor_data624_pow_forecasts.xlsx")

```

```

pow.fcst.ak <- read_excel("Kapoor_data624_pow_forecasts.xlsx", skip=0, col_types = c("text","numeric"))
pow.fcst.ak %>%
  kbl() %>%

```

DateTime	Waterflow
2014-Jan	9501939
2014-Feb	8519991
2014-Mar	6633101
2014-Apr	5970555
2014-May	5921709
2014-Jun	8262795
2014-Jul	9524528
2014-Aug	10077395
2014-Sep	8489194
2014-Oct	5838172
2014-Nov	6108522
2014-Dec	7596503

```
kable_paper() %>%
  scroll_box(width = "500px", height = "200px")
```

Part C - Waterflow Pipe

In part C we have been provided 2 datasets for waterflow pipes. These are simple 2 columns sets, however they have different time stamps. Each dataset contains 1000 records and has no missing data.

```
download.file(url="https://github.com/amit-kapoor/data624/blob/main/Project1/Waterflow_Pipe1.xlsx?raw=t",
             destfile = temp.file,
             mode = "wb",
             quiet = TRUE)
pipe1.data <- read_excel(temp.file, skip=0, col_types = c("date","numeric"))

download.file(url="https://github.com/amit-kapoor/data624/blob/main/Project1/Waterflow_Pipe2.xlsx?raw=t",
             destfile = temp.file,
             mode = "wb",
             quiet = TRUE)

pipe2.data <- read_excel(temp.file, skip=0, col_types = c("date","numeric"))
```

It is apparent here that the data is recorded on different time intervals for pipe1 and pipe2. For pipe1 it shows records for multiple time intervals within an hour i.e. not evenly distributed while for pipe2 it seems hourly recorded.

```
pipe1.data %>%
  kbl() %>%
  kable_paper() %>%
  scroll_box(width = "500px", height = "200px")
```

```
pipe2.data %>%
  kbl() %>%
  kable_paper() %>%
  scroll_box(width = "500px", height = "200px")
```

Exploratory Analysis

In this first step for analysis, We will check the daily frequency count for both datasets. It is apparent here that number of records daily for pipe1 is not evenly distributed and there are more recording per hour for

Date Time	WaterFlow
2015-10-23 00:24:06	23.369599
2015-10-23 00:40:02	28.002881
2015-10-23 00:53:51	23.065895
2015-10-23 00:55:40	29.972809
2015-10-23 01:19:17	5.997953
2015-10-23 01:23:58	15.935223
2015-10-23 01:50:05	26.617330
2015-10-23 01:55:33	33.282900
2015-10-23 01:59:15	12.426692
2015-10-23 02:51:51	21.833494
2015-10-23 02:59:49	8.483647
2015-10-23 03:14:40	29.336901
2015-10-23 03:18:09	19.809146
2015-10-23 03:22:13	31.019744
2015-10-23 03:46:57	18.339962
2015-10-23 03:59:21	16.888527
2015-10-23 04:11:50	13.664312
2015-10-23 04:45:43	17.300074
2015-10-23 05:05:41	23.260984
2015-10-23 05:13:49	8.152496
2015-10-23 05:14:38	19.875628
2015-10-23 05:15:41	32.886499
2015-10-23 05:16:57	22.260364
2015-10-23 05:32:35	5.778369
2015-10-23 05:42:52	32.545557
2015-10-23 05:44:30	30.687744
2015-10-23 05:45:52	29.080907
2015-10-23 06:03:38	30.047784
2015-10-23 06:17:16	5.752456
2015-10-23 06:33:18	30.414162
2015-10-23 06:33:38	26.175716
2015-10-23 06:38:30	27.155307
2015-10-23 06:42:23	13.605943
2015-10-23 06:56:22	11.184568
2015-10-23 06:59:29	20.383057
2015-10-23 07:36:34	13.405331
2015-10-23 07:55:43	14.461091
2015-10-23 07:56:42	27.234671
2015-10-23 08:11:22	9.089498
2015-10-23 08:26:45	29.162384
2015-10-23 08:58:29	24.123214
2015-10-23 09:00:59	6.207443
2015-10-23 09:11:30	27.666923
2015-10-23 09:35:45	29.781898
2015-10-23 10:01:02	19.035463
2015-10-23 10:05:50	14.539055
2015-10-23 10:35:35	16.304829
2015-10-23 11:02:31	9.089600
2015-10-23 11:03:49	24.160478
2015-10-23 11:38:25	33.013436
2015-10-23 12:05:33	14.924758
2015-10-23 12:35:50	20.688466
2015-10-23 12:51:15	25.396306
2015-10-23 13:33:23	21.661800
2015-10-23 13:38:34	23.214093
2015-10-23 13:43:01	3.811189
2015-10-23 14:02:58	37.300056
2015-10-23 14:09:00	23.432521

Date Time	WaterFlow
2015-10-23 01:00:00	18.810791
2015-10-23 02:00:00	43.087025
2015-10-23 03:00:00	37.987705
2015-10-23 04:00:00	36.120379
2015-10-23 05:00:00	31.851259
2015-10-23 06:00:00	28.238090
2015-10-23 07:00:00	9.863582
2015-10-23 08:00:00	26.679610
2015-10-23 09:00:00	55.773785
2015-10-23 10:00:00	54.156889
2015-10-23 11:00:00	68.374904
2015-10-23 12:00:00	55.710359
2015-10-23 13:00:00	56.968260
2015-10-23 14:00:00	17.206276
2015-10-23 15:00:00	35.093275
2015-10-23 16:00:00	44.424928
2015-10-23 17:00:00	57.322408
2015-10-23 18:00:00	37.344924
2015-10-23 19:00:00	11.483011
2015-10-23 20:00:00	32.117940
2015-10-23 21:00:00	49.081861
2015-10-23 22:00:00	49.133546
2015-10-23 23:00:00	42.064648
2015-10-24 00:00:00	58.380027
2015-10-24 01:00:00	53.408031
2015-10-24 02:00:00	42.332775
2015-10-24 03:00:00	45.922190
2015-10-24 04:00:00	32.741215
2015-10-24 05:00:00	47.879252
2015-10-24 06:00:00	47.460516
2015-10-24 07:00:00	52.264966
2015-10-24 08:00:00	35.389582
2015-10-24 09:00:00	14.435578
2015-10-24 10:00:00	45.038179
2015-10-24 11:00:00	38.896592
2015-10-24 12:00:00	39.991833
2015-10-24 13:00:00	56.883595
2015-10-24 14:00:00	62.728660
2015-10-24 15:00:00	67.382484
2015-10-24 16:00:00	29.645108
2015-10-24 17:00:00	51.586668
2015-10-24 18:00:00	61.987103
2015-10-24 19:00:00	46.394571
2015-10-24 20:00:00	32.838673
2015-10-24 21:00:00	53.416554
2015-10-24 22:00:00	70.723677
2015-10-24 23:00:00	21.570847
2015-10-25 00:00:00	66.762107
2015-10-25 01:00:00	36.322123
2015-10-25 02:00:00	45.114342
2015-10-25 03:00:00	53.473562
2015-10-25 04:00:00	27.209341
2015-10-25 05:00:00	44.193703
2015-10-25 06:00:00	59.296910
2015-10-25 07:00:00	42.253496
2015-10-25 08:00:00	44.747970
2015-10-25 09:00:00	50.182502
2015-10-25 10:00:00	32.232820

pipe1.

