---
title: "Project 1"
author: "Jatin Jain"
output:
  pdf_document: default
  html_document: default

DATA 624 - PROEJCT 1

In this project, we are tasked to create models to make forecast on the following three sets of data:

* Daily cash withdrawn data for 4 different ATM machines from 5/1/2009 to 4/30/2010. We are to forecast for May, 2010.
* Monthly residential power usage from January 1998 to December 2013. We are asked to make monthly forecast for 2014.
* Hourly water flow data in two pipes from 10/23/2015 to 12/3/2015. We are asked to make a week of forward forecast.

The project is separated into 3 parts. In each part, we first explore the given time series, clean the data set, and fit the time series with forecast techniques including the following:

* Seasonal and trend decomposition using Loess (STL)
* Exponential smoothing (ETS)
* Auto regressive integrated moving average (ARIMA)
* Bagging exponential smoothing with STL decomposition and Box-Cox transformation

For Part A, technique 1 through 3 are used. For Part B technique 1 through 4 are used. For Part C, we use technique 2 and 3.

We then compare the models, select the best model through cross validation, and generate the final forecast. The final forecast for each time series are attached in the project submission as MS Excel files.

Part A - ATM Forecast

Exploratory Data Analysis

In this given time series, the raw data has 3 columns. The 1st column contains the date. The 2nd column indicates the ATM machine. And the 3rd column is the total cash amount drawn in that day. We first separate the data into 4 time series, one for each of the ATM machine.

Below, the time series are plotted along with their corresponding ACF and PACF.


```{r}
library(tseries)
library(xts)
library(urca)
library(fpp2)
library(kableExtra)
library(Metrics)
library(dplyr)
```

```{r}
atm <- readxl::read_excel("/Users/jatinjain/Desktop/CUNY COURSES/Predictive Analytics/ATM624Data.xlsx")
atm$DATE <- as.Date(atm$DATE, origin = "1899-12-30")
```

```{r}
summary(atm)
```

```{r}
atm1 <- subset(atm, ATM=='ATM1', select=c(DATE, Cash))
atm1 <- xts(atm1$Cash, atm1$DATE)
atm2 <- subset(atm, ATM=='ATM2', select=c(DATE, Cash))
atm2 <- xts(atm2$Cash, atm2$DATE)
atm3 <- subset(atm, ATM=='ATM3', select=c(DATE, Cash))
atm3 <- xts(atm3$Cash, atm3$DATE)
atm4 <- subset(atm, ATM=='ATM4', select=c(DATE, Cash))
atm4 <- xts(atm4$Cash, atm4$DATE)

ggtsdisplay(atm1, main='ATM 1')
```

```{r}
ggtsdisplay(atm2, main='ATM 2')
```

```{r}
ggtsdisplay(atm3, main='ATM 3')
```

```{r}
ggtsdisplay(atm4, main='ATM 4')
```

From these plots, we can make the following observation:

* There are apparent weekly seasonal patterns in ATM 1 and 2.
* For ATM 3, all of the values are zero except for the last 3 days.
* For ATM 4, there is one outlier in the data set.

While plotting, we also notice there are some missing values in the data set. Below is a table identifying the dates of these missing values:

```{r}
atm1.na <- atm1[is.na(atm1)] %>% time()
atm2.na <- atm2[is.na(atm2)] %>% time()
atm3.na <- atm3[is.na(atm3)] %>% time()
atm4.na <- atm4[is.na(atm4)] %>% time()

df <- data.frame('Dates with missing values' = c(
  paste(atm1.na, collapse = ', '),
  paste(atm2.na, collapse = ', '),
  paste(atm3.na, collapse = ', '),
  paste(atm4.na, collapse = ', ')))
row.names(df) <- paste('ATM', seq(1:4), sep='')
kable(df) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width = F)
```

The various descriptive statistics for the 4 time series are calculated below as well, which confirm some of our observations above:

```{r}
stats <- function(df){
  minimum <- round(apply(df, 2, min, na.rm=T),2)
  maximum <- round(apply(df, 2, max, na.rm=T),2)
  quantile1 <- round(apply(df, 2, quantile, 0.25, na.rm=T),2)
  quantile3 <- round(apply(df, 2, quantile, 0.75, na.rm=T),2)
```

```
  medians <- round(apply(df, 2, median, na.rm=T),2)
  means <- round(apply(df, 2, mean, na.rm=T),2)
  stdev <- round(apply(df, 2, sd, na.rm=T),2)
  nas <- apply(apply(df, 2, is.na), 2, sum, na.rm=T)
  temp <- data.frame(means, stdev, minimum, maximum, quantile1, medians, quantile3, nas)
  colnames(temp) <- c('Mean', 'St.Dev', 'Min.', 'Max.', '1st.Qu.', 'Median', '3rd.Qu.', "NA's")
  return(temp)
}

df <- rbind(stats(atm1), stats(atm2), stats(atm3), stats(atm4))
row.names(df) <- paste('ATM', seq(1:4), sep='')
kable(df) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Data Preparation

The outlier in the ATM 4 time series seems to be a very unusual occurrence. It is unlikely for a ATM machine to dispense over 10,000 dollars in a single day. This is likely a machine error or data input error. It only happened once in the whole year, and it is very unlikely to occur again. We remove this outlier in the ATM 4 data, and treat it as a missing value. To impute the missing values in ATM 1, 2, and 4, since there are clear weekly seasonality in these time series, we wish to maintain this characteristic. So we impute these missing values using the mean of the values in the same day of week. Below is a table listing the mean values imputed for the missing values.

