Machine Learning II: Final Project

Team 14: Hope Braue, Jose Quan Lopez, Yuke Luo, Julia Wilson

4/22/2020

Kaggle: Store Item Demand Forecasting Challenge

Kaggle Link: https://www.kaggle.com/c/demand-forecasting-kernels-only (https://www.kaggle.com/c/demand-forecasting-kernels-only)

Overview

Given five years of ten different stores' sales data, this Kaggle competition asks participants to predict three months of future sales for 50 items at these ten store locations. The objective is to forecast these sales for each store correctly.

We are given files for both a training dataset and a test dataset. The X features of the data include sale date, store ID, item ID, and sales. The sales feature indicates the number of items sold at a particular store on a particular date. Sales data is available for an item for every day of the year (365 days). The date feature includes a month, date, and year. The training data includes sales information for the years 2013 through 2017. The test data includes the sales information that needs to be predicted for the year 2018. Therefore, we will need to forecast for the daily sales for January, February, and March of 2018.

We are also provided with a sample submission file that describes the correct format for submitting our solution. For each id in the test set (a unique id for a particular store-item pair), we should predict the number of sales. The submission file should contain both the item ID and the forecasted sale for that specific item within a 3-month span.

Critique of Existing Kaggle Notebooks

In an effort to review already existing solutions in the Kaggle Notebooks, we have outlined techniques that utilize machine learning as well as other concepts that we have learned in our other classes, such as in Artificial Intelligence and Statistics. Also, it is important to note that given that solutions are created using many different programming languages, we have decided to focus on those solutions that use R for the purposes of this project.

Existing Techniques for Challenge:

- ETS
- Holt-Winters
- ARIMA (Auto-Regressive Integrated Moving Average)
    - A class of model that explains a given time series based on its past values, that is, its own lags and the lagged forecast errors, so that the equation can be used to forecast future values
- Neural Networks
- LGBM (LightGBM)
    - A high-performance gradient boosting framework based on decision tree algorithm

- Prophet model
    - A procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.

Packages in R for Time Series:

- forecast package (arima, exponential smoothing)
- nnfor package (neural networks)
- tsfknn package (knn)

Notebooks Examined:

“EDA+Prophet+MLP Neural Network Forecasting” by Arindam Dutta

This notebook is the highest rated on Kaggle for this particular competition. A reason why this notebook might be rated so highly may be due to its excellent report. It explains time series and its own procedure in such a way that it makes it easy for another to understand the Kaggle problem, time-series, and the data scientist’s solution. The code and report are intertwined making it easy to understand and follow step by step.

Dutta first checks to make sure that there are no missing values in the data (a good practice). There are no missing values. He then performs some exploratory data analysis by checking the structure of the data and by plotting the data. Dutta creates many visuals to get a good sense of the data and any trends in the time series such as histograms, growth by date, store, and item.

The report then briefly outlines the prophet model and its advantages over other time series models. Dutta then creates his prediction using the prophet model.

In Part 2 of this notebook, Dutta uses an artificial neural network to create another sales forecast. He uses the “nnfor” package in R to create his network.

“Introduction to Forecasting” by Bruce Wayne

This notebook uses a variety of modeling techniques. Wayne breaks up his notebook into four main sections: exploratory data analysis, forecasting models, random tree forest model, and prophet model. The forecasting models used include ETS, Holt-Winters, and ARIMA.

Wayne visualizes the data first before creating any models. First, he makes a seasonal naive forecast and then conducts simple exponential smoothing, which does not take into account seasonality. The next models a Holt-Winters forecast which does take seasonality into account. Afterward, Wayne creates forecasts using ETS and ARIMA (adding seasonality and using additive vs multiplicative methods). For his last mode, Wayne uses the Prophet model which he notes was inspired by the other notebooks as many of the notebooks use this model.

“LGBM plus Average Sales” by TI

This notebook uses the tidyverse and lightlgb libraries in R to create its model. The LightLGB package in R is “a fast high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and other machine learning tasks.” One critique of this notebook is its inclusion of only code but no report of its procedure or results. This makes the data scientist’s approach and solution somewhat more difficult to replicate without an explanation of the work done.

Our Forecasting Models

For our project, we have decided to create three models using three different forecasting techniques: **S-ARIMA**, **ETS**, and **KNN**. While other Kaggle notebooks used the first two techniques, we did not find that the ones we examined used KNN (K nearest neightbors) as a technique. We will use R's tsfknn package to create this model. We are particularly interested in how this KNN model will perform. Ultimately, our goal is to compare the results and perfomance of these three forecast models.

First, we import the libraries that we will need.

```
#Import libraries
library(tidyr)
library(dplyr)
library(forecast)
library(tsfknn)
library(data.table)
library(ggplot2)
library(urca)
```

Next, we import the training and testing data provided by Kaggle.

```
#Import data
train <- read.table("train.csv", header=T, sep=',')
test <- read.table("test.csv", header=T, sep=',')
```

In total, we have 10 store locations and 50 items. We break down the data by stores and then by items.

```
store1 <- with(train,train[train$store==1,])
#store2 <- with(train,train[train$store==2,])
#store3 <- with(train,train[train$store==3,])
#store4 <- with(train,train[train$store==4,])
#store5 <- with(train,train[train$store==5,])
#store6 <- with(train,train[train$store==6,])
#store7 <- with(train,train[train$store==7,])
#store8 <- with(train,train[train$store==8,])
#store9 <- with(train,train[train$store==9,])
#store10 <- with(train,train[train$store==10,])

#Create a function to split the table:
split_table <- function(table, col = 'col') table %>% split(., .[, col])
store1_item <- split_table(store1, 'item')
#store2_item <- split_table(store2, 'item')
#store3_item <- split_table(store3, 'item')
#store4_item <- split_table(store4, 'item')
#store5_item <- split_table(store5, 'item')
#store6_item <- split_table(store6, 'item')
#store7_item <- split_table(store7, 'item')
#store8_item <- split_table(store8, 'item')
#store9_item <- split_table(store9, 'item')
#store10_item <- split_table(store10, 'item')

#For example, if you want to access store1_item1, type this in console: store1_item$'1'
store1_item1 <- with(store1, store1[store1$item==1,])

#Setting a test sample for the same store and item as done above
#store1_item1test <- store1_item1[1705:1826,]

#Delete the test data from the training data frame
#store1_item1 <- store1_item1[-c(1705:1826),]
```