```
p1 <- pipe1.data
p1$Date <- ymd_hms(p1$`Date Time`)
p1$Date <- as.Date(p1$Date)

p1 <- p1 %>% group_by(Date) %>% summarise("Pipe1 Records count"=n())

p2 <- pipe2.data
p2$Date <- ymd_hms(p2$`Date Time`)
p2$Date <- as.Date(p2$Date)

p2 <- p2 %>% group_by(Date) %>% summarise("Pipe2 Records count"=n())

merge(p1,p2,by="Date")
```

##	Date	Pipe1 Records count	Pipe2 Records count
## 1	2015-10-23	96	23
## 2	2015-10-24	109	24
## 3	2015-10-25	95	24
## 4	2015-10-26	118	24
## 5	2015-10-27	99	24
## 6	2015-10-28	104	24
## 7	2015-10-29	96	24
## 8	2015-10-30	111	24
## 9	2015-10-31	91	24
## 10	2015-11-01	81	24

It is evident here that pipe1 dataset has an uneven count distribution for given date while pipe2 has even count daily.

Data Cleaning

In this step we will make pipe1 data evenly distributed (per hour) for every day. As mentioned in the problem *for multiple recordings within an hour, take the mean*, we will take mean of records per hour and then recreate the Date Time column. Next we will do the same count comparison that was done earlier and check number of records per day.

```
pipe1.data <- pipe1.data %>%
  mutate(Date=date(`Date Time`), Hour=hour(`Date Time`)) %>%
  group_by(Date, Hour) %>%
  summarise(WaterFlow=mean(WaterFlow)) %>%
  ungroup() %>%
  mutate(`Date Time`=ymd_h(paste(Date, Hour))) %>%
  select(`Date Time`, WaterFlow)
```

`summarise()` has grouped output by 'Date'. You can override using the `.groups` argument.

check the number of records per hour daily

```
p1 <- pipe1.data
p1$Date <- ymd_hms(p1$`Date Time`)
p1$Date <- as.Date(p1$Date)

p1 <- p1 %>% group_by(Date) %>% summarise("Pipe1 Records count"=n())

p2 <- pipe2.data
p2$Date <- ymd_hms(p2$`Date Time`)
```

```
p2$Date <- as.Date(p2$Date)
```

```
p2 <- p2 %>% group_by(Date) %>% summarise("Pipe2 Records count"=n())
```

```
merge(p1,p2,by="Date")
```

```
##           Date Pipe1 Records count Pipe2 Records count
## 1 2015-10-23           24           23
## 2 2015-10-24           24           24
## 3 2015-10-25           24           24
## 4 2015-10-26           24           24
## 5 2015-10-27           23           24
## 6 2015-10-28           23           24
## 7 2015-10-29           24           24
## 8 2015-10-30           24           24
## 9 2015-10-31           24           24
## 10 2015-11-01          22           24
```

Time Series

In this section we will create the timeseries out of both datasets and check out the best model that will eventually provide a week forward forecast.

Pipe1

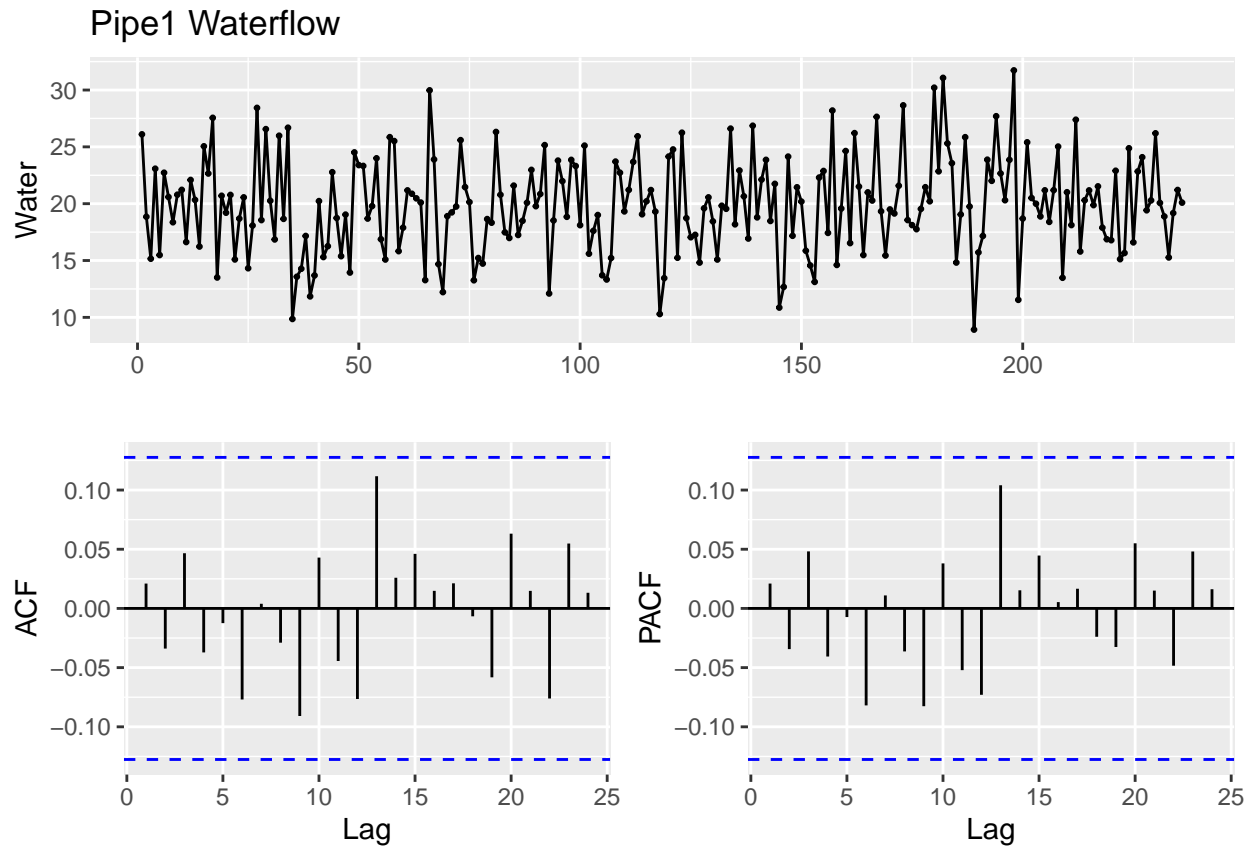
We will start with pipe1 dataset by creating its time series, plot the waterflow data and check the trend and seasonality, if exist.