```{r}
weekday_mean <- function(ts, weekday){
  temp <- as.data.frame(ts)
  temp[,2] <- weekdays(as.Date(row.names(temp)))
  temp <- temp[temp[,2]==weekday, 1]
  return(c(round(mean(temp, na.rm=T))))
}

atm1[atm1.na[1]] <- atm1.na[1] %>% as.Date() %>% weekdays() %>% weekday_mean(atm1, .)
atm1[atm1.na[2]] <- atm1.na[2] %>% as.Date() %>% weekdays() %>% weekday_mean(atm1, .)
atm1[atm1.na[3]] <- atm1.na[3] %>% as.Date() %>% weekdays() %>% weekday_mean(atm1, .)
atm2[atm2.na[1]] <- atm2.na[1] %>% as.Date() %>% weekdays() %>% weekday_mean(atm2, .)
atm2[atm2.na[2]] <- atm2.na[2] %>% as.Date() %>% weekdays() %>% weekday_mean(atm2, .)
atm4.na <- atm4[atm4==max(atm4)] %>% time()
atm4[atm4.na] <- NA
atm4[atm4.na] <- atm4.na %>% as.Date() %>% weekdays() %>% weekday_mean(atm4, .)

df <- data.frame(Dataset=c(rep('ATM 1', 3), rep('ATM 2', 2), 'ATM 4'),
                 'Day.of.Week'=c(weekdays(atm1.na), weekdays(atm2.na), weekdays(atm4.na)),
                 'Day of Week Mean'=c(atm1[atm1.na[1]], atm1[atm1.na[2]], atm1[atm1.na[3]],
                                      atm2[atm2.na[1]], atm2[atm2.na[2]], atm4[atm4.na]))

kable(df) %>%
  kable_styling( full_width = F, position='center')
```

Below, the ATM 4 time series is plotted again with the outlier replaced. The plot shows that ATM 4 is more active than ATM 1 and 2 - the daily cash withdrawn are generally more than ATM 1 and 2; and there does not seem to be a strong weekly seasonality in ATM 4 as illustrated in its ACF and PACF plots.

```{r}
ggtsdisplay(atm4, main='ATM 4')
```

For ATM 3, as mentioned, it only has 3 days worth of data (non-zero). This is not enough to make a forecast on its own. This maybe a new ATM machine that was just starting to disperse money on 4/28/2010. It is reasonable to make forecast for ATM 3 using the forecast of ATM 1, 2 and 4 by taking the average.

Time Series Models

STL Decomposition

We fit the data of ATM 1, 2, and 4 using the stl() function in R. From the ACF and PACF of the time series, it is apparent that the period is 7 for weekly seasonality (s.window=7). Below are the plots for the STL decomposition.

```{r}
stl.atm1 <- atm1 %>% as.vector() %>% ts(frequency = 7) %>% stl(s.window=7, robust=T)
stl.atm2 <- atm2 %>% as.vector() %>% ts(frequency = 7) %>% stl(s.window=7, robust=T)
stl.atm4 <- atm4 %>% as.vector() %>% ts(frequency = 7) %>% stl(s.window=7, robust=T)

autoplot(stl.atm1) + ggtitle('ATM 1')
```

```{r}
autoplot(stl.atm2) + ggtitle('ATM 2')
```

```{r}
autoplot(stl.atm4) + ggtitle('ATM 4')
```

Exponential Smoothing

The ets() function is used to apply exponential smoothing to the time series, with the following settings for the parameters:

* model: This parameter is set to default ('ZZZ') to allow the function to look for the best model using information criterion.
* damped: Set to default (NULL) so that the model selects it based on information criterion.
* lambda: We test both turning this on and off. When this parameter is turned on, the BoxCox.lambda() is used to find the best lambda parameter.
* biasadj: This parameter is turned on whenever Box-Cox Transformation is used.

Below are the exponential smoothing models fitted for ATM 1, 2, and 4, respectively:

```{r}
ets.fit <- function(ts){
  ts.fit <- ts %>% as.vector() %>% ts(frequency = 7)
  aic <- Inf

  for (bc in c(FALSE, TRUE)){
    if (bc == TRUE){
      fit <- ets(ts.fit, lambda=BoxCox.lambda(ts), biasadj=T)
    } else {
      fit <- ets(ts.fit)
    }
    if (fit$aic < aic){
      aic <- fit$aic
      best.fit <- fit
    }
  }
  return(best.fit)
}

(ets.atm1 <- ets.fit(atm1))
```

```{r}
(ets.atm2 <- ets.fit(atm2))
```

```

```{r}
(ets.atm4 <- ets.fit(atm4))
```

As you can see, for all 3 time series, the ETS(A,N,A) model best fits the data - with additive error, no trend component, and additive seasonality. For ATM 2 and 4, Box-Cox transform is first used to transform the time series for a better fit than just using the raw time series.