Data Overview

Before we begin modeling, we want to get a sense of our data's dimensions and structure.

```
sprintf('The train data set has %d observations and %d features', nrow(train), ncol(train))
```

```
## [1] "The train data set has 913000 observations and 4 features"
```

```
str(train)
```

```
## 'data.frame':    913000 obs. of  4 variables:
##  $ date : Factor w/ 1826 levels "2013-01-01","2013-01-02",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ store: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ item : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ sales: int  13 11 14 13 10 12 10 9 12 9 ...
```

```
sprintf('The test data set has %d observations and %d features', nrow(test), ncol(test))
```

```
## [1] "The test data set has 45000 observations and 4 features"
```

```
str(test)
```

```
## 'data.frame':    45000 obs. of  4 variables:
##  $ id   : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ date : Factor w/ 90 levels "2018-01-01","2018-01-02",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ store: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ item : int  1 1 1 1 1 1 1 1 1 1 ...
```

Let's check to see if the data has any missing values.

```
colSums(is.na(train))
```

```
##  date store  item sales
##     0     0     0     0
```

```
colSums(is.na(test))
```

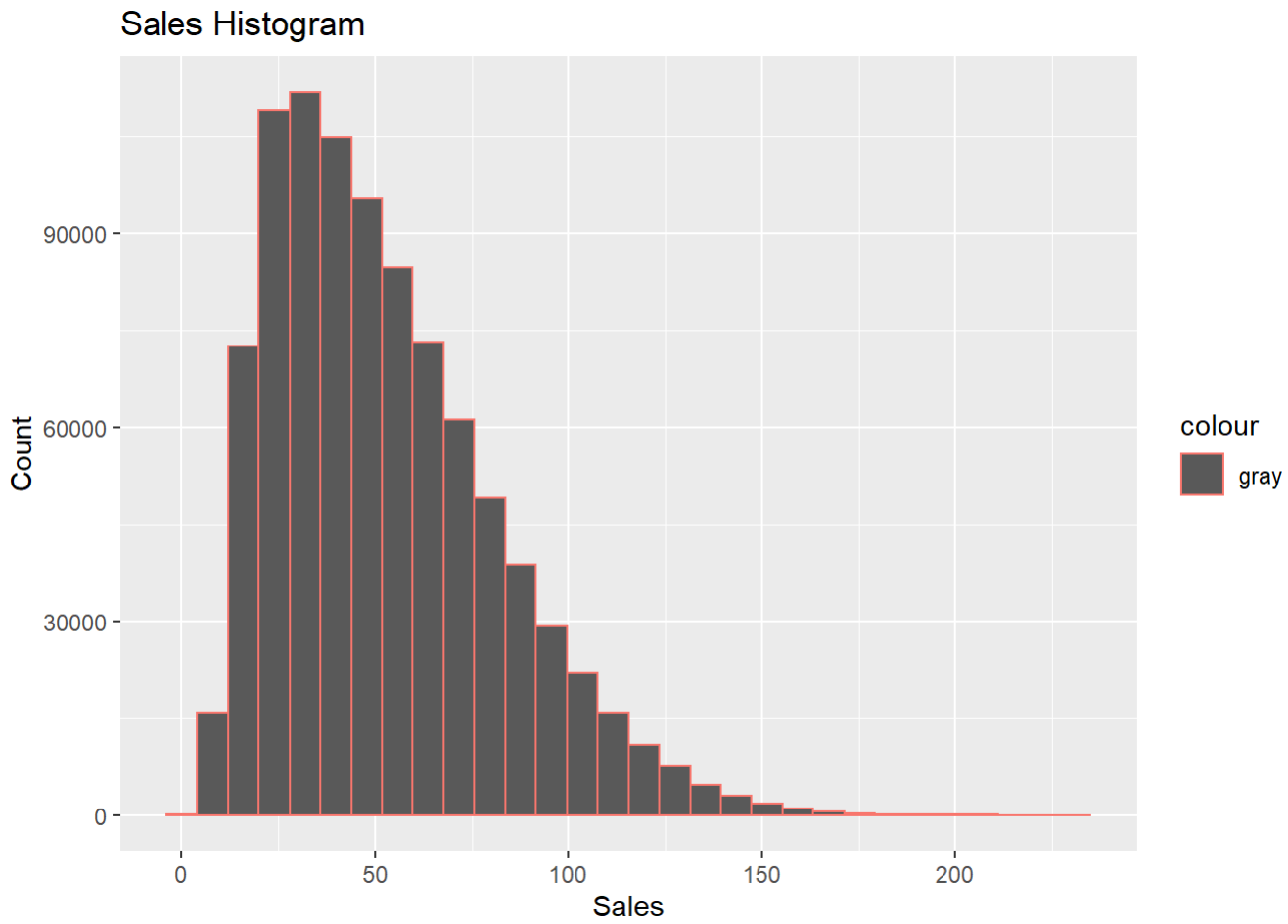
```
##    id  date store  item
##     0     0     0     0
```

The data has no missing values, so no action is needed.

Exploratory Analysis

We will now peform some exploratory analysis.

```
ggplot(train, aes(x=sales, col = 'gray'))+
  geom_histogram()+
  labs(title = "Sales Histogram", x = 'Sales', y = 'Count')
```



From the histogram, we can see a distribution skewed to the left. We see a trend in which the daily sales amount of most items is close to 50.