```
pipe1.ts <- ts(pipe1.data$WaterFlow)
```

```
pipe1.ts %>% summary()
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 8.923 17.033 19.784 19.893 22.789 31.730
```

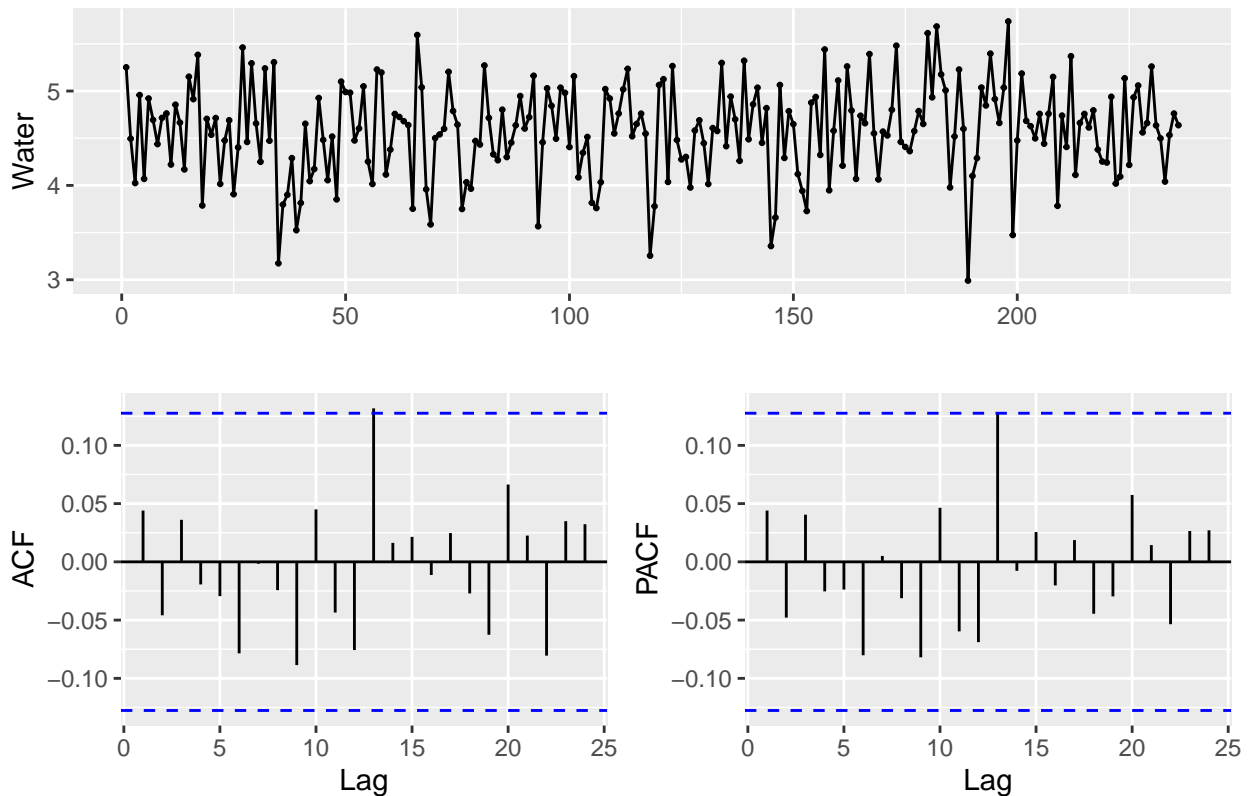
```
ggtsdisplay(pipe1.ts, main="Pipe1 Waterflow", ylab="Water")
```



By seeing the time series, it is apparent there is no seasonality or trend in the data. The ACF and PACF plots shows white noise. It depicts that the time series is stationary and doesn't require differencing.

```
pipe1.lambda <- BoxCox.lambda(pipe1.ts)
pipe1.ts.bc <- BoxCox(pipe1.ts, pipe1.lambda )
ggtsdisplay(pipe1.ts.bc, main=paste("Pipe1 Waterflow",round(pipe1.lambda, 3)), ylab="Water")
```

Pipe1 Waterflow 0.272



```
# Number of differences required for a stationary series
ndiffs(pipe1.ts.bc)
```

```
## [1] 0
```

```
pipe1.ts.bc %>% ur.kpss() %>% summary()
```

```
##
## #####
## # KPSS Unit Root Test #
## #####
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.2346
##
## Critical value for a significance level of:
##          10pct  5pct  2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary.

```
# ets
pipe1.ets.model <- pipe1.ts %>% ets(lambda = pipe1.lambda)
pipe1.ets.model
```

```
## ETS(A,N,N)
##
```



```
## Call:
## ets(y = ., lambda = pipe1.lambda)
##
## Box-Cox transformation: lambda= 0.272
##
## Smoothing parameters:
##   alpha = 1e-04
##
## Initial states:
##   l = 4.5771
##
## sigma: 0.4936
##
##      AIC      AICc      BIC
## 960.1697 960.2731 970.5612
```

We can see here that the ets model that best describes the data is ETS(A,N,N) i.e. exponential smoothing with additive error, no trend and no seasonality.

Next we will find the best Arima model that fits this time series data.

```
# arima
pipe1.arima.model <- pipe1.ts %>% auto.arima(lambda = pipe1.lambda)
pipe1.arima.model
```

```
## Series: .
## ARIMA(0,0,0) with non-zero mean
## Box Cox transformation: lambda= 0.271971
##
## Coefficients:
##      mean
##      4.5771
## s.e. 0.0320
##
## sigma^2 estimated as 0.2425: log likelihood=-167.21
## AIC=338.42 AICc=338.47 BIC=345.35
```

The best Arima model comes out is ARIMA(0,0,0) with non-zero mean.

