Predicting Weeekly Sales from Store Features

Data Science: Capstone Project (HarvardX PH125.9x)

Aditya Prasad Dash

Introduction

Understanding the structure and patterns in the data, developing data structures for effectively storing and visualizing it, and using insights from it to predict the target variable of interest from values of other variables is the realm of data science. In this project, I have used the Walmart dataset form kaggle [1] to predict the weekly sales of a store based on the various features of the store and the environment affecting the sales. In the first step, as the Weekly_Sales has numerical values, a linear regression model was trained to predict the Weekly_Sales from the other features. Then, a k-Nearest neighbors model was trained over the dataset for this regression task and the best value of k was found after evaluating the model performance for different k. Then as sometimes it is more useful to understand if a given set of conditions would produce a high or low weekly_sales, the Weekly_Sales column was categorized into high, medium and low weekly sales and a knn model was trained to predict the class from the feature variables.

Methods

Reading, visualizing and preprocessing the data

The first step is to load the required libraries for the analysis.

```r
if(!require(tidyverse)) install.packages("tidyverse")
library(tidyverse)
if(!require(tidyr)) install.packages("tidyr")
library(tidyr)
if(!require(caret)) install.packages("caret")
library(caret)
if(!require(stringr)) install.packages("stringr")
library(stringr)
if(!require(ggplot2)) install.packages("ggplot2")
library(ggplot2)
if(!require(lubridate)) install.packages("lubridate")
library(lubridate)
if(!require(corrplot)) install.packages("corrplot")
library(corrplot)
#library(neuralnet)
```

I downloaded the dataset into my computer, therefore, I used the read.csv function in R to read the Walmart dataset and store it into a dataframe called walmart_dataset. The function str(walmart_dataset) displays information about the number of rows and columns of the dataset, the data type in each column and the first few entries of each column.

```r
walmart_dataset <- read.csv("/Users/aditya/Documents/Coursework/Online_Courses/Machine_Learning_and_Dee
str(walmart_dataset)
```

```
## 'data.frame':    6435 obs. of  8 variables:
##  $ Store       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Date        : chr  "05-02-2010" "12-02-2010" "19-02-2010" "26-02-2010" ...
##  $ Weekly_Sales: num  1643691 1641957 1611968 1409728 1554807 ...
##  $ Holiday_Flag: int  0 1 0 0 0 0 0 0 0 0 ...
##  $ Temperature : num  42.3 38.5 39.9 46.6 46.5 ...
##  $ Fuel_Price  : num  2.57 2.55 2.51 2.56 2.62 ...
##  $ CPI         : num  211 211 211 211 211 ...
##  $ Unemployment: num  8.11 8.11 8.11 8.11 8.11 ...
```

We notice that the dataset contains 6435 rows and 8 columns. As we want to predict Weekly_Sales from the other variables, I will refer to Weekly_Sales as the target variable and all other variables as feature variables. The data type of "Store" and "Holiday_Flag" is integer, that for "Date" is chr and for the other variables, and that for "Weekly_Sales","Holiday_Flag","Temperature","Fuel_Price","CPI" and "Unemployment" is num.

The function head(dataset,n) displays the first n rows of the dataset, we can use it to visualize walmart_dataset shows the entries of the first 5 rows.

```r
head(walmart_dataset,5)
```

```
##   Store       Date Weekly_Sales Holiday_Flag Temperature Fuel_Price      CPI
## 1     1 05-02-2010      1643691            0       42.31      2.572 211.0964
## 2     1 12-02-2010      1641957            1       38.51      2.548 211.2422
## 3     1 19-02-2010      1611968            0       39.93      2.514 211.2891
## 4     1 26-02-2010      1409728            0       46.63      2.561 211.3196
## 5     1 05-03-2010      1554807            0       46.50      2.625 211.3501
##   Unemployment
## 1        8.106
## 2        8.106
## 3        8.106
## 4        8.106
## 5        8.106
```

We first omit all the rows with nan with the function na.omit(dataset) function. Then we check the range of our target variable Weekly_Sales and as both minimum and maximum Weekly sales are positive, it seems reasonable. Next we look at the store column which has integer entries. To find the number of stores included in the dataset we can use the unique() function with argument walmart_dataset$Store to find the unique store numbers.

```r
walmart_dataset<-na.omit(walmart_dataset)
range(walmart_dataset$Weekly_Sales)
```