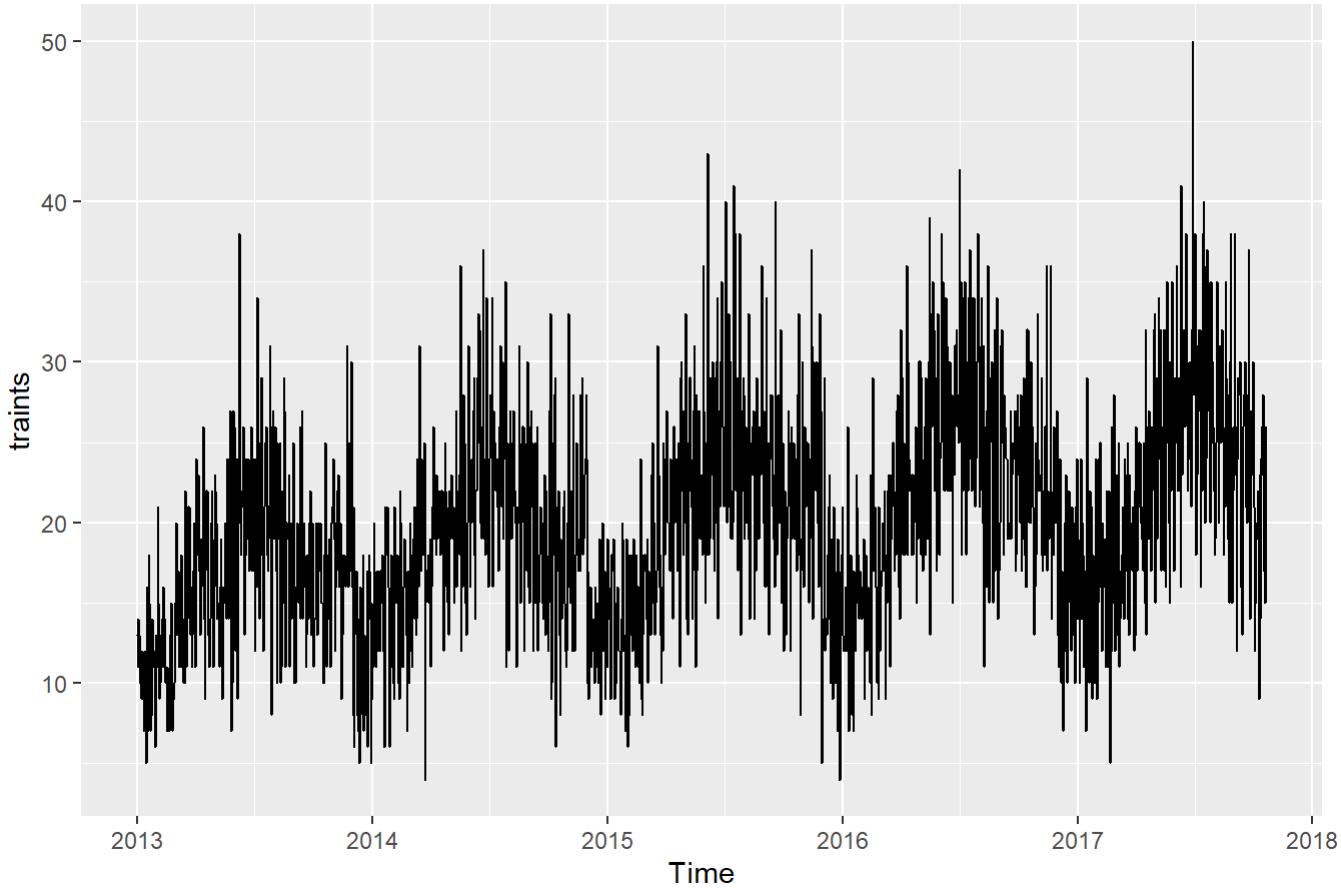
Before forecasting for all items at every store, we will work with a subset sample, so that we can understand the structure and have a better idea of the broader picture. Now, we will create a time-series object specifically for item 1 in store 1. We also create a training and test set.

```
#Time Series; Frequency = 365 because of annual cycle
storets <- ts(store1_item1$sales, frequency = 365, start = c(2013, 1))

# Create test and train series
traints <-window(storets,frequency = 365, end=c(2017.8))
testts <-window(storets,frequency = 365, start=c(2017.8))
```

Continuing with our exploratory analysis, we will use the autoplot function from the forecast package.

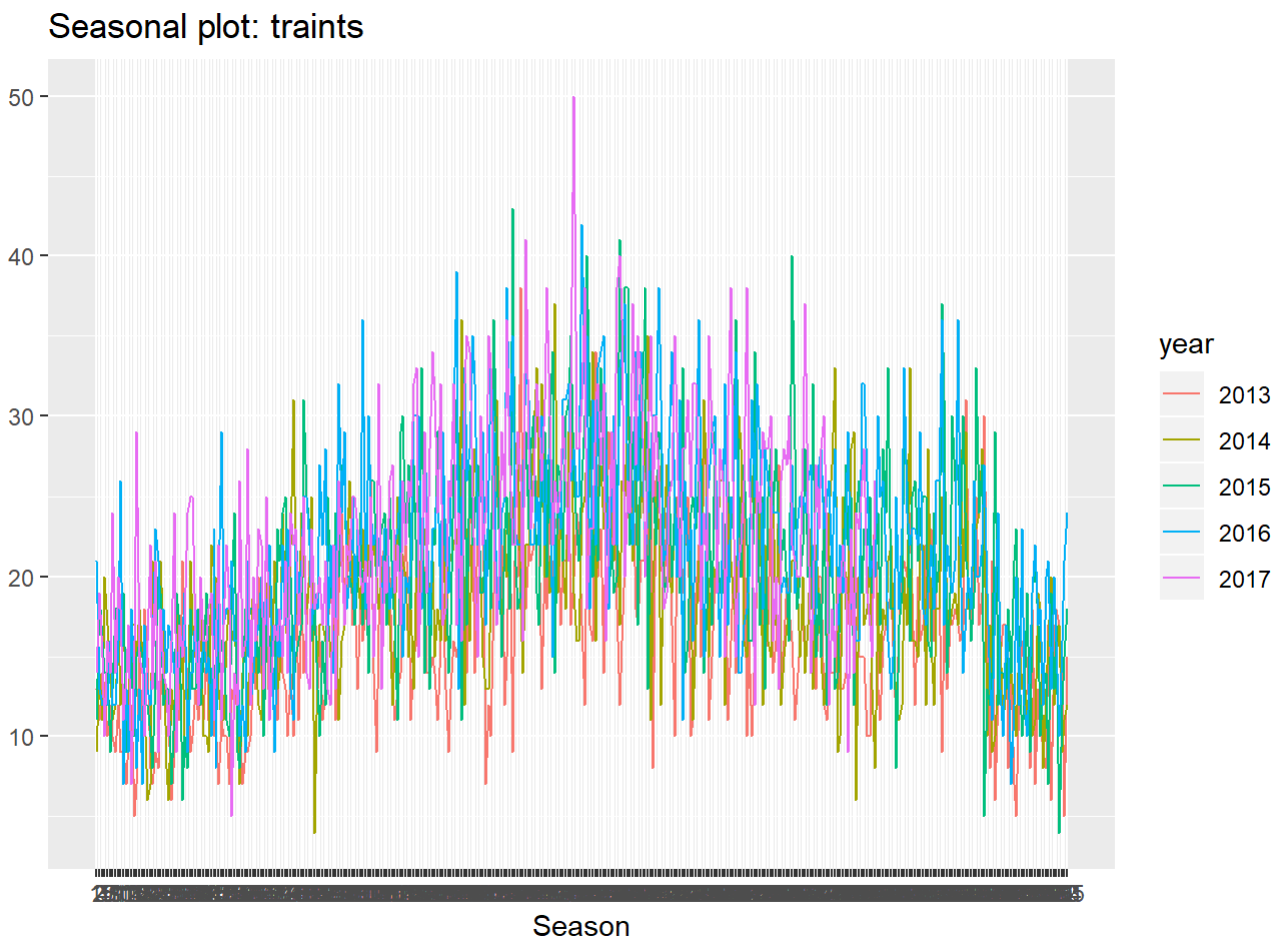
```
autoplot(traints)
```



The aim of this plot was to understand the growth of sales over time for a specific item. As we can see, it has a multiplicative growth with an increasing trend over time and seasonality.