ARIMA Models

The auto.arima() function is used to automatically select models based on the information criterion, with the following adjustments to the parameters:

* stepwise: This parameter is set to FALSE to allow wider range of models to be searched by the function.
* approximation: This parameter is set to FALSE to disallow approximation for accurate modeling.
* seasonal: Both TRUE and FALSE are tested. Selection is based on information criterion.
* lambda: This parameter is tested to see if it would improve the information criterion. The BoxCox.lambda() function is used to find the best lambda for the time series, before passing it to the lambda parameter for the auto.arima() function.
* biasadj: This parameter is set to TRUE, if Box-Cox transform is used.

Below are the ARIMA models fitted for ATM 1, 2, and 4, respectively:

```{r}
arima.fit <- function(ts){
  bc.lambda <- BoxCox.lambda(ts)
  ts <- ts %>% as.vector() %>% ts(frequency = 7)
  aic <- Inf
  for (s in c(FALSE, TRUE)){
    for (bc in c(FALSE, TRUE)){
      if (bc == TRUE){
        fit <- auto.arima(ts, seasonal=s, lambda=bc.lambda, biasadj=T, stepwise=F, approximation=F)
      } else {
        fit <- auto.arima(ts, seasonal=s, stepwise=F, approximation=F)
      }
      if (fit$aic < aic){
        aic <- fit$aic
        best.fit <- fit
      }
    }
  }
  return(best.fit)
}

(arima.atm1 <- arima.fit(atm1))
```

```{r}
(arima.atm2 <- arima.fit(atm2))
```

```{r}
(arima.atm4 <- arima.fit(atm4))
```

Here, it appears that our custom function has determine that for all three time series, seasonal ARIMA models produce the better fit than non-seasonal ARIMA:

* For ATM 1, the model selected is ARIMA(0,0,1)(1,1,1)[7].
* For ATM 2, the model selected is ARIMA(2,0,2)(0,1,1)[7], with Box-Cox transformation.
* For ATM 4, the model selected is ARIMA(0,0,0)(2,0,0)[7] with non-zero mean and Box-Cox transformation.
```

Below, we check the residuals of the ARIMA models for ATM 1, 2, and 4, respectively:

```{r}
checkresiduals(arima.atm1)
```

```{r}
checkresiduals(arima.atm2)
```

```{r}
checkresiduals(arima.atm4)
```

It appears that the residuals for the 3 ARIMA models are indeed white noises. For the models built for ATM 1 and 2, the ACF plots show that there are no longer significant auto-correlation left in the residuals, and the histograms show that the residuals are by and large normal. For ATM 4 model, the residuals appears to be slightly left skewed. However, we do not think it is a concern, since the Ljung-Box test still returns a p-value of higher than 5%, which means that the null hypothesis (the residuals are independently distributed) cannot be rejected.

Model Comparison and Selection

Below, we plot the 31-day forecasts for ATM 1, 2, and 4, using the models built above.

```{r}
plot.models <- function(ts, ts.name, stl.model, ets.model, arima.model){
  ts <- ts %>% as.vector %>% ts()
  stl.fc <- forecast(stl.model, h=31)$mean %>% ts(start=366, end=396)
  ets.fc <- forecast(ets.model, h=31)$mean %>% ts(start=366, end=396)
  arima.fc <- forecast(arima.model, h=31)$mean %>% ts(start=366, end=396)

 autoplot(ts) +
    autolayer(stl.fc, series='STL Decomposition', PI=FALSE) +
    autolayer(ets.fc, series='Exponential Smoothing', PI=FALSE) +
    autolayer(arima.fc, series='ARIMA Model', PI=FALSE) +
    ggtitle(paste('May 2010 Forecast for',ts.name)) +
    xlab('Day') + ylab('Dollar') +
    guides(colour=guide_legend(title="Forecasting Models")) +
    theme(legend.position = c(0.85,0.9))
}

plot.models(atm1, 'ATM 1', stl.atm1, ets.atm1, arima.atm1)
```

```{r}
plot.models(atm2, 'ATM 2', stl.atm2, ets.atm2, arima.atm2)
```

```{r}
plot.models(atm4, 'ATM 4', stl.atm4, ets.atm4, arima.atm4)
```

To select the best model for each ATM, the information criterion can no longer be used, since they are calculated differently for the different classes of models. Instead, we perform a time series cross validation split of the data, using last 3 month of the time series to perform the cross validation:

* Build model using data from 2009-05-01 to 2010-01-31, and test model using data from 2010-02-01 to 2010-02-28.
* Build model using data from 2009-05-01 to 2010-02-28, and test model using data from 2010-03-01 to 2010-03-31.
* Build model using data from 2009-05-01 to 2010-03-31, and test model using data from 2010-04-01 to 2010-04-30.

The root mean square error (RMSE) is the criterion used to select the model. We simply average the RMSE calculated from the 3 test sets. Below is a table of the RMSE calculated from the cross validation:

```{r}
tscv.split <- function(ts, year, month){
  # This function splits the time series into train-test sets.
  train <- ts[paste('/', year, '-', month, sep='')] %>% as.vector() %>% ts(frequency = 7)
  test <-  ts[paste(year, '-', month+1, sep='')] %>% as.vector() %>% ts(frequency = 7)
  return(list(train, test))
}

tscv.atm <- function(ts, atm){
  # This function performs the time series cross validation, using last 3 months as the validation sets.
  stl.rmse <- c()
  ets.rmse <- c()
  arima.rmse <- c()
  for (m in c(1,2,3)){
    temp <- tscv.split(ts, 2010, m)
    train <- temp[[1]]
    test <- temp[[2]] %>% as.vector()

    stl.fit <- stl(train, s.window=7, robust=T)
    if (atm==1){
      ets.fit <- ets(train, model='ANA')
      arima.fit <- Arima(train, order=c(0,0,1), seasonal=c(1,1,1))
    }
    if (atm==2){
      ets.fit <- ets(train, model='ANA', lambda = ets.atm2$lambda, biasadj = T)
      arima.fit <- Arima(train, order=c(2,0,2), seasonal=c(0,1,1), lambda = arima.atm2$lambda, biasadj = T)
    }
    if (atm==4){
      ets.fit <- ets(train, model='ANA', lambda = ets.atm4$lambda, biasadj = T)
      arima.fit <- Arima(train, order=c(0,0,0), seasonal=c(2,0,0), lambda = arima.atm4$lambda, biasadj = T)
    }

    stl.fc <- forecast(stl.fit, h=length(test))$mean %>% as.vector()
    ets.fc <- forecast(ets.fit, h=length(test))$mean %>% as.vector()
    arima.fc <- forecast(arima.fit, h=length(test))$mean %>% as.vector()

    stl.rmse[m] <- rmse(stl.fc, test)
    ets.rmse[m] <- rmse(ets.fc, test)
    arima.rmse[m] <- rmse(arima.fc, test)

  }
  temp.mean <- round(c(mean(stl.rmse), mean(ets.rmse), mean(arima.rmse)), 2)
  temp.sd <- round(c(sd(stl.rmse), sd(ets.rmse), sd(arima.rmse)), 2)
  return(list(temp.mean, temp.sd))
}

tscv.atm1 <- tscv.atm(atm1, 1)
tscv.atm2 <- tscv.atm(atm2, 2)
tscv.atm4 <- tscv.atm(atm4, 4)
df <- data.frame(tscv.atm1[[1]], tscv.atm1[[2]],
                 tscv.atm2[[1]], tscv.atm2[[2]],
                 tscv.atm4[[1]], tscv.atm4[[2]])
names(df) <- rep(c('Mean', 'Std.Dev'), 3)
row.names(df) <- c('STL', 'ETS', 'ARIMA')
kable(df) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width = F) %>%
  add_header_above(c(' ', 'ATM 1' = 2, 'ATM 2' = 2, 'ATM 4' = 2))
```

The table suggests that:

* The best model for ATM 1 is ARIMA(0,0,1)(1,1,1)[7].
* The best model for ATM 2 is Exponential Smoothing with Box-Cox transformation (lambda=0.1845).
* The best model for ATM 4 is ARIMA(0,0,0)(2,0,0)[7] with non-zero mean and Box-Cox transformation (lambda=0.2392).

Lastly, for ATM 3, given the limited information (there are only 3 days worth of data), we use the daily average of ATM 1, 2, and 4 to make forecast for ATM 3.

```{r}
atm1.fc <- forecast(arima.atm1, h=31)$mean %>% as.vector()
atm2.fc <- forecast(ets.atm2, h=31)$mean %>% as.vector()
atm4.fc <- forecast(arima.atm4, h=31)$mean %>% as.vector()
atm3.fc <- apply(cbind(atm1.fc, atm2.fc, atm4.fc), 1, mean)


c1 <- seq(as.Date('2010-05-01'), as.Date('2010-05-31'), 1)
c2 <- c(rep('ATM1', 31), rep('ATM2', 31), rep('ATM3', 31), rep('ATM4', 31))
c3 <- c(atm1.fc, atm2.fc, atm3.fc, atm4.fc)

##df <- data.frame('DATE'=c1, 'ATM'=c2, 'Cash'=c3)
##openxlsx::write.xlsx(df, "Desktop/CUNY COURSES/Predictive Analytics/ATM_forecasts.csv")


atm3 <- atm3 %>% as.vector() %>% ts()
atm3.fc <- ts(atm3.fc, start=366, end=396)
autoplot(atm3) +
  autolayer(atm3.fc, series='Average of ATM 1, 2, and 4') +
  ggtitle(paste('May 2010 Forecast for ATM 3')) +
  xlab('Day') + ylab('Dollar') +
  guides(colour=guide_legend(title="Forecasting Models"))
```

Part B - Power Usage Forecast

Data Exploration and Preparation

The raw data set has 192 rows and 3 columns. The 1st column is the index column, which will not be used in this project. The 2nd column is the year and month of the time series, and the 3rd column is the power usage in KWH. Below is a plot of the time series, from which we can make the following observation:

* There is one missing value in the time series, evidenced by a small gap between 2008 and 2009.
* There is one outlier in the timer series, a large dip in the power usage in 2010.
* The time series exhibits a slight upward trend characteristic.
* The time series exhibits a seasonal characteristics. The ACF suggests a 6-month seasonal cycle.

```{r}
power.df <- readxl::read_excel("/Users/jatinjain/Desktop/CUNY COURSES/Predictive Analytics/Project 1/ResidentialCustomerForecastLoad-624.xlsx")
power <- power.df$KWH %>% ts(start=c(1998, 1), frequency = 12)
ggtsdisplay(power, main='Power Usage in KWH')
```

Below are the histogram and the box plot of the time series. From both graphs, we can see that the outlier is way outside of the group where the time series typically is.