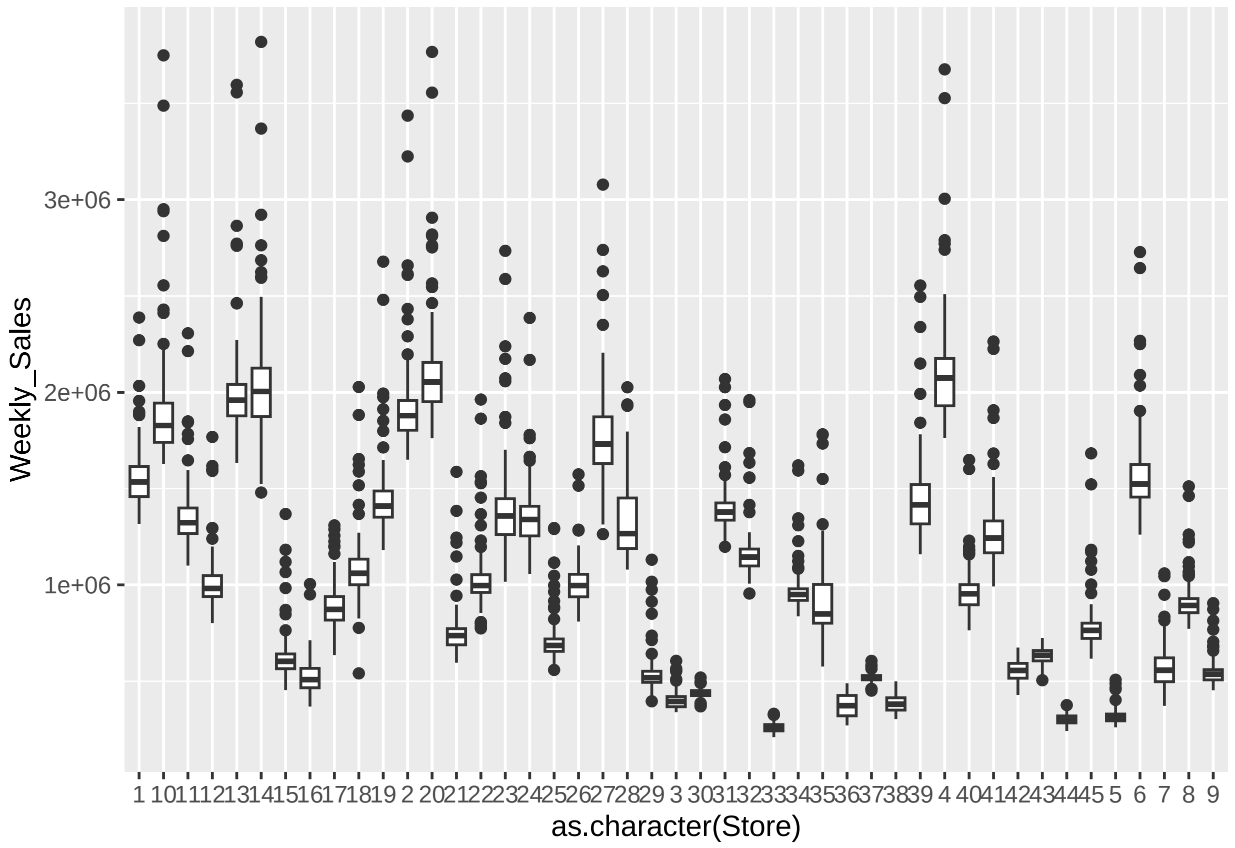
```
## [1]  209986.2 3818686.5
```

```r
unique(walmart_dataset$Store)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
```

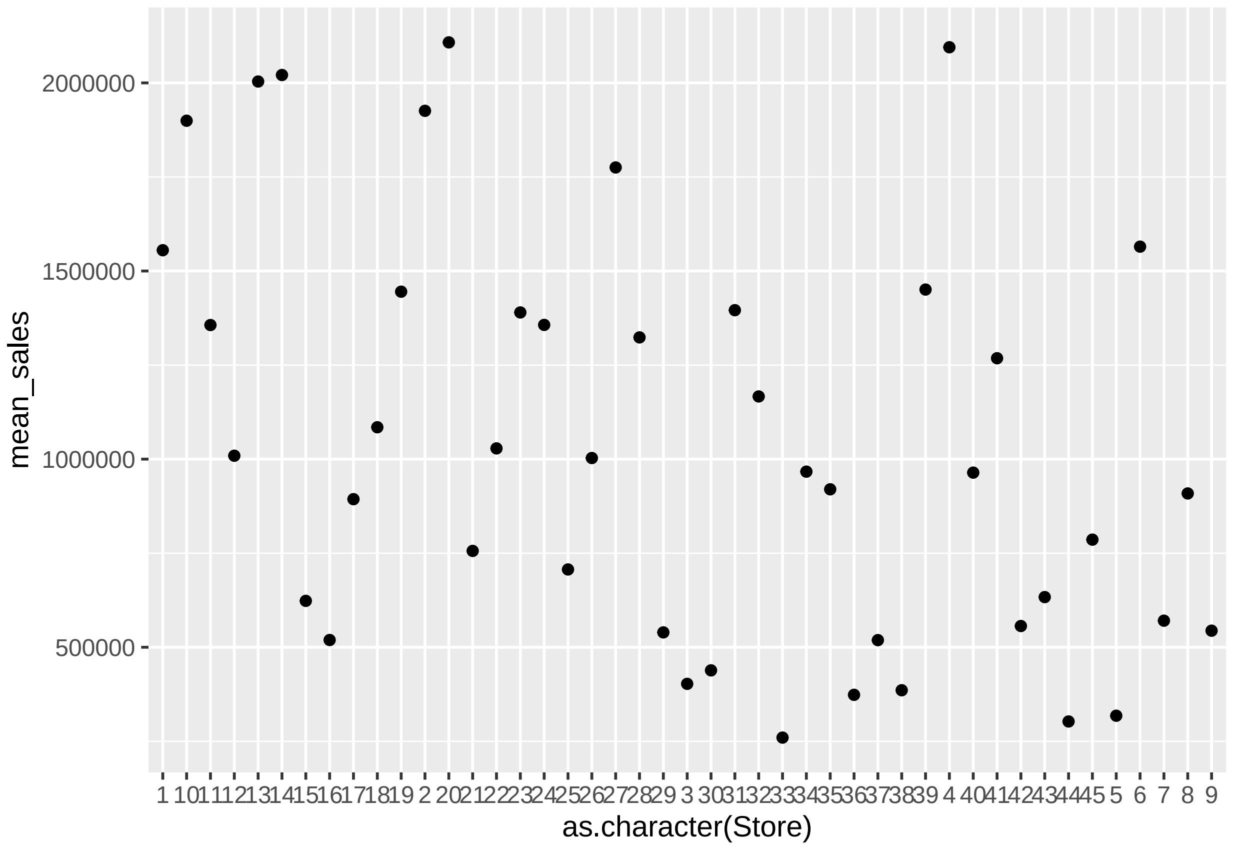
We can notice that there are 45 stores in the dataset with the store numbers ranging from 1 to 45. Some stores can have more sales than others due to location, total volume of products and other factors. Therefore, its useful to visualize the distribution of Weekly sales for different stores. For that, we can use a boxplot which displays the distribution of a numerical variable for each instance of a categorical variable.

```
walmart_dataset%>% ggplot(aes(as.character(Store), Weekly_Sales)) + geom_boxplot()
```