The ggseasonplot function plots a seasonal plot. This is like a time plot except that the data are plotted against the season in separate years.

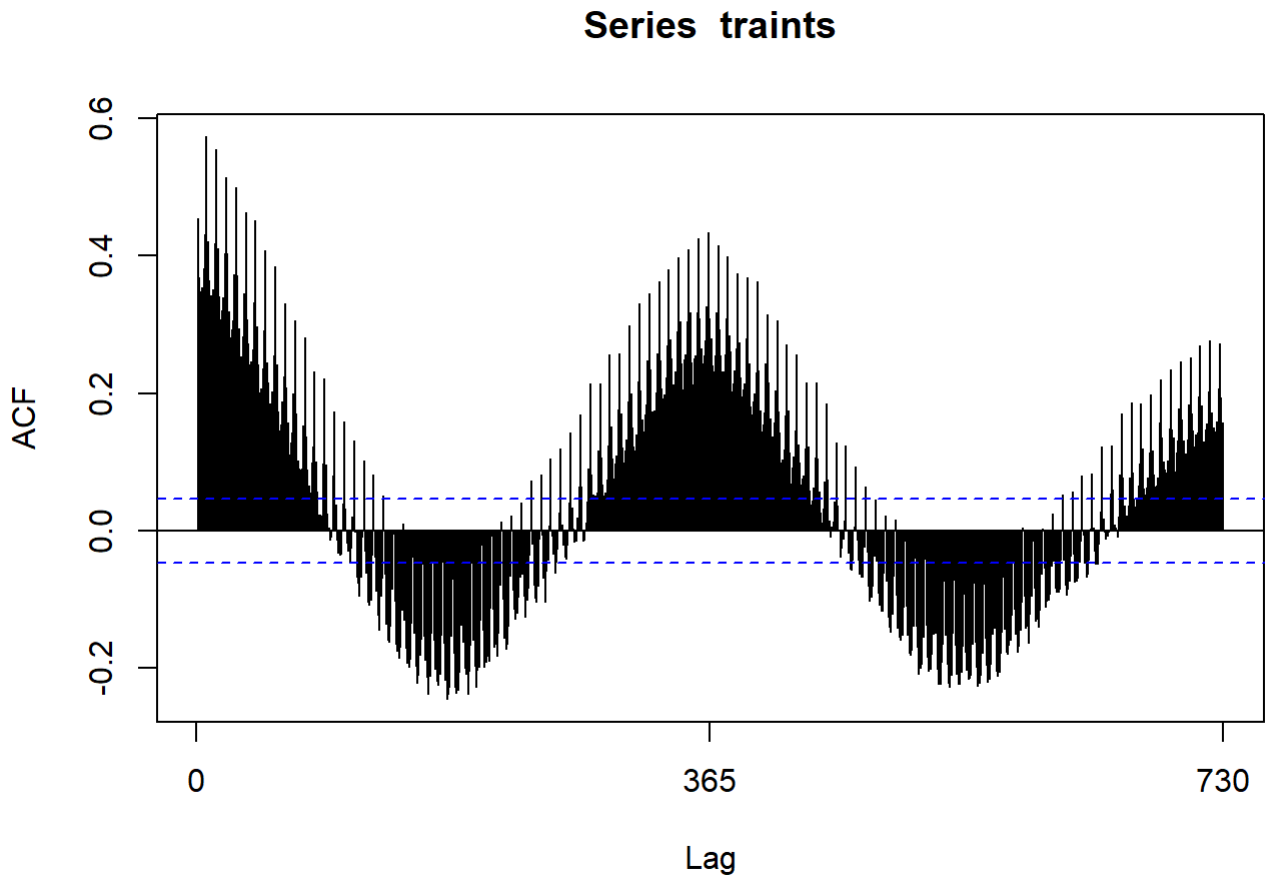
```
ggseasonplot(traints)
```



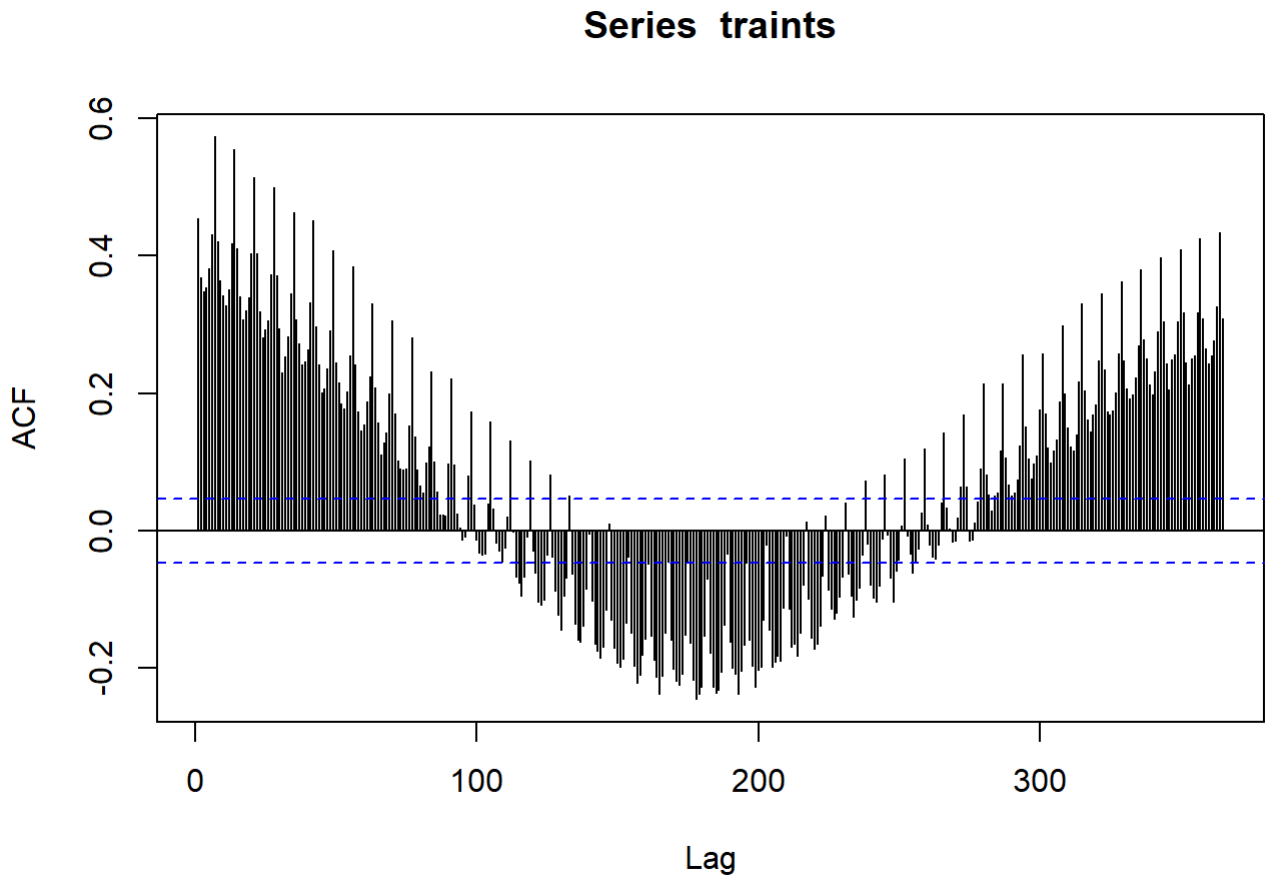
In the graph, we can see an increase towards the middle and a decrease afterwards. Possibly, the season is halfway through the year during summer time.

Finally, we want to take a look at the auto-correlation function below. Auto-correlation is the correlation of a signal with a delayed copy of itself as a function of delay.

```
Acf(traints)
```




```
Acf(traints, lag.max = 365)
```



In the graph, the auto correlation plot shows that the slow decrease in ACF plot is due to a trend decrease while the peaks and troughs are due to seasonality. The default lag number was used with the following formula $10 * log10(N/m)$ where NN is the number of observations and mm the number of series. Another option is setting the lag.max to the number of periods of the time series

Now, we want to see if the data is stationary.

```
traindiff<-diff(log(train$sales)) #Transforming the train data into difference of logs
traindiff%>%ur.kpss()%>%summary()
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 39 lags.
##
## Value of test-statistic is: NaN
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

```
ndiffs(traindiff)
```

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