Next is to plot the forecasts using both the models described above: ets and arima. For this, we will create a generic function which will accept the time series and pipe number and plot the forecast using both the models.

```
# function to plot forecast(s)
pipe.forecast <- function(timeseries, pipe_num) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)

  # models for forecast
  ets.model <- timeseries %>% ets(lambda = lambda)
  arima.model <- timeseries %>% auto.arima(lambda = lambda)

  # forecast h=24*7=168
  pipe.ets.fcst <- forecast(ets.model, h=168)
  #print(pipe.ets.fcst$mean)
  pipe.arima.fcst <- forecast(arima.model, h=168)
  #print(pipe.arima.fcst$mean)
```

```

# plot forecasts
p1 <- autoplot(timeseries) +
  autolayer(pipe.ets.fcst, PI=FALSE, series="ETS") +
  autolayer(pipe.arima.fcst, PI=FALSE, series="ARIMA") +
  theme(legend.position = "top") +
  ylab("Water")

# zoom in plot
if (pipe_num == 1) {
  p2 <- p1 +
    labs(title = "Zoom in ") +
    xlim(c(225,325))
} else {
  p2 <- p1 +
    labs(title = "Zoom in ") +
    xlim(c(990,1150))
}

grid.arrange(p1,p2,ncol=1)
}

```

Lets plot the forecast for pipe1. The forecast is for every hour in a day for a week.

```

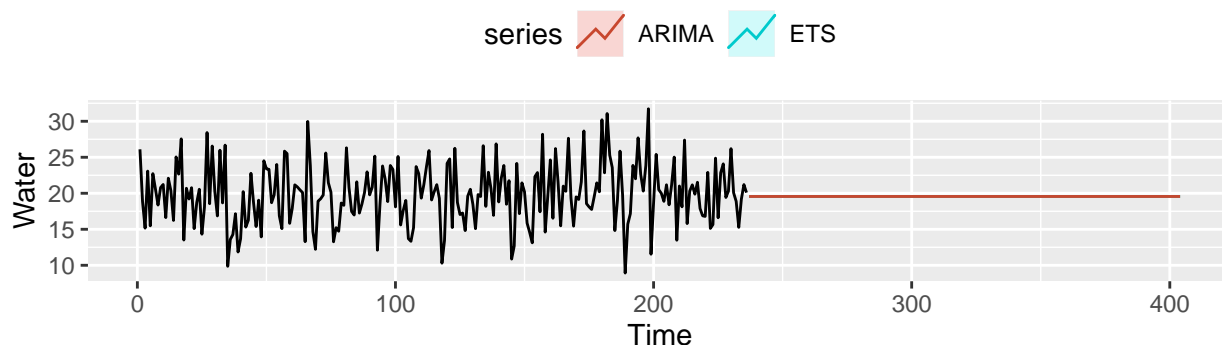
pipe1.ets.fcst <- forecast(pipe1.ets.model, h=168)
pipe1.arima.fcst <- forecast(pipe1.arima.model, h=168)
pipe.forecast(pipe1.ts, 1)

```

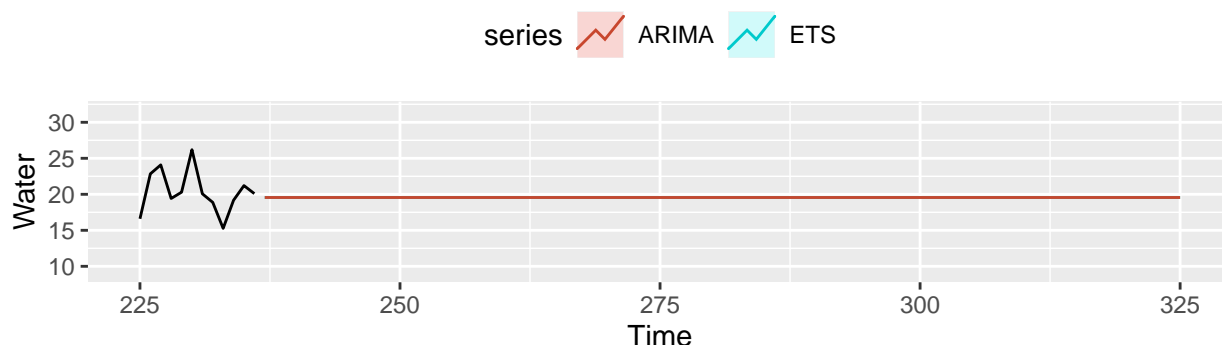
```

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

```



Zoom in



We can see that forecasts for both ETS and ARIMA models are almost on top of each other and for pipe1, the waterflow is forecasted to be 19.5548.

Now we will check the accuracy of both the models ETS and ARIMA. Again for this purpose, we have created a function that accepts the timeseries and a timebreak (for train and test data). In this function we will first divide the data for training and testing, train both models with train set and then find out RMSE using test data.

```
pipe_accuracy <- function(timeseries, time_break) {
  # lambda value
  lambda <- BoxCox.lambda(timeseries)

  # split the data to train and test
  train <- window(timeseries, end=time_break)
  test <- window(timeseries, start=time_break+1)

  # models for forecast
  ets.model <- train %>% ets(model="ANN", lambda = lambda, biasadj = TRUE)
  arima.model <- train %>% Arima(order=c(0,0,0),
                                lambda = lambda,
                                biasadj = TRUE)

  # forecast
  ets.frct = forecast(ets.model, h = length(test))$mean
  arima.frct = forecast(arima.model, h = length(test))$mean

  # dataframe having rmse
  rmse = data.frame(RMSE=cbind(accuracy(ets.frct, test)[,2],
                                accuracy(arima.frct, test)[,2]))
  names(rmse) = c("ETS", "ARIMA")
}
```

```
# display rmse
rmse
}
```

By passing multiple time breaks for training and test data, we see below RMSEs.

```
pipe_accuracy(pipe1.ts, 190)
```

```
##          ETS      ARIMA
## 1 3.950938 3.951593
```

```
pipe_accuracy(pipe1.ts, 178)
```

```
##          ETS      ARIMA
## 1 4.585535 4.585881
```