We see that for each store the weekly sales are distributed over a wide range. Moreover, the average (mean) weekly sales for a given store, shown by the central line in the boxplot is different accross different stores. To visualize the mean sales better, we can group the dataset by store numbers and then extract the mean sales for each store and store it in a tibble called mean_sales_store. Next, we can use ggplot and the geom_point function to make a scatterplot of mean scales as a function of the store number.

```
mean_sales_store<-walmart_dataset%>%group_by(Store)%>%summarise(mean_sales=mean(Weekly_Sales))
mean_sales_store %>% ggplot(aes(as.character(Store), mean_sales)) + geom_point()
```



We notice that the weekly sales are spread in each store and they have different averages for different stores, therefore it is useful to use this information in predicting weekly sales. As the store number should not have a heirarchy, we should use one-hot-encoding (https://stackoverflow.com/questions/74706268/how-to-create-dummy-variables-in-r-based-on-multiple-values-within-each-cell-in) to make separate columns for each store in which the entry for a row (observation) is 1 if the Store corresponds to that store and 0 otherwise. To do this we can use the pivot wider function in R to separate the Store numbers into different columns and use length as a funcion to put 1 if that observation is from that Store and 0 (values_fill is set to 0) otherwise and update the walmart_dataset.

```
walmart_dataset<-walmart_dataset%>% pivot_wider(names_from = Store, names_prefix="Store",values_from = S
```

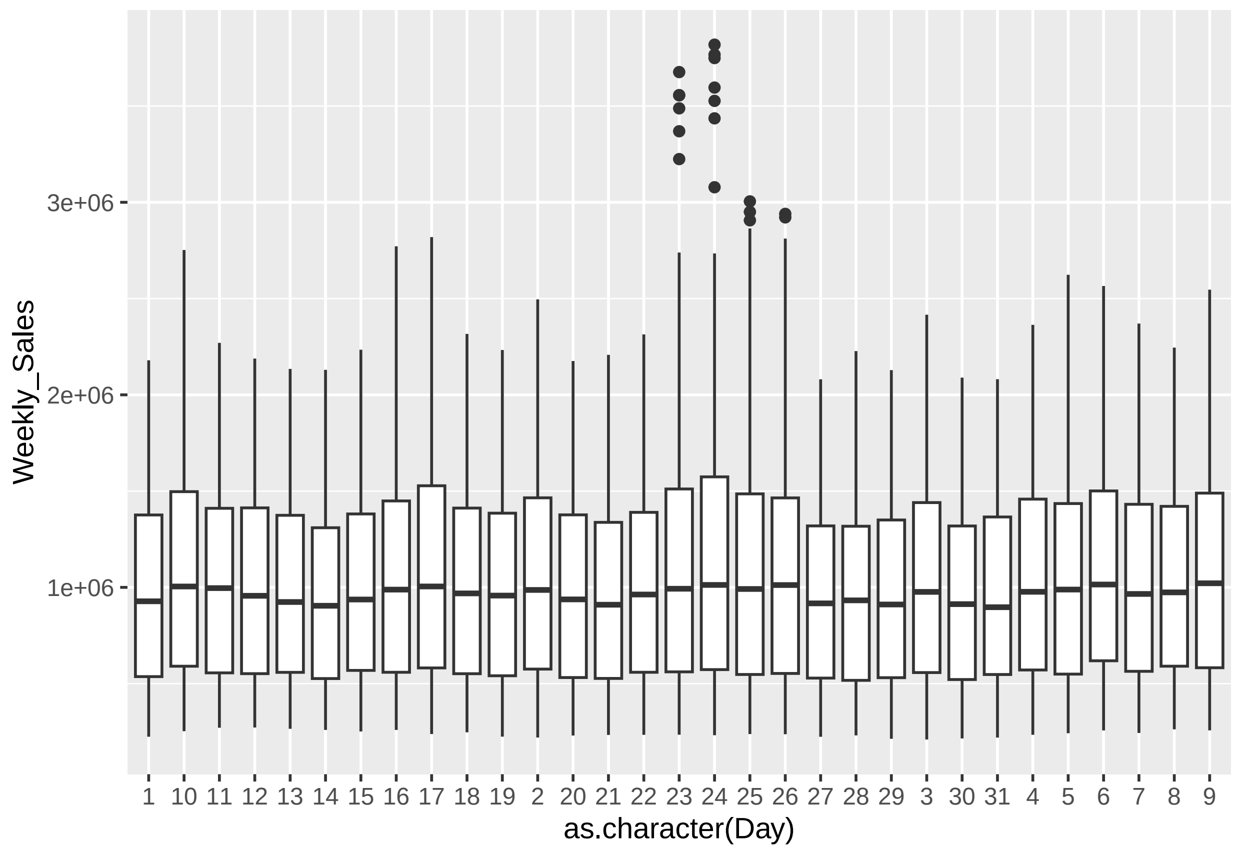
Next we try to understand the date column. The format of the date in the dataset is day-month-year as a character. However, to extract meaningful insights from we should separate the month day and year into separate columns. To do this we can the dmy function in lubridate package to convert Date into the appropriate format and then mutate the dataset with the Day, Month and Year columns after extracting it from the Date variable. Displaying the first 5 entries using the head function shows that indeed we have created new columns corresponding to the day, month and year.

```
walmart_dataset<-walmart_dataset%>%mutate(Day=day(dmy(Date)),Month=month(dmy(Date)), Year=year(dmy(Date
head(walmart_dataset%>%select(c("Day","Month","Year")),5)
```