Yes, the data is stationary as ndiffs equals 0.

S-Arima

First, we are going to try to forecast with an Arima model. For simplicity purposes, we will use the auto.arima function and this will return the best possible model according to AIC, AICc, and BIC.

```
Arimafit <- auto.arima(traints)
Arimafit
```

```
## Series: traints
## ARIMA(5,1,2)
##
## Coefficients:
##          ar1      ar2      ar3      ar4      ar5      ma1     ma2
##       0.0659  -0.1938  -0.1773  -0.1605  -0.1353  -0.9406  0.1341
## s.e.  0.0823   0.0341   0.0317   0.0303   0.0328   0.0803  0.0727
##
## sigma^2 estimated as 27.79:  log likelihood=-5395.81
## AIC=10807.61   AICc=10807.7   BIC=10851.36
```

The best Arima model is with AR(5) and MA(2).

Now, we will forecast the series 90 periods ahead (3 months).

```
Arimaforecast<-forecast::forecast(Arimafit,h=90)
#Arimaforecast
```

Finally, we will assess the accuracy for the Arima model using the test data.

```
(ArimaAcc<-accuracy(Arimaforecast,testts))
```

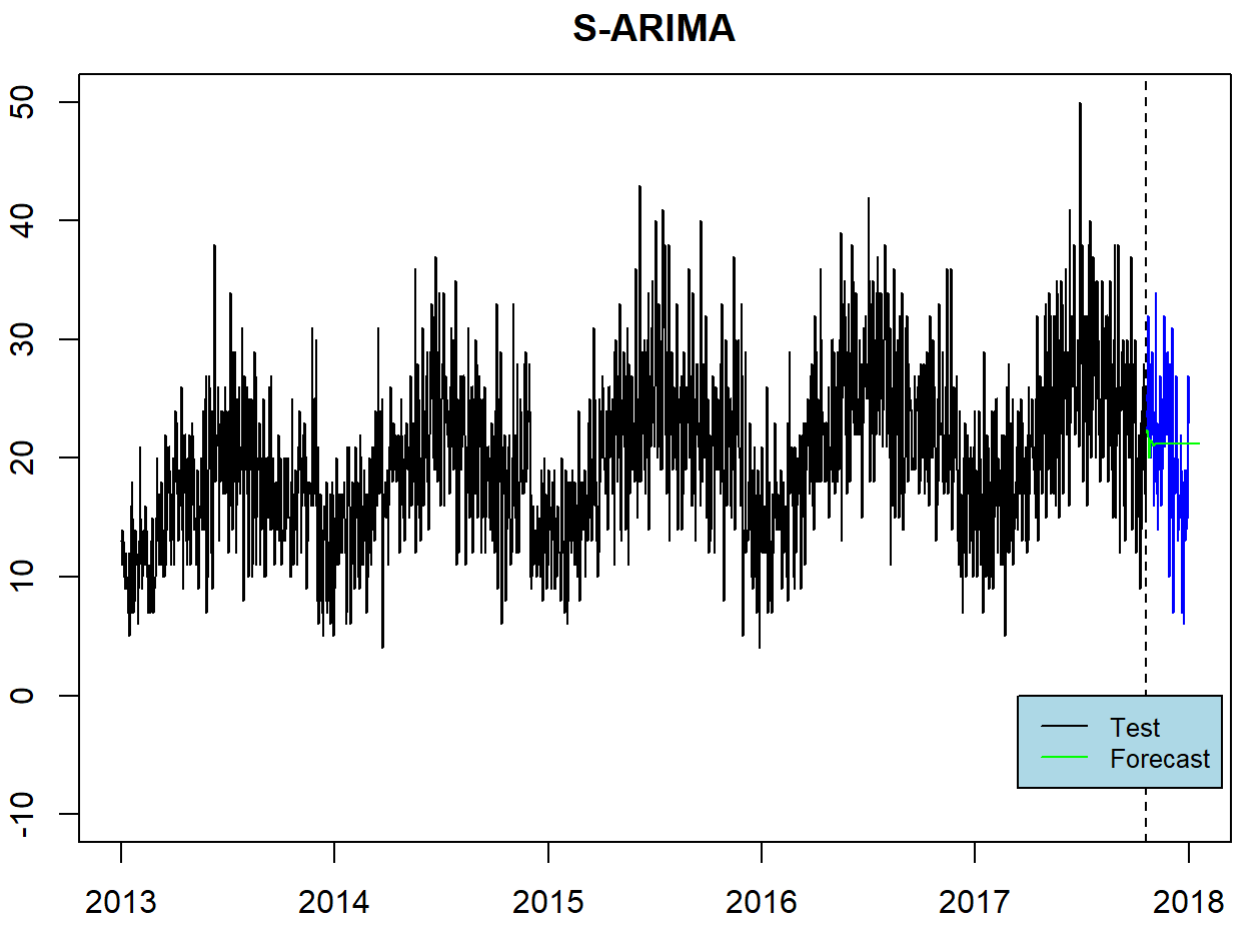
```
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set  0.04276758 5.259338 4.138761  -7.888535 24.37942 0.7052917
## Test set     -0.81024000 6.050362 4.826005 -16.930847 31.19720 0.8224057
##                      ACF1 Theil's U
## Training set -0.005034803        NA
## Test set      0.225786669 0.7007904
```

We will now store the accuracy values in a data frame to later compare to our other forecast models.