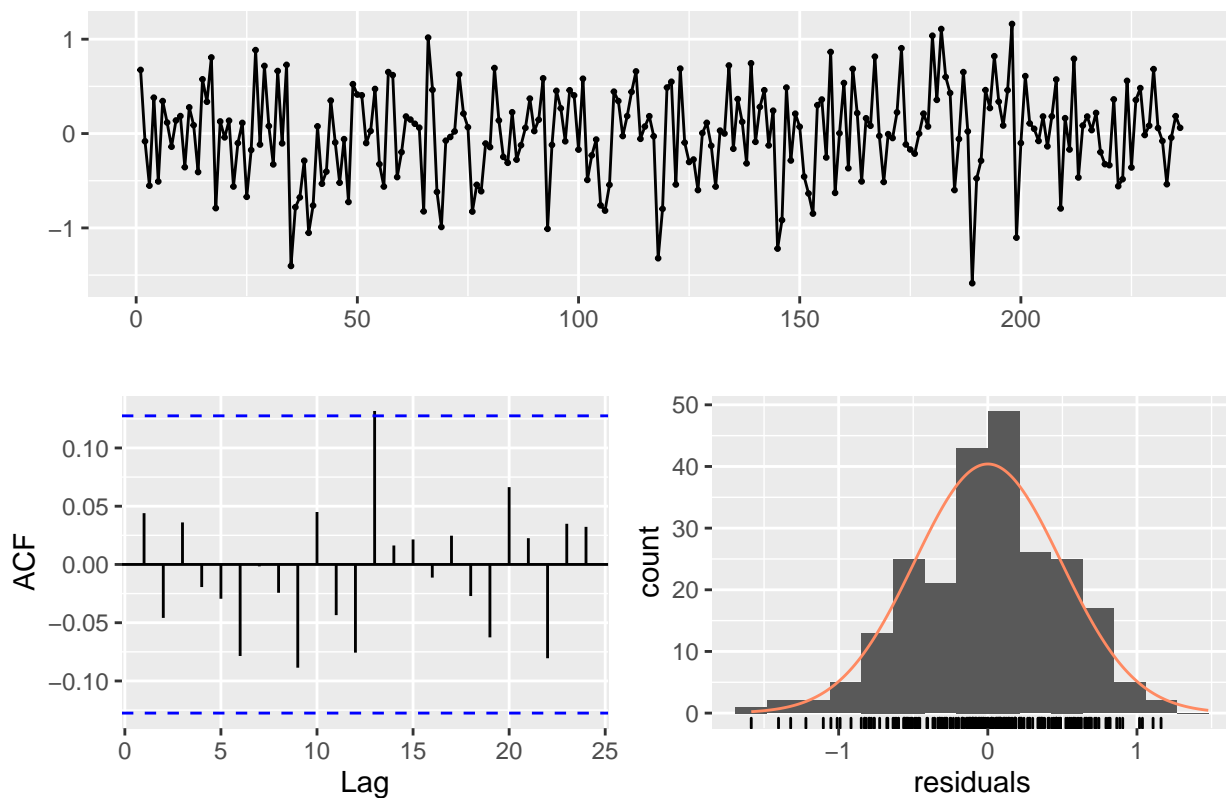
```
pipe_accuracy(pipe1.ts, 200)
```

```
##          ETS      ARIMA
## 1 3.275964 3.276062
```

It is evident here that the model best describes the pipe1 time series is **ETS(A,N,N)** with **Box-Cox transformation of 0.272**.

```
checkresiduals(pipe1.ets.model)
```

Residuals from ETS(A,N,N)



```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,N,N)
```

```
## Q* = 5.684, df = 8, p-value = 0.6826
##
## Model df: 2.    Total lags used: 10
```

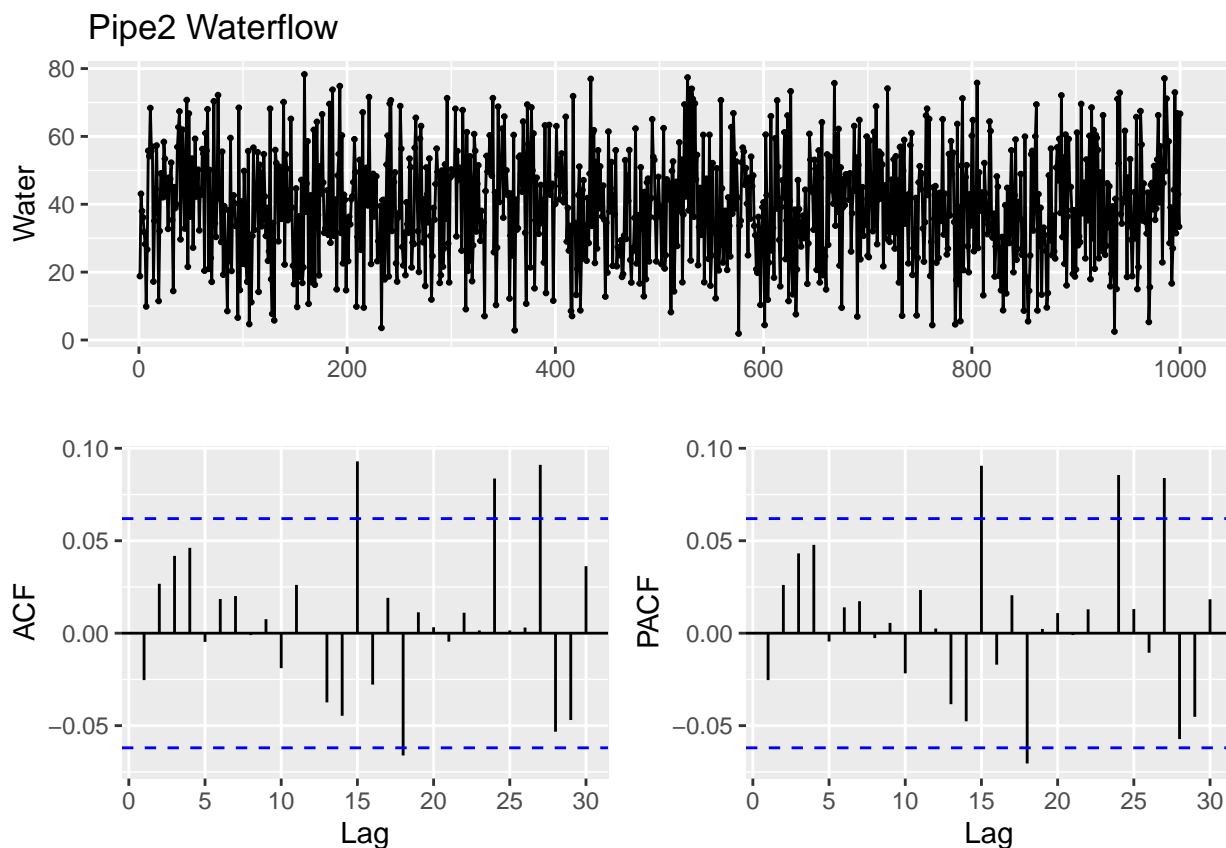
Pipe2

Similar to pipe1, we will create time series for pipe2, plot the waterflow data and check the trend and seasonality, if exist.

```
pipe2.ts <- ts(pipe2.data$WaterFlow)
pipe2.ts %>% summary()
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.885  28.140  39.682  39.556  50.782  78.303
```