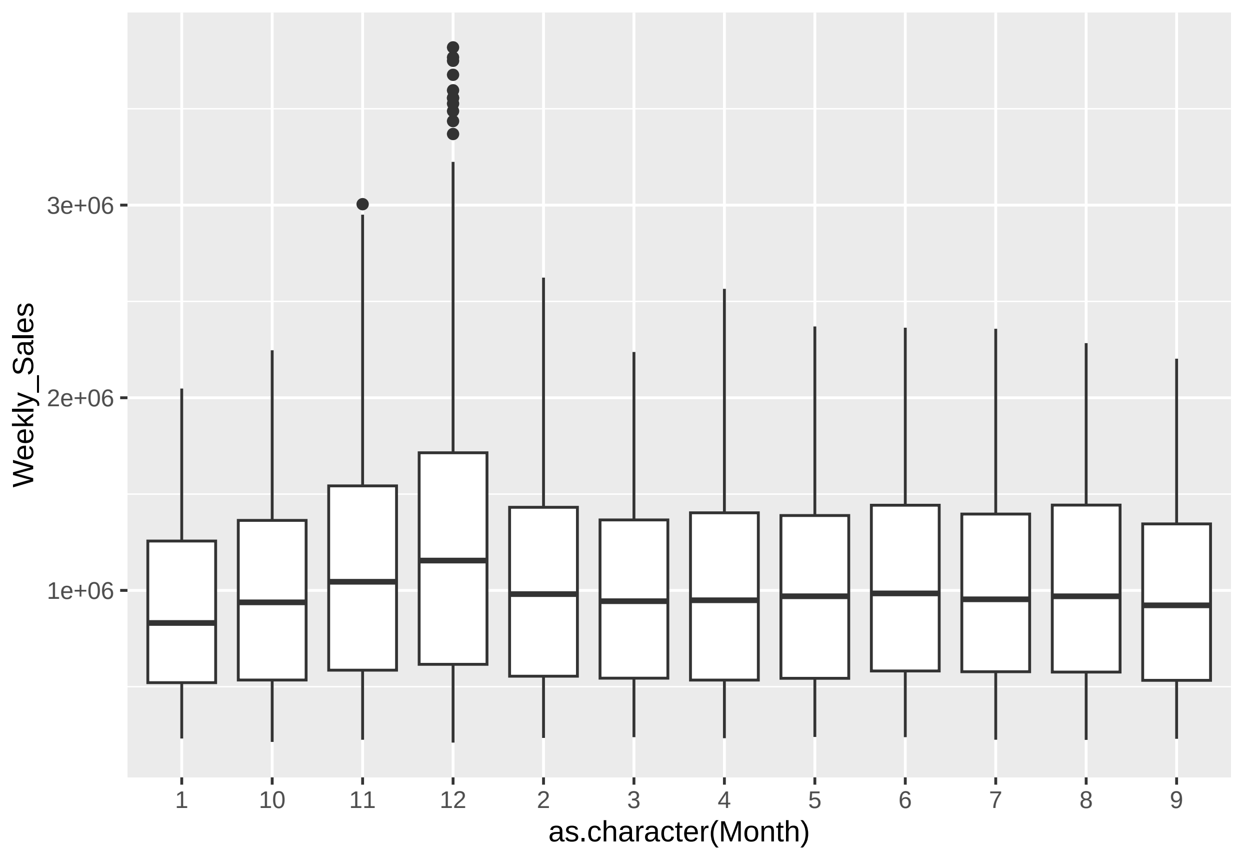
```
## # A tibble: 5 x 3
##     Day Month  Year
##   <int> <dbl> <dbl>
## 1     5     2  2010
## 2    12      2  2010
## 3    19      2  2010
## 4    26      2  2010
## 5     5      3  2010
```

Now, we try to visualize the distribution of weekly sales accross day, month and year by creating 3 different boxplots correspondin to each of them.