```{r}
par(mfrow=c(1,2))
hist(as.numeric(power), main='Histogram of Power Usage', ylab='Power (KWH)', breaks=15)
boxplot(as.numeric(power), main='Boxplot of Power Usage')
```

```

#### The descriptive statistics of the time series are calculated below. The missing value and the outlier (minimum value) are also located:
```{r}
summary(power)
```


```{r}
missing <- which(is.na(power.df$KWH))
outlier <- which(power.df$KWH==min(power.df$KWH, na.rm = T))
paste('A missing value is found on', power.df[missing,]$`YYYY-MMM`, '.')
```

```{r}
paste('A minimum value is found on', power.df[outlier,]$`YYYY-MMM`, '.')
```

The outlier is an order of magnitude smaller than other data points. Normally an extreme data point such as this warrants a deeper look at the root cause - is it an incorrect data entry, due to machine or human error, or is it a genuine spike? This knowledge will help improve the model so that it is better equipped to predict outliers. Since these information are not given in this assignment, we opt to treat the outlier as missing value, and note down that the model can be improved upon a deeper investigation of the outliers.

It is also important to note that even if it is not an error but a genuine spike, considering that an extreme point like this only occurred once in 16 years, the chance of it happening again is remote. By ignoring this outlier, the model built will be better at predicting regular power usage based on the season and trends. The trade off of doing so is that the model will be less able to forecast extreme events such as this in the future. This becomes a business decision based on the cost of missing a spike forecast vs the benefit of better forecasts on regular usage.

To impute the two missing values, this time we experience with the na.interp() function in the forecast package. This function linearly interpolates the missing values in a time series. We also compute the the average of data in the same month, for instance the average of all September data for values missing in September. This simpler method was used in Part A for imputation. Below is a side-by-side comparison of the two methods. As you can see, the values computed are not that far off.


```{r}
power[which(power==min(power, na.rm=T))] <- NA
sep.mean <- mean(power[seq(9, 192, 12)], na.rm=T)
jul.mean <- mean(power[seq(7, 192, 12)], na.rm=T)

power <- na.interp(power)

df <- data.frame(c(sep.mean, power[missing]), c(jul.mean, power[outlier]))
names(df) <- c('2008-Sep', '2010-Jul')
row.names(df) <- c('Average', 'na.interp()')

kable(df) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width = F)
```

Below, the time series is plotted again, with the missing values imputed using na.interp(). The gap and the outlier are now gone, and the ACF plot remains unchanged. The seasonal plot and seasonal sub-series plot are also created. From the seasonal plot, it can be confirmed that the seasonal cycle repeats every 6 month. From the sub-series plot, it can be seen that the upward trend (increasing over the years) is strong for the months from Dec through May, and less apparent from Jun through Nov.


```{r}
ggtsdisplay(power, main='Power Usage in KWH')
```
```

```{r}
ggseasonplot(power)
```

```{r}
ggsubseriesplot(power) +
  ylab("KWH") +
  ggtitle("Seasonal subseries plot: power usage")
```

Traditional Time Series Models

STL Decomposition

The stl() function in R is used to perform a STL decomposition of the time series. This time, instead of choosing the s.window subjectively as was done in Part A, we test a range of values and select the parameter base on 5-fold time series cross validation. The 5 folds are as following:

* Train on data from 1998-Jan to 2008-Dec, test on data from 2009-Jan to 2009-Dec.
* Train on data from 1998-Jan to 2009-Dec, test on data from 2010-Jan to 2010-Dec.
* Train on data from 1998-Jan to 2010-Dec, test on data from 2011-Jan to 2011-Dec.
* Train on data from 1998-Jan to 2011-Dec, test on data from 2012-Jan to 2012-Dec.
* Train on data from 1998-Jan to 2012-Dec, test on data from 2013-Jan to 2013-Dec.

Once a model is trained, the forecast() function with the parameter h set to 12 are used to produce the 12 months forecasts. The forecasts on the test sets are compared with the actual data, and the RMSE are calculated and averaged over the 5 folds. Below, we plot the range of s.window parameters vs the RMSE returned from the cross validation.

```{r}
tscv.split <- function(ts, train.end, test.start, test.end){
  # This function splits the time series based on the input
  train <- window(ts, end=train.end)
  test <-  window(ts, start=test.start, end=test.end)
  return(list(train, test))
}

stl.cv <- function(ts, s.window){
  # This function performas the cross validation
  rmse <- c()
  i <- 0
  for (year in seq(2009, 2013, 1)){
    i <- i + 1
    temp <- tscv.split(ts, c(year-1, 12), c(year, 1), c(year,12))
    train <- temp[[1]]
    test <- temp[[2]]
    fit <- stl(train, s.window=s.window, robust=T)
    fc <- forecast(fit, h=length(test))$mean
    rmse[i] <- rmse(as.vector(fc), as.vector(test))
  }
  return(rmse)
}

s.window.cv <- seq(7,23,2)
rmse.cv <- c()

i <- 0
for (s.window in s.window.cv){
  i <- i + 1
  rmse.cv[i] <- stl.cv(power, s.window=s.window) %>% mean()
}

plot(rmse.cv~s.window.cv, type='l', xlab='s.window', ylab='Average RMSE from CV')
```