```
models<-data.frame(ArimaAcc[2,])
```

Plotting the S-Arima Forecast:

```
par(mai=c(0.5,0.5,0.5,0.5))
plot(window(traints),type="l",xlim=c(2013.0,2018.0),ylim=c(-10,50),
     ylab="Percent Monthly Change",xlab="Period", main="S-ARIMA")
lines(testts,col="blue",lwd=1)
lines(Arimaforecast$mean,col="green",lwd=1)
abline(v=2017.8,lty=2)
legend(2017.2, 0, legend=c("Test", "Forecast"),
       col=c("black","green", "red", "red"),
       lty=c(1,1,2,2), cex=0.8, text.font=1, bg='lightblue')
```



Error Trend Seasonal (ETS)

For our next forecast model, we will use the Error Trend Seasonal method. This method uses a univariate forecasting method; its uses focuses on trend and seasonal components.

```
ETSfit <- stlf(traints)
#summary(ETSfit)
```

We now use the forecast method on the fit that we created previously to forecaset 90 days ahead from the training time series.

```
ETSforecast<-forecast::forecast(ETSfit,h=90)
#ETSforecast
```

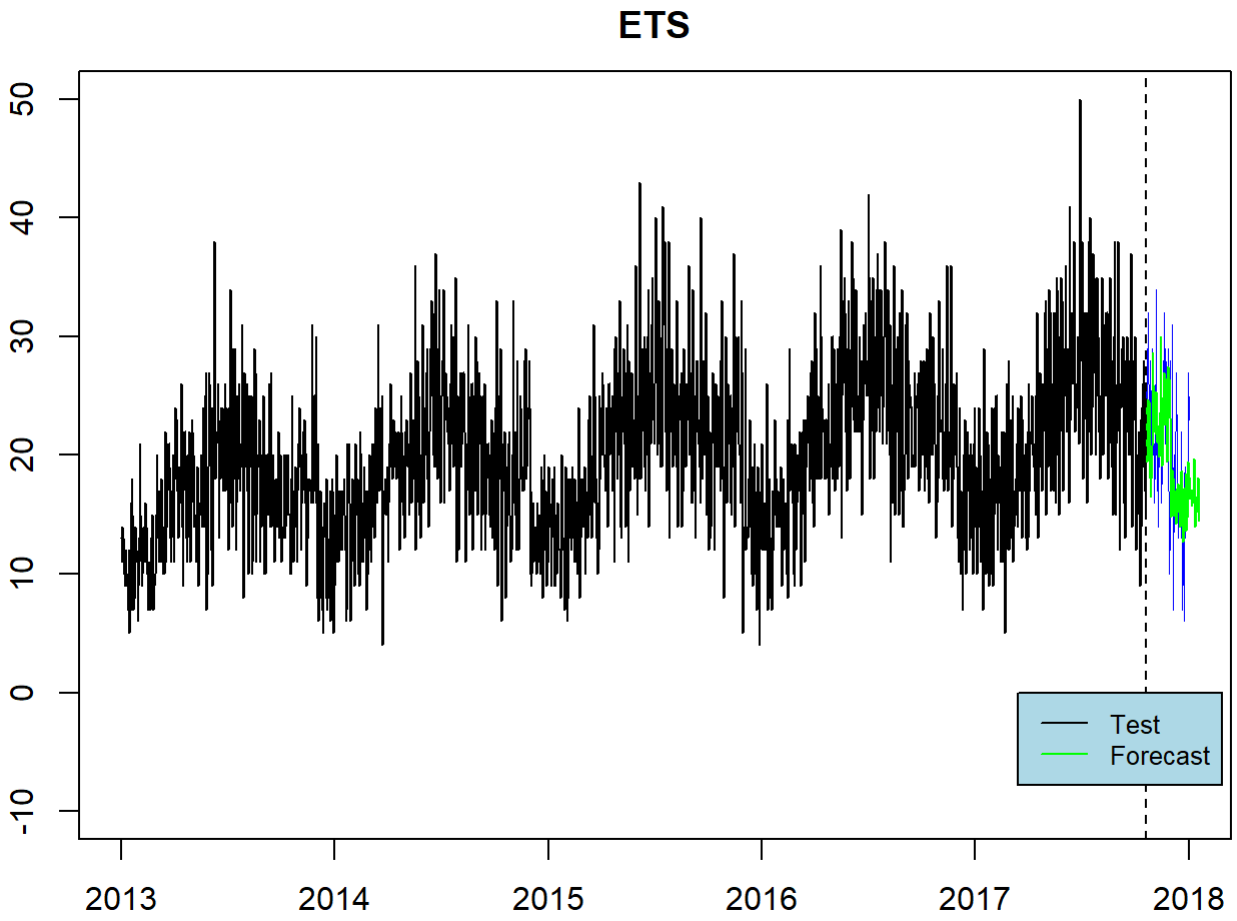
We now compare the accuracy of the forecast against the testing dataset.