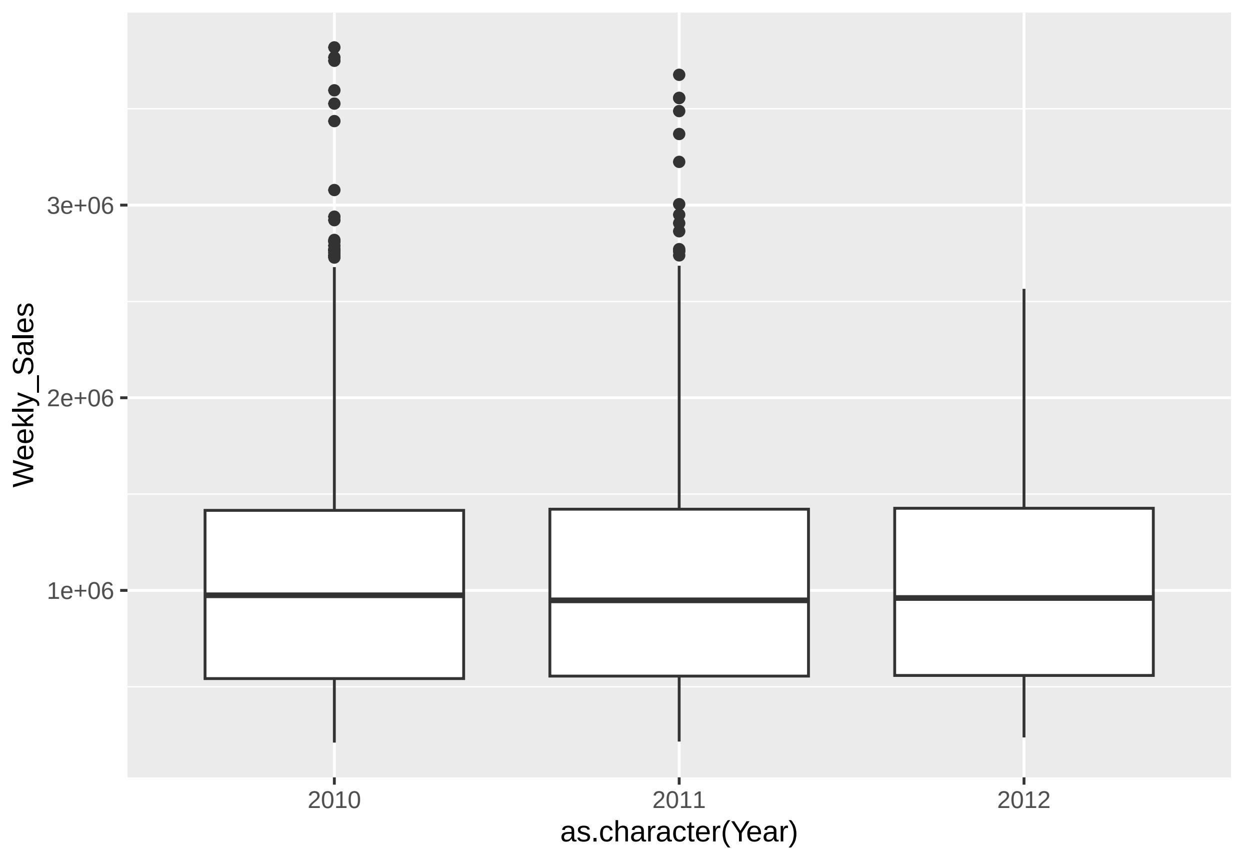
```
walmart_dataset%>% ggplot(aes(as.character(Day), Weekly_Sales)) + geom_boxplot()
```



```
walmart_dataset%>% ggplot(aes(as.character(Month), Weekly_Sales)) + geom_boxplot()
```

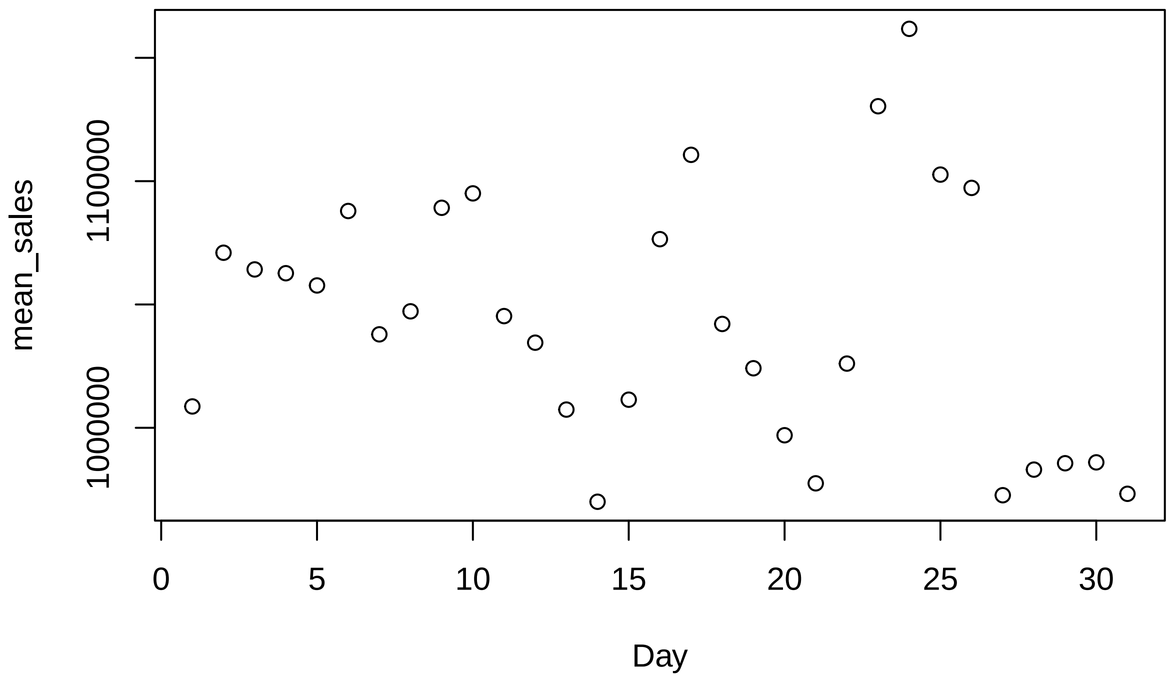


```
walmart_dataset%>% ggplot(aes(as.character(Year), Weekly_Sales)) + geom_boxplot()
```