```

```{r}
names(rmse.cv) <- s.window.cv
paste('The minimum RMSE is ', round(min(rmse.cv)), ' at s.window=', names(rmse.cv[rmse.cv==min(rmse.cv)]), '.',
sep='')
```

From the plot, it can be seen that setting s.window to 9 produces the optimal cross validation score. The s.window parameter can also be set to a string ‘periodic’. This is tested as well:

```{r}
paste('The average RMSE from cross validation when setting s.window to periodic is:', round(mean(stl.cv(power,
s.window='periodic'))))
```

So s.window=9 is still the best choice. Below, we plot the STL decomposition.

```{r}
stl.power <- stl(power, s.window=9, robust = T)
autoplot(stl.power) + ggtitle('STL Decomposition of the Power Usage Time Series')
```

Exponential Smoothing

Just as in Part A, a function is created to utilize the ets() function from the forecast package to automatically search for the best Exponential Smoothing model. Here, the function tests whether Box Cox transformation will improve the model. Below is the result:

```{r}
ets.fit <- function(ts, bc.lambda= -0.2163697){
  aic <- Inf
  for (bc in c(FALSE, TRUE)){
    if (bc == TRUE){
      fit <- ets(ts, lambda=bc.lambda, biasadj=T)
    } else {
      fit <- ets(ts)
    }
    if (fit$aic < aic){
      aic <- fit$aic
      best.fit <- fit
    }
  }
  return(best.fit)
}

(ets.power <- ets.fit(power))
```

Therefore, the best Exponential Smoothing model is ETS(A,A,A) - additive error, additive trend, and additive seasonality. The model also suggests a Box-Cox transformation with a lambda of -0.2164.

ARIMA Models

####Again, auto.arima() is used to automatically select ARIMA models based on the information criterion. As was done in Part A, a function is created to test whether seasonality and/or Box-Cox transformation will help improve the model.

```{r}

```
arima.fit <- function(ts, bc.lambda=-0.2163697){
  aic <- Inf
  for (s in c(FALSE, TRUE)){
    for (bc in c(FALSE, TRUE)){
      if (bc == TRUE){
        fit <- auto.arima(ts, seasonal=s, lambda=bc.lambda, biasadj=T, stepwise=F, approximation=F)
      } else {
        fit <- auto.arima(ts, seasonal=s, stepwise=F, approximation=F)
      }
      if (fit$aic < aic){
        aic <- fit$aic
        best.fit <- fit
      }
    }
  }
  return(best.fit)
}

(arima.power <- arima.fit(power))
```

Therefore, the best ARIMA model is ARIMA(0,0,3)(0,1,1)[12] with drift and Box-Cox transformation with lambda of -0.2164. The residuals are verified below. Because this ARIMA model has a seasonal component, the first 12 months were seasonally differences. Thus the residuals are nearly constant for the first 12 months. After that, it appears the residuals are indeed white noises. The ACF confirms that there is no longer significant seasonality left to exploit, and the histogram is by and large normally shaped. The Ljung-Box test of the residuals also confirms that the they are independently randomly distributed.

```{r}
checkresiduals(arima.power)
```

Bagging ETS with STL

In this section, we experience with a more advanced time series model technique named “Bagging Exponential Smoothing with STL Decomposition and Box-Cox Transformation”, proposed by Christopher Bergmeir, Rob Hyndman, and Jose Beritez on the International Journal of Forecasting in 2015. The article PDF can be found here. Basically, the technique randomly samples the remainder component of the STL decomposition, and then combine it with the seasonal and trend components to create bootstrapped time series. After many new time series are generated this way, exponential smoothing are done on each time series and forecasts are created. The final forecast is “bagged”; in another word, taking the mean or median of the forecasts. Below are more detail steps:

* The original time series is first Box-Cox transformed.
* STL decomposition is done on the time series, separating it into trend, seasonal, and remainder components.
* A technique called Moving Block Bootstrap is used to resample the remainder component, basically forming a new remainder component.
* The bootstrapped remainder is added back to the trend and seasonal components.
* A reversed Box-Cox transform is done. We now have a new time series.
* Exponential smoothing is done on the new time series, and forecasts are created.
* Repeat the above steps N times, so we have N forecasts. The final forecasts are the median of all forecasts.

The detail of the Moving Block Bootstrap is as following:

* Decide on the block size (i.e. window). For example, the article recommend block size of 24 months for monthly time series data.
* Randomly sample with replacement a block of 24 months from the remainder.
* Repeat the sampling until we have enough data having the same length of the original data.
* Put the blocks together to create the new data. Trim the end of data there is overflow.

We execute the algorithms described above to create three functions:

* mbb() to perform the moving block bootstrap. Block size of 24 are used for default.

* bstrap() to generate new time series. The stl() function is used for the STL decomposition, with s.window set to 9.
* bstrap.fc() to fit all of the new time series generated using the ets.fit() function created above, and returned a bagged forecasts in the end.