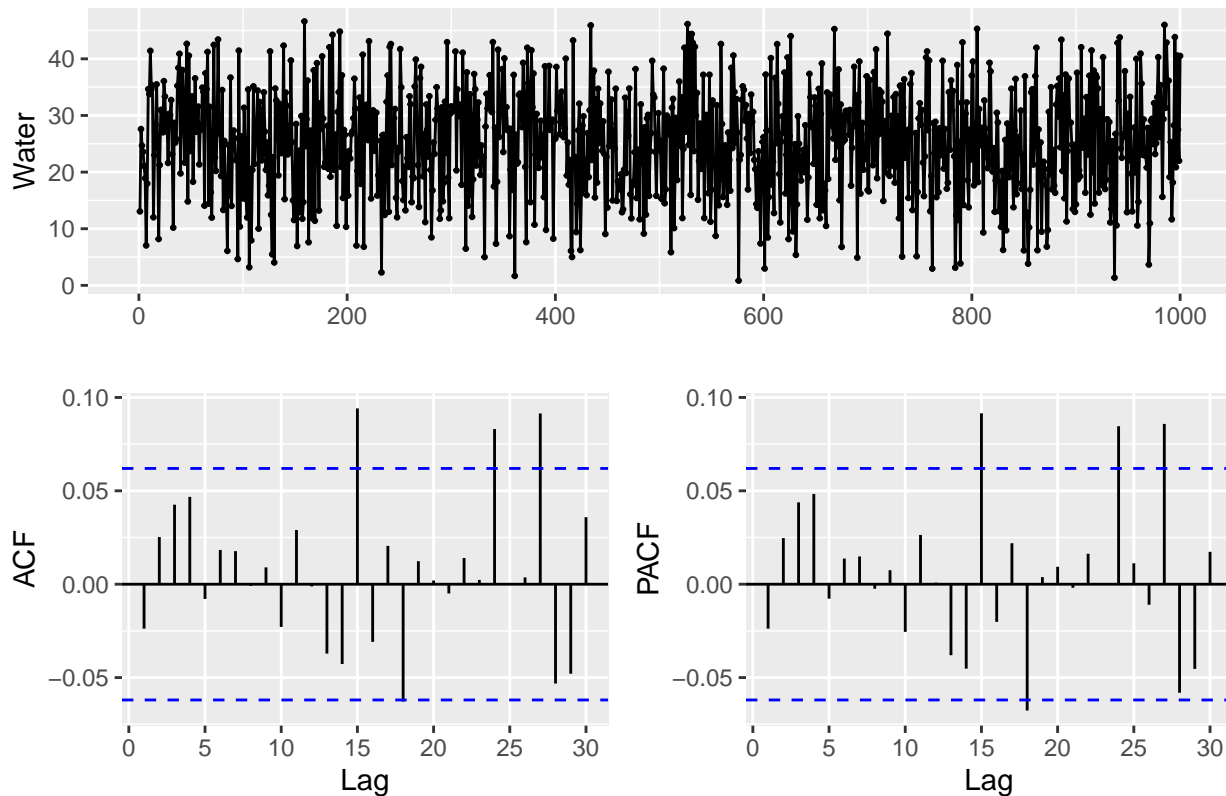
```
ggtsdisplay(pipe2.ts, main="Pipe2 Waterflow", ylab="Water")
```



By seeing the time series, it is apparent there is no seasonality or trend in the data. The ACF and PACF plots shows few significant auto correlations.

```
pipe2.lambda <- BoxCox.lambda(pipe2.ts)
pipe2.ts.bc <- BoxCox(pipe2.ts, pipe2.lambda )
ggtsdisplay(pipe2.ts.bc, main=paste("Pipe2 Waterflow",round(pipe2.lambda, 3)), ylab="Water")
```

Pipe2 Waterflow 0.849



```
# Number of differences required for a stationary series
ndiffs(pipe2.ts.bc)
```

```
## [1] 0
```

```
pipe2.ts.bc %>% ur.kpss() %>% summary()
```

```
##
## #####
## # KPSS Unit Root Test #
## #####
##
## Test is of type: mu with 7 lags.
##
## Value of test-statistic is: 0.1067
##
## Critical value for a significance level of:
##          10pct  5pct  2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

We can see the test statistic small and well within the range we would expect for stationary data. So we can conclude that the data are stationary.

```
# ets
pipe2.ets.model <- pipe2.ts %>% ets(lambda = pipe2.lambda)
pipe2.ets.model
```

```
## ETS(A,N,N)
##
```

```
## Call:
## ets(y = ., lambda = pipe2.lambda)
##
## Box-Cox transformation: lambda= 0.8493
##
## Smoothing parameters:
##   alpha = 1e-04
##
## Initial states:
##   l = 25.2727
##
## sigma: 9.3622
##
##      AIC      AICc      BIC
## 11385.12 11385.14 11399.84
```

We can see here that the ets model that best describes the data is ETS(A,N,N) i.e. exponential smoothing with additive error, no trend and no seasonality.

Next we will find the best Arima model that fits this time series data.