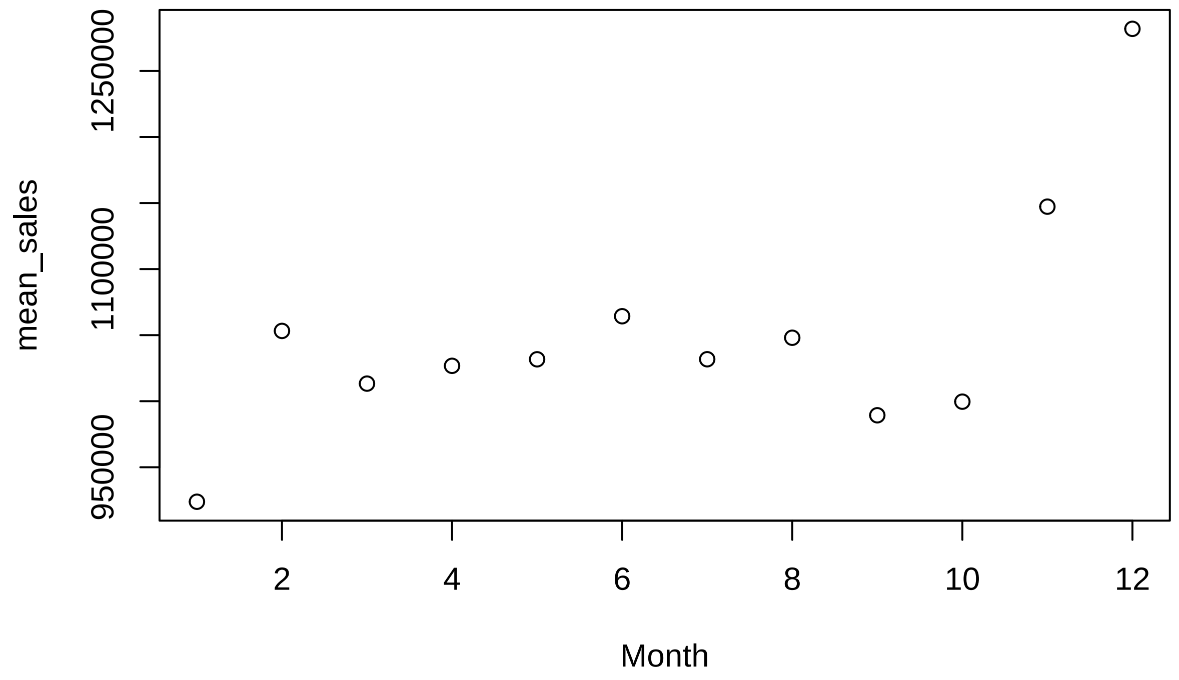


We see that the distribution of sales shows slight variation with day, month and year. Especially, the average sales is higher around december and 2010 had higher mean weekly sales than 2011 which was higher than that in 2012. To visualize the distribution of mean weekly sales, we can plot the scatter plot of the mean of the weekly sales with day, month and year which agrees with our findings from the boxplots.