```
(ETSAcc<-accuracy(ETSforecast,testts))
```

```
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.2241945 4.639167 3.659454 -5.347651 21.06472 0.6236122
## Test set     0.3442113 5.670502 4.282828 -7.445377 25.67712 0.7298423
##                    ACF1 Theil's U
## Training set 0.01859466        NA
## Test set     0.07598504 0.6413499
```

Plotting ETS forecast:

```
par(mai=c(0.5,0.5,0.5,0.5))
plot(traints,type="l",xlim=c(2013.0,2018.0),ylim=c(-10,50),
     ylab="Percent Monthly Change",xlab="Period", main="ETS")
lines(testts,col="blue",lwd=0.5)
lines(ETSforecast$mean,col="green",lwd=1)
abline(v=2017.8,lty=2)
legend(2017.2, 0, legend=c("Test", "Forecast"),
       col=c("black","green", "red", "red"),
       lty=c(1,1,2,2), cex=0.8, text.font=1, bg='lightblue')
```



Storing results:

```
models <- data.frame(models, ETSAcc[2,])
```

KNN Forecasting

This is a new method we researched for forecasting. The knn_forecasting uses KNN regression to aggregate the targets of the nearest neighbor. We calculate our k to be equal to 21 according to the default k formula of sqrt(N)/2.

The parameters for this function are as follows: h = the forecast horizon lag = integer vector indicating lagged values of the targets (usually frequency of the time series) k = # of nearest neighbors

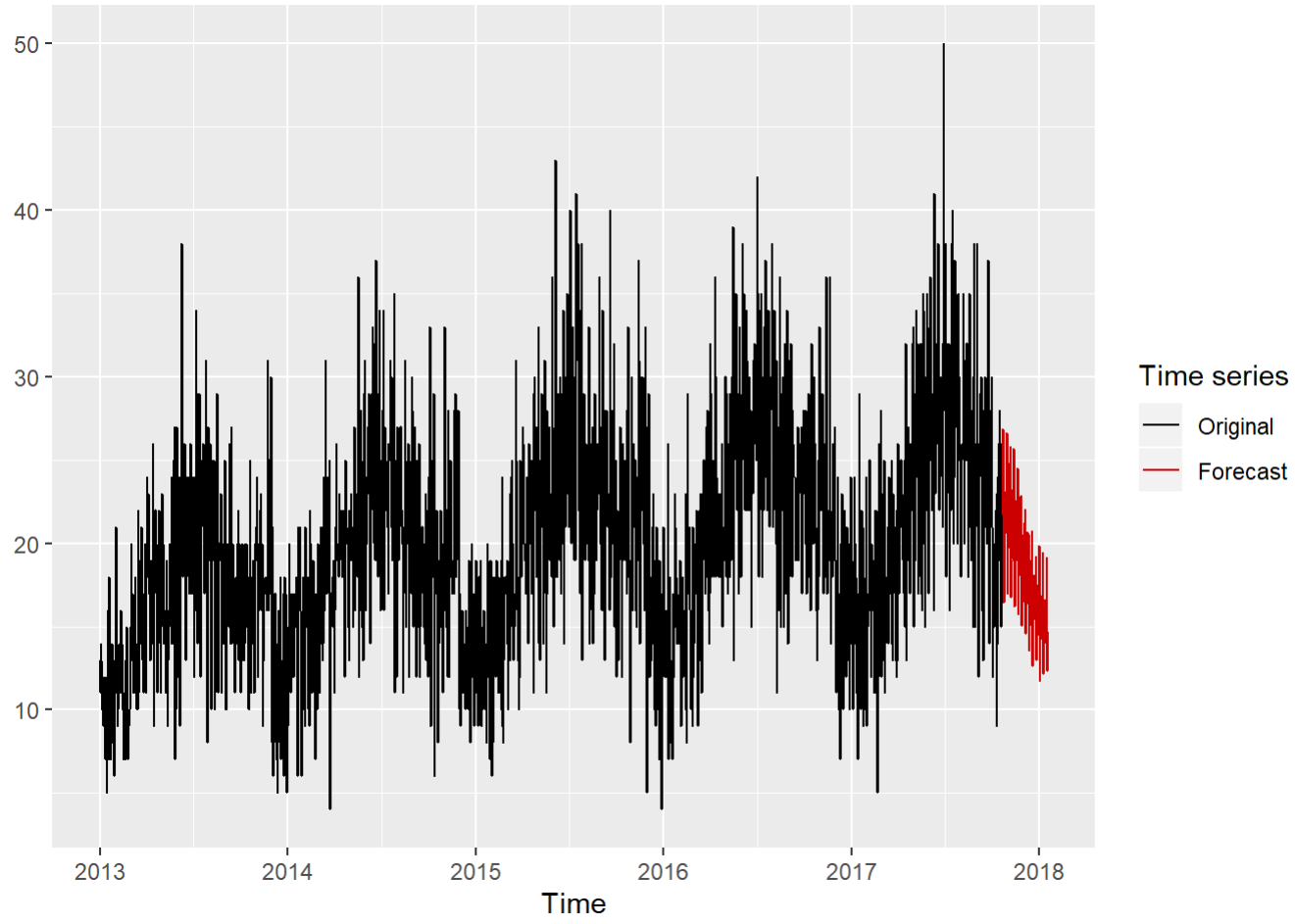
```
KNNfit <- knn_forecasting(traints, lag=1:365, h = 90, k = 21)
```

Plotting KNN Forecast:

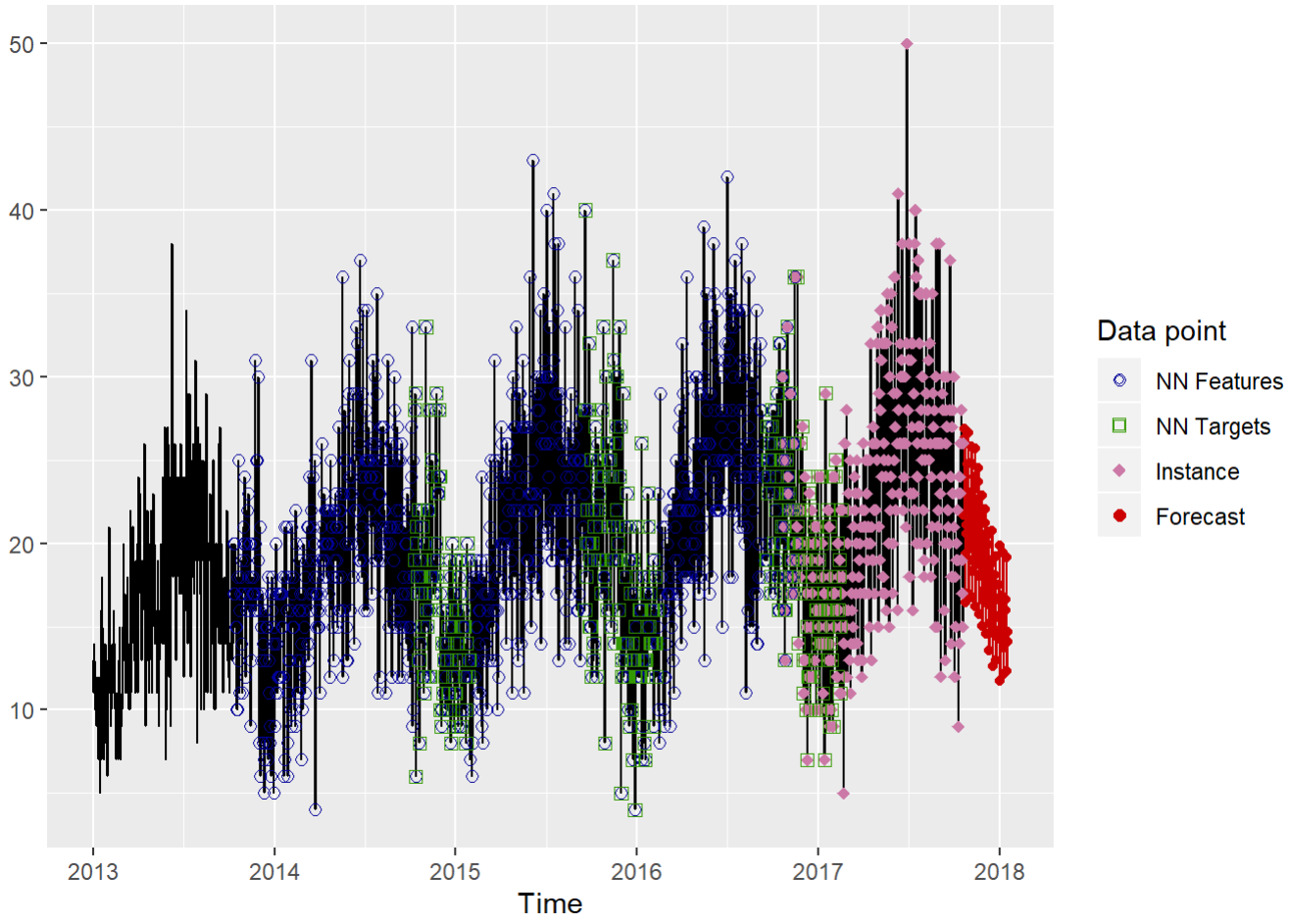
```
print(KNNfit$prediction)
```