```
# arima
pipe2.arima.model <- pipe2.ts %>% auto.arima(lambda = pipe2.lambda)
pipe2.arima.model
```

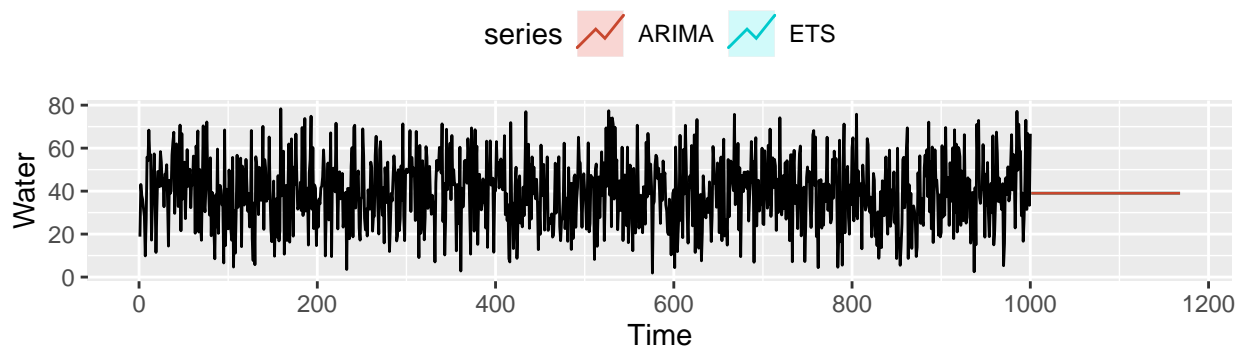
```
## Series: .
## ARIMA(0,0,0) with non-zero mean
## Box Cox transformation: lambda= 0.8493285
##
## Coefficients:
##      mean
##    25.2700
## s.e.    0.2957
##
## sigma^2 estimated as 87.55: log likelihood=-3654.57
## AIC=7313.14 AICc=7313.15 BIC=7322.96
```

The best Arima model for pipe1 comes out is ARIMA(0,0,0) with non-zero mean, similar to pipe1.

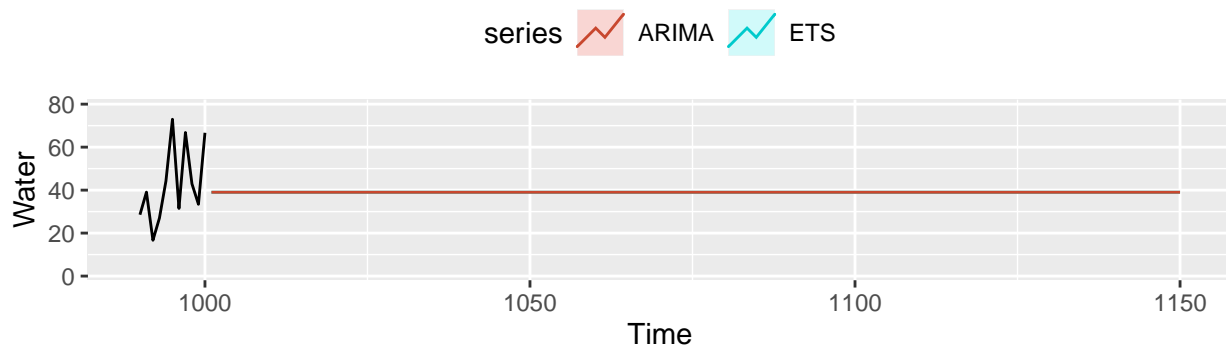
Next we will see the forecast plots using both these models for pipe1.

```
pipe2.ets.fcst <- forecast(pipe2.ets.model, h=168)
pipe2.arima.fcst <- forecast(pipe2.arima.model, h=168)
pipe.forecast(pipe2.ts, 2)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



Zoom in



Similar to pipe1, we can see that forecasts in this case for both ETS and ARIMA models are almost on top of each other and for pipe2, the waterflow is forecasted to be 39.012.

By passing multiple time breaks for training and test data, we see below RMSEs.

```
pipe_accuracy(pipe2.ts, 969)
```

```
##           ETS      ARIMA
## 1 18.36835 18.35826
```

```
pipe_accuracy(pipe2.ts, 850)
```

```
##           ETS      ARIMA
## 1 17.00739 17.00745
```

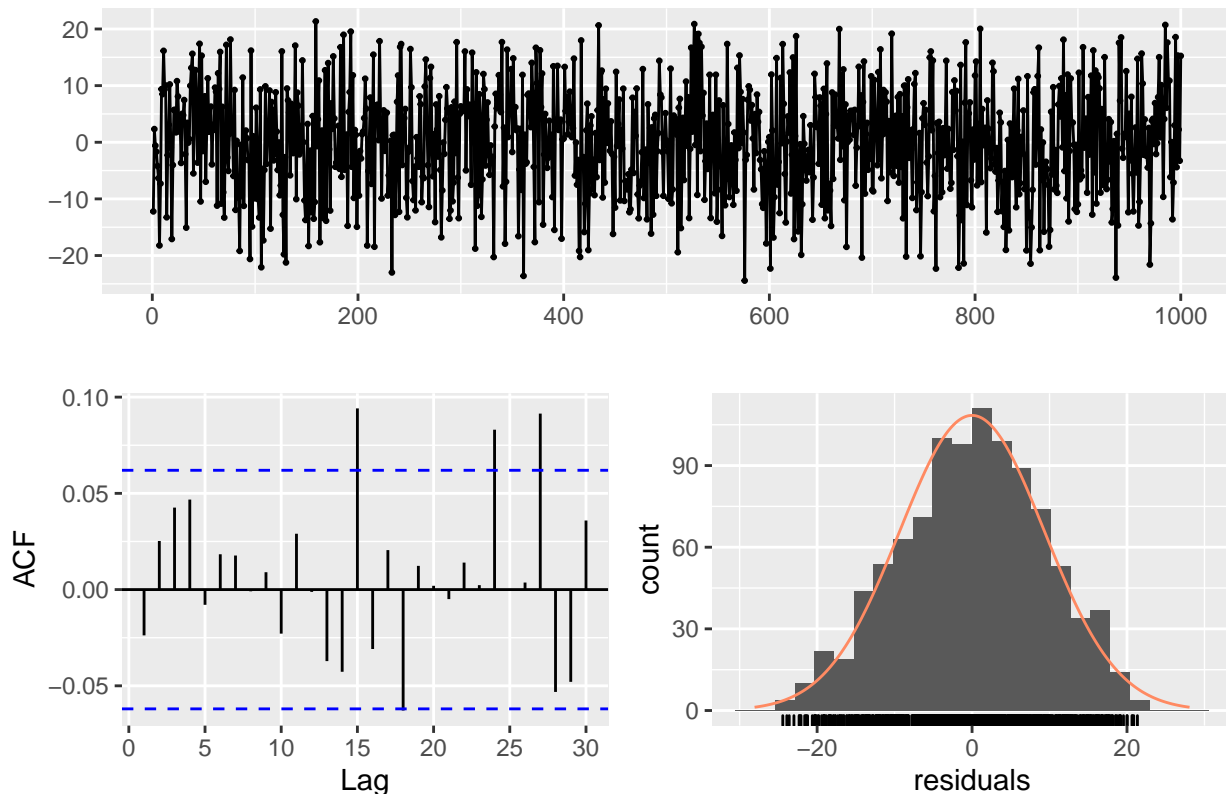
```
pipe_accuracy(pipe2.ts, 877)
```

```
##           ETS      ARIMA
## 1 16.60519 16.60477
```

It is evident here that the model best describes the pipe1 time series is ARIMA(0,0,0) with Box-Cox transformation of 0.849.

```
checkresiduals(pipe2.arima.model)
```


Residuals from ARIMA(0,0,0) with non-zero mean



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,0) with non-zero mean
## Q* = 6.564, df = 9, p-value = 0.6824
##
## Model df: 1.   Total lags used: 10
```

Forecast a week forward

Finally lets do the forecast for both pipes (pipe1 and pipe2) using the above selected best models and save the corresponding xlsx.