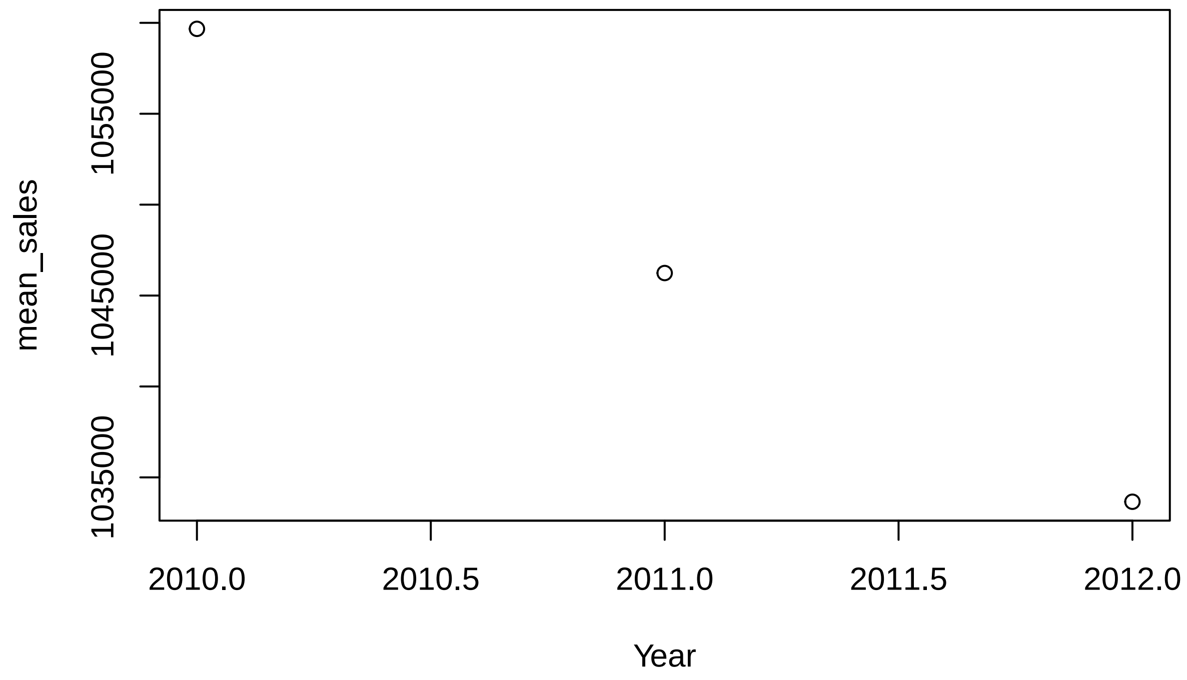
```
plot(walmart_dataset%>%group_by(Day)%>%summarise(mean_sales=mean(Weekly_Sales)))
```



```r
plot(walmart_dataset%>%group_by(Month)%>%summarise(mean_sales=mean(Weekly_Sales)))
```



```
plot(walmart_dataset%>%group_by(Year)%>%summarise(mean_sales=mean(Weekly_Sales)))
```



To understand the dependence of the mean weekly sales on Day, Month and year we can group the dataset with the correspoding variable and arrange the categories with descending values of mean sales. We find that on average, the sales were highest on day and minimum on day 14, highest in December and lowest in January and highest in 2010 and loweset in 2012.

```
walmart_dataset%>%group_by(Day)%>%summarise(n=n(),mean_sales=mean(Weekly_Sales))%>%arrange(desc(mean_sal
```

```
## # A tibble: 31 x 3
##      Day     n mean_sales
##    <int> <int>      <dbl>
##  1    24   225   1161746.
##  2    23   225   1130371.
##  3    17   225   1110669.
##  4    25   225   1102654.
##  5    26   225   1097265.
##  6    10   225   1095039.
##  7     9   225   1089198.
##  8     6   225   1087901.
##  9    16   225   1076483.
## 10     2   225   1070976.
## # i 21 more rows
```

```
walmart_dataset%>%group_by(Month)%>%summarise(n=n(),mean_sales=mean(Weekly_Sales))%>%arrange(desc(mean_s
```

```
## # A tibble: 12 x 3
##    Month     n mean_sales
##    <dbl> <int>      <dbl>
##  1    12   450   1281864.
##  2    11   360   1147266.
##  3     6   585   1064325.
##  4     2   540   1053200.
##  5     8   585   1048017.
##  6     7   630   1031748.
##  7     5   540   1031714.
##  8     4   630   1026762.
##  9     3   585   1013309.
## 10    10   585    999632.
## 11     9   585    989335.
## 12     1   360    923885.
```

```
walmart_dataset%>%group_by(Year)%>%summarise(n=n(),mean_sales=mean(Weekly_Sales))%>%arrange(desc(mean_sa
```

```
## # A tibble: 3 x 3
##    Year     n mean_sales
##   <dbl> <int>      <dbl>
## 1  2010  2160   1059670.
## 2  2011  2340   1046239.
## 3  2012  1935   1033660.
```

As we see a dependene of the weekly sales on day, month and year, we can separate them into different columns using one-hot encoding similar to that done for Stores. However, as we have 31 days, including each day as a feature can overcomplicate the model, therefore we only consider the effect of Month and Year. We can use pivot wider to create separate columns for each month and Year which takes the value 1 if the observation corresponds to that month or year and 0 otherwise.

```
walmart_dataset<-walmart_dataset%>% pivot_wider(names_from = Month, names_prefix="Month",values_from = M
walmart_dataset<-walmart_dataset%>% pivot_wider(names_from = Year, names_prefix="Year",values_from = Yea
colnames(walmart_dataset)
```

```
##  [1] "Date"         "Weekly_Sales" "Holiday_Flag" "Temperature"  "Fuel_Price"
##  [6] "CPI"          "Unemployment" "Store1"       "Store2"       "Store3"
## [11] "Store4"       "Store5"       "Store6"       "Store7"       "Store8"
## [16] "Store9"       "Store10"      "Store11"      "Store12"      "Store13"
## [21] "Store14"      "Store15"      "Store16"      "Store17"      "Store18"
## [26] "Store19"      "Store20"      "Store21"      "Store22"      "Store23"
## [31] "Store24"      "Store25"      "Store26"      "Store27"      "Store28"
## [36] "Store29"      "Store30"      "Store31"      "Store32"      "Store33"
## [41] "Store34"      "Store35"      "Store36"      "Store37"      "Store38"
## [46] "Store39"      "Store40"      "Store41"      "Store42"      "Store43"
## [51] "Store44"      "Store45"      "Day"          "Month2"       "Month3"
## [56] "Month4"       "Month5"       "Month6"       "Month7"       "Month8"
## [61] "Month9"       "Month10"      "Month11"      "Month12"      "Month1"
## [66] "Year2010"     "Year2011"     "Year2012"
```