```
## Time Series:
## Start = c(2017, 294)
## End = c(2018, 18)
## Frequency = 365
##  [1] 21.66667 26.28571 26.85714 16.42857 19.33333 20.57143 21.09524 23.09524
##  [9] 24.76190 26.61905 16.95238 19.71429 19.80952 20.61905 22.57143 24.80952
## [17] 25.80952 16.71429 18.33333 18.90476 19.04762 23.23810 24.00000 25.71429
## [25] 16.19048 18.71429 19.14286 18.95238 22.57143 23.66667 24.52381 15.76190
## [33] 18.04762 19.04762 18.14286 21.76190 22.80952 22.85714 15.04762 16.80952
## [41] 17.14286 16.47619 20.52381 21.19048 22.04762 14.57143 16.61905 18.04762
## [49] 16.38095 20.71429 19.90476 20.57143 13.57143 15.19048 16.28571 15.09524
## [57] 18.76190 18.95238 20.76190 12.61905 14.61905 15.76190 15.42857 18.09524
## [65] 16.71429 19.23810 13.00000 14.38095 14.76190 14.52381 17.52381 16.61905
## [73] 19.85714 11.76190 14.23810 14.33333 14.80952 16.85714 15.66667 19.47619
## [81] 12.14286 13.76190 14.33333 14.04762 16.61905 16.00000 19.14286 12.33333
## [89] 14.09524 14.66667
```

```
autoplot(KNNfit)
```




```
autoplot(KNNfit,highlight = 'neighbors', faceting = F)
```



Computing and comparing the accuracy of the forecast against the test:

```
(KNNAcc<-round(accuracy(KNNfit$prediction,testts),2))
```

```
##            ME RMSE  MAE   MPE  MAPE  ACF1 Theil's U
## Test set 1.21 4.84 3.81 -1.73 21.73 -0.06      0.55
```

Saving results of KNN model:

```
models <- models[-6,]
models <- data.frame(models, KNNAcc[1,])
colnames(models)<-c("S-ARIMA","ETS", "KNN")
```

Comparison of Models and Results

```
print(models)
```

```
##               S-ARIMA         ETS   KNN
## ME         -0.8102400  0.34421128  1.21
## RMSE        6.0503618  5.67050174  4.84
## MAE         4.8260048  4.28282838  3.81
## MPE       -16.9308466 -7.44537721 -1.73
## MAPE       31.1972038 25.67711613 21.73
## ACF1        0.2257867  0.07598504 -0.06
## Theil's U   0.7007904  0.64134986  0.55
```

RMSE indicates the absolute fit of the model to the data, more specifically, how close the observed data points are to the models's predicted values.

MAE measures the average magnitude of the errors in a set of predictions.

MPE is the average percentage of errors in a forecast model.

MAPE is a measure of prediction accuracy of a forecasting method in statistics.

Overall, we see that the **KNN Forecasting model** yields the best accuracy and the lowest error in most of the categories including RMSE, MAE, MPE, MAPE. For item 1 in store 1, our KNN Forecasting model outperforms the S-ARIMA and ETS models.

Further Analysis

Given time constraints, we were unable to show predictions for all 50 items in every store location. To continute this analysis, we would ideally examine all 50 items at the 10 store locations to determine which forecast model performs the best across all item-store combinations.