```{r}
set.seed(1)

mbb <- function(x, b){
  n <- length(x)
  x.new <- c()
  for (i in 1:ceiling(n/b)){
    idx <- (i-1)*b + 1
    end <- sample(b:n, 1)
    x.new[idx:(idx+b-1)] <- x[(end-b+1):end]
  }
  return(x.new[1:n])
}

bstrap <- function(ts, num.boot, b=24, s.window=9){
  lambda <- BoxCox.lambda(ts)
  ts.bc <- BoxCox(ts, lambda)
  fit <- stl(ts.bc, s.window = s.window, robust = TRUE)
  s <- seasonal(fit)
  t <- trendcycle(fit)
  r <- remainder(fit)
  recon.series <- list()
  recon.series[[1]] <- ts
  for (i in 2:num.boot) {
    boot.sample <- mbb(r, b)
    recon.series.bc <- t + s + boot.sample
    recon.series[[i]] <- InvBoxCox(recon.series.bc, lambda)
  }
   return(recon.series)
}

bstrap.fc <- function(recon.series, h, func=median){
  fc.list <- list()
  for (i in 1:length(recon.series)){
    fit <- ets.fit(recon.series[[i]], bc.lambda=BoxCox.lambda(recon.series[[i]]))
    fc.list[[i]] <- forecast(fit, h=h)$mean
  }
  fc.mat <- do.call('rbind', fc.list)
  return(apply(fc.mat, 2, func))
}
```

We make the 12-month forecast using this technique, using 100 rounds of bootstrapping, and plot them below.

```{r}
new.ts <- bstrap(power, num.boot=100)
fc <- bstrap.fc(new.ts, h=12, median)

fc <- ts(fc, start=c(2014,1), frequency = 12)
autoplot(power) +
  autolayer(fc, series='Forecast') +
  ylab('Power Usage KWH') +
  guides(colour=guide_legend(title="")) +
  theme(legend.position = c(0.85,0.9))
```

As you can see, the forecasts appear to capture the seasonality as well as the trend in the time series.

Model Comparison and Selection

####In this section, we will compare the forecasts from the three traditional models as well as the advanced model (bagged ETS with STL decomposition and Box-Cox transformed). Below, we visualize the 4 forecasts in one plot for comparison:

```{r}
plot.models <- function(ts, stl.model, ets.model, arima.model, adv.fc){
  stl.fc <- forecast(stl.model, h=12)
  ets.fc <- forecast(ets.model, h=12)
  arima.fc <- forecast(arima.model, h=12)
  autoplot(ts) +
    autolayer(stl.fc, series='STL Decomposition', PI=FALSE) +
    autolayer(ets.fc, series='Exponential Smoothing', PI=FALSE) +
    autolayer(arima.fc, series='ARIMA Model', PI=FALSE) +
    autolayer(adv.fc, series='Bagged ETS w. STL') +
    ggtitle(paste('Power Usage Forecasts')) +
    xlab('Time') + ylab('KWH') +
    guides(colour=guide_legend(title="Forecasting Models")) +
    theme(legend.position = c(0.5,0.8))
}

plot.models(power, stl.power, ets.power, arima.power, fc)
```

As you can see the 4 models produced by and large similar forecasts that capture the seasonality and trend. The best model is selected based on time series cross validation. Again, the 5-fold expanding window cross validation is used, the same way described in the STL decomposition above. Below, we tabulate the RMSE calculated in each CV set and their average for each forecast techniques.

```{r}
tscv.split <- function(ts, train.end, test.start, test.end){
  train <- window(ts, end=train.end)
  test <-  window(ts, start=test.start, end=test.end)
  return(list(train, test))
}

power.cv <- function(ts, model){
  rmse <- c()
  i <- 0
  for (year in seq(2009, 2013, 1)){
    i <- i + 1
    temp <- tscv.split(ts, c(year-1, 12), c(year, 1), c(year,12))
    train <- temp[[1]]
    test <- temp[[2]]
    if (model=='stl'){
      fit <- stl(train, s.window=9, robust=T)
    }
    if (model=='ets'){
      fit <- ets(train, model='AAA', lambda = -0.2163697, biasadj = T)
    }
    if (model=='arima'){
      fit <- Arima(train, order=c(0,0,3), seasonal=c(0,1,1), lambda = -0.2163697, biasadj = T, include.drift = T)
    }
    if (model=='adv'){
      new.ts <- bstrap(train, num.boot=100)
    }
    if (model=='adv'){
      fc <- bstrap.fc(new.ts, h=12, median)
    } else{
```

```
      fc <- forecast(fit, h=length(test))$mean
    }
    rmse[i] <- rmse(as.vector(fc), as.vector(test))
  }
  return(rmse)
}

tscv.stl <- power.cv(power, 'stl')
tscv.ets <- power.cv(power, 'ets')
tscv.arima <- power.cv(power, 'arima')
tscv.adv <- power.cv(power, 'adv')
df <- data.frame(c(tscv.stl, mean(tscv.stl)),
                 c(tscv.ets, mean(tscv.ets)),
                 c(tscv.arima, mean(tscv.arima)),
                 c(tscv.adv, mean(tscv.adv))) %>% round()
row.names(df) <- c('2009', '2010', '2011', '2012', '2013', 'Average')
names(df) <- c('STL', 'ETS', 'ARIMA', 'Bagging')
kable(t(df)) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width = F)
```

It appears that the ARIMA model is the best, having the lowest average RMSE across the CV sets. For this time series, the bagged ETS with STL technique beats the traditional ETS model, but cannot beat the STL decomposition forecast model. Below, we plot the final prediction using ARIMA(0,0,3)(0,1,1)[12] with drift.