```
pipe1.fcst.date <- seq(ymd_hm('2015-11-01 24:00'), ymd_hm('2015-11-08 23:00'), by="hour")
pipe2.fcst.date <- seq(ymd_hm('2015-12-03 17:00'), ymd_hm('2015-12-10 16:00'), by="hour")
```

```
write.xlsx(data.frame('DateTime' = pipe1.fcst.date, 'Waterflow'= pipe1.ets.fcst$mean),
           "Kapoor_data624_pipe1_forecasts.xlsx")
```

```
write.xlsx(data.frame('DateTime' = pipe2.fcst.date, 'Waterflow'= pipe2.arma.fcst$mean),
           "Kapoor_data624_pipe2_forecasts.xlsx")
```

```
pipe1.fcst.ak <- read_excel("Kapoor_data624_pipe1_forecasts.xlsx", skip=0, col_types = c("date","numerical"))
pipe1.fcst.ak %>%
  kbl() %>%
  kable_paper() %>%
  scroll_box(width = "500px", height = "200px")
```

DateTime	Waterflow
2015-11-02 00:00:00	19.55481
2015-11-02 01:00:00	19.55481
2015-11-02 02:00:00	19.55481
2015-11-02 03:00:00	19.55481
2015-11-02 04:00:00	19.55481
2015-11-02 05:00:00	19.55481
2015-11-02 06:00:00	19.55481
2015-11-02 07:00:00	19.55481
2015-11-02 08:00:00	19.55481
2015-11-02 09:00:00	19.55481
2015-11-02 10:00:00	19.55481
2015-11-02 11:00:00	19.55481
2015-11-02 12:00:00	19.55481
2015-11-02 13:00:00	19.55481
2015-11-02 14:00:00	19.55481
2015-11-02 15:00:00	19.55481
2015-11-02 16:00:00	19.55481
2015-11-02 17:00:00	19.55481
2015-11-02 18:00:00	19.55481
2015-11-02 19:00:00	19.55481
2015-11-02 20:00:00	19.55481
2015-11-02 21:00:00	19.55481
2015-11-02 22:00:00	19.55481
2015-11-02 23:00:00	19.55481
2015-11-03 00:00:00	19.55481
2015-11-03 01:00:00	19.55481
2015-11-03 02:00:00	19.55481
2015-11-03 03:00:00	19.55481
2015-11-03 04:00:00	19.55481
2015-11-03 05:00:00	19.55481
2015-11-03 06:00:00	19.55481
2015-11-03 07:00:00	19.55481
2015-11-03 08:00:00	19.55481
2015-11-03 09:00:00	19.55481
2015-11-03 10:00:00	19.55481
2015-11-03 11:00:00	19.55481
2015-11-03 12:00:00	19.55481
2015-11-03 13:00:00	19.55481
2015-11-03 14:00:00	19.55481
2015-11-03 15:00:00	19.55481
2015-11-03 16:00:00	19.55481
2015-11-03 17:00:00	19.55481
2015-11-03 18:00:00	19.55481
2015-11-03 19:00:00	19.55481
2015-11-03 20:00:00	19.55481
2015-11-03 21:00:00	19.55481
2015-11-03 22:00:00	19.55481
2015-11-03 23:00:00	19.55481
2015-11-04 00:00:00	19.55481
2015-11-04 01:00:00	19.55481
2015-11-04 02:00:00	19.55481
2015-11-04 03:00:00	19.55481
2015-11-04 04:00:00	19.55481
2015-11-04 05:00:00	19.55481
2015-11-04 06:00:00	19.55481
2015-11-04 07:00:00	19.55481
2015-11-04 08:00:00	19.55481
2015-11-04 09:00:00	19.55481

```
pipe2.fcst.ak <- read_excel("Kapoor_data624_pipe2_forecasts.xlsx", skip=0, col_types = c("date","numerical"))
pipe2.fcst.ak %>%
  kbl() %>%
  kable_paper() %>%
  scroll_box(width = "500px", height = "200px")
```

DateTime	Waterflow
2015-12-03 17:00:00	39.01295
2015-12-03 18:00:00	39.01295
2015-12-03 19:00:00	39.01295
2015-12-03 20:00:00	39.01295
2015-12-03 21:00:00	39.01295
2015-12-03 22:00:00	39.01295
2015-12-03 23:00:00	39.01295
2015-12-04 00:00:00	39.01295
2015-12-04 01:00:00	39.01295
2015-12-04 02:00:00	39.01295
2015-12-04 03:00:00	39.01295
2015-12-04 04:00:00	39.01295
2015-12-04 05:00:00	39.01295
2015-12-04 06:00:00	39.01295
2015-12-04 07:00:00	39.01295
2015-12-04 08:00:00	39.01295
2015-12-04 09:00:00	39.01295
2015-12-04 10:00:00	39.01295
2015-12-04 11:00:00	39.01295
2015-12-04 12:00:00	39.01295
2015-12-04 13:00:00	39.01295
2015-12-04 14:00:00	39.01295
2015-12-04 15:00:00	39.01295
2015-12-04 16:00:00	39.01295
2015-12-04 17:00:00	39.01295
2015-12-04 18:00:00	39.01295
2015-12-04 19:00:00	39.01295
2015-12-04 20:00:00	39.01295
2015-12-04 21:00:00	39.01295
2015-12-04 22:00:00	39.01295
2015-12-04 23:00:00	39.01295
2015-12-05 00:00:00	39.01295
2015-12-05 01:00:00	39.01295
2015-12-05 02:00:00	39.01295
2015-12-05 03:00:00	39.01295
2015-12-05 04:00:00	39.01295
2015-12-05 05:00:00	39.01295
2015-12-05 06:00:00	39.01295
2015-12-05 07:00:00	39.01295
2015-12-05 08:00:00	39.01295
2015-12-05 09:00:00	39.01295
2015-12-05 10:00:00	39.01295
2015-12-05 11:00:00	39.01295
2015-12-05 12:00:00	39.01295
2015-12-05 13:00:00	39.01295
2015-12-05 14:00:00	39.01295
2015-12-05 15:00:00	39.01295
2015-12-05 16:00:00	39.01295
2015-12-05 17:00:00	39.01295
2015-12-05 18:00:00	39.01295
2015-12-05 19:00:00	39.01295
2015-12-05 20:00:00	39.01295
2015-12-05 21:00:00	39.01295
2015-12-05 22:00:00	39.01295
2015-12-05 23:00:00	39.01295
2015-12-06 00:00:00	39.01295
2015-12-06 01:00:00	39.01295
2015-12-06 02:00:00	39.01295