```{r}
arima.fc <- forecast(arima.power, h=12)
c <- seq(as.Date('2014-01-01'), as.Date('2014-12-01'), by='month') %>% format('%Y-%m')
df <- data.frame('YYYY-MM'=c, 'KWH'=arima.fc$mean)
##openxlsx::write.xlsx(df, "Desktop/CUNY COURSES/Predictive Analytics/Project 1/
power_usage_forecasts.xlsx")

autoplot(power) +
  autolayer(arima.fc, PI=T) +
  theme(legend.position = c(0,0)) +
  ylab('KWH') +
  ggtitle('Power Usage Forecast')
```

Part C - Water Flow Forecast

Data Wrangling and Exploration

Below, the time series of the two water pipes are read into R data frame:

```{r}
pipe1 <- readxl::read_xlsx("/Users/jatinjain/Desktop/CUNY COURSES/Predictive Analytics/Project 1/
Waterflow_Pipe1.xlsx")
pipe2 <- readxl::read_xlsx("/Users/jatinjain/Desktop/CUNY COURSES/Predictive Analytics/Project 1/
Waterflow_Pipe2.xlsx")
pipe1$`Date Time` <- round(as.POSIXlt(pipe1$`Date Time` *60*60*24, origin='1899-12-30', tz='GMT'),
units='secs')
pipe2$`Date Time` <- round(as.POSIXlt(pipe2$`Date Time` *60*60*24, origin='1899-12-30', tz='GMT'),
units='secs')

pipe1
```

```{r}
pipe2
```

Inspecting the first few rows of both time series, it can be seen that Pipe 1 reading was recorded on uneven intervals, while Pipe 2 reading was recorded on even (hourly) intervals. We aggregate the Pipe 1 water flow data based on hour by taking the mean.

```{r}
pipe1$Date <- as.Date(pipe1$`Date Time`)
pipe1$Hour <- format(pipe1$`Date Time`, '%H')
pipe2$Date <- as.Date(pipe2$`Date Time`)
pipe2$Hour <- format(pipe2$`Date Time`, '%H')
pipe1 <- pipe1[,-c(1)]
pipe2 <- pipe2[,-c(1)]

pipe1 %>% group_by(Date, Hour) %>% summarise(MeanFlow=mean(WaterFlow)) -> pipe1

pipe1
```

```{r}
(pipe2 <- pipe2[,c(2,3,1)])
```

The two time series shares the same starting date, but different end date. Pipe 1 recording ends on 2015-11-01 and Pipe 2 ends on 2015-12-03. Therefore, Pipe 2 has a lot more data than Pipe 1.

```{r}
tail(pipe1,1)
```

```{r}
tail(pipe2,1)
```

Below, the two time series are plotted. Both time series appear to be white noises, with no apparent trend or seasonality.

```{r}
p1 <- ts(pipe1$MeanFlow)
p2 <- ts(pipe2$WaterFlow)

ggtsdisplay(p1, main='Pipe 1 Water Flow')
```

```{r}
ggtsdisplay(p2, main='Pipe 2 Water Flow')
```

Time Series Forecast

Without seasonality, we cannot perform stl() decomposition. The forecast techniques available to us are ets() and auto.arima(). Below, we fit the two time series using ets() and auto.arima, using their default parameters.

```{r}
(p1.ets <- ets(p1))
```

```{r}
(p2.ets <- ets(p2))
```

```{r}
(p1.arima <- auto.arima(p1))
```

```

```{r}
(p2.arima <- auto.arima(p2))
```

As can be seen, the exponential smoothing models detect no seasonality and trend in both time series. The model is using the mean of the time series as the initial state, with the smoothing parameter alpha close to zero. This is essentially forecasting using the mean of the past. Likewise, the ARIMA models also detect no seasonality and trend in the two time series; and the best forecast is its mean. Indeed, the mean is the best forecast you can make for seemingly randomly generated data.

Below, we generate the 1-week ahead (168 hours) forecasts and plots based on the ARIMA model:
```{r}
p1.fc <- forecast(p1.arima, h=168)
p2.fc <- forecast(p2.arima, h=168)

date.p1 <- seq(as.POSIXct("2015-11-01 24:00", tz="GMT"), as.POSIXct("2015-11-08 23:00:00", tz="UTC"),
by='hour')
date.p2 <- seq(as.POSIXct("2015-12-03 17:00", tz="GMT"), as.POSIXct("2015-12-10 16:00:00", tz="UTC"),
by='hour')
df1 <- data.frame('Date Time'=date.p1, 'WaterFlow'=p1.fc$mean)
df2 <- data.frame('Date Time'=date.p2, 'WaterFlow'=p2.fc$mean)
##openxlsx::write.xlsx(df1, "Desktop/CUNY COURSES/Predictive Analytics/Project 1/Pipe1_forecasts.xlsx")
##openxlsx::write.xlsx(df2, "Desktop/CUNY COURSES/Predictive Analytics/Project 1/Pipe2_forecasts.xlsx")

autoplot(p1) +
  autolayer(p1.fc, PI=T) +
  theme(legend.position = c(0,0)) +
  ylab('KWH') +
  ggtitle('Pipe 1 Forecast')
```

```{r}
autoplot(p2) +
  autolayer(p2.fc, PI=T) +
  theme(legend.position = c(0,0)) +
  ylab('KWH') +
  ggtitle('Pipe 2 Forecast')

```
```