Also, we remove the Date column from the dataset as its information is already contained in columns corresponding to different Month and Year.

```r
walmart_dataset<-walmart_dataset%>%select(-c("Date"))
walmart_dataset<-walmart_dataset%>%select(-c("Day"))
```

Scaling the dataset

As the numerical values of each of the feautre variables can have a different magnitude, giving unequal importance to the training parameters corresponding to them during the training process. Therefore, we should scale them to have similar magnitudes, two common methods of scaling are the standard scaler which centers the values of the column at the column mean and then divides each entry with the standard devaition. However, as the outliers are also included in calculating the mean standard deviation, it can bias the centering of the column. Therefore, we can use minmax scaling (https://www.geeksforgeeks.org/how-to-normalize-and-standardize-data-in-r/) in which we subtract the minimum value of the column from each entry and then divide each entry of the column by the range (max-min) of that column.

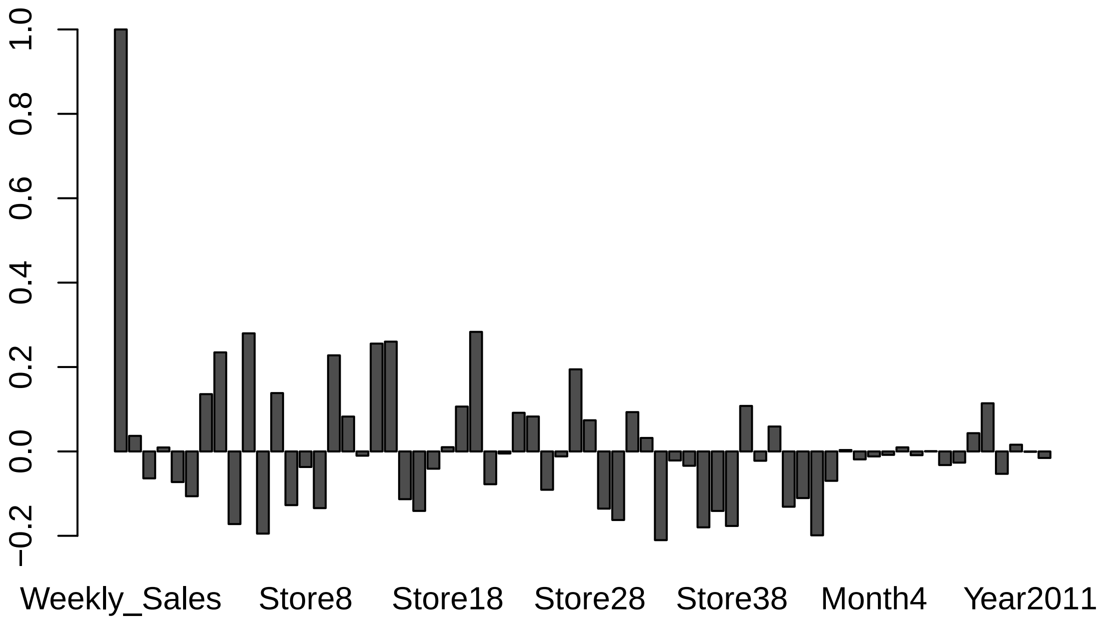
```r
#Scaling the dataset
fMinMaxSclaer <- function(x) ( #https://www.geeksforgeeks.org/how-to-normalize-and-standardize-data-in-
  (x - min(x)) / (max(x) - min(x))
)

walmart_dataset[-1]<-as.data.frame(lapply(walmart_dataset[-1],fMinMaxSclaer))
head(walmart_dataset)
```

```
## # A tibble: 6 x 66
##   Weekly_Sales Holiday_Flag Temperature Fuel_Price   CPI Unemployment Store1
##          <dbl>        <dbl>       <dbl>      <dbl> <dbl>        <dbl>  <dbl>
## 1     1643691.            0       0.434     0.0501 0.840        0.405      1
## 2     1641957.            1       0.397     0.0381 0.842        0.405      1
## 3     1611968.            0       0.411     0.0210 0.842        0.405      1
## 4     1409728.            0       0.476     0.0446 0.843        0.405      1
## 5     1554807.            0       0.475     0.0767 0.843        0.405      1
## 6     1439542.            0       0.586     0.0977 0.843        0.405      1
## # i 59 more variables: Store2 <dbl>, Store3 <dbl>, Store4 <dbl>, Store5 <dbl>,
## #   Store6 <dbl>, Store7 <dbl>, Store8 <dbl>, Store9 <dbl>, Store10 <dbl>,
## #   Store11 <dbl>, Store12 <dbl>, Store13 <dbl>, Store14 <dbl>, Store15 <dbl>,
## #   Store16 <dbl>, Store17 <dbl>, Store18 <dbl>, Store19 <dbl>, Store20 <dbl>,
## #   Store21 <dbl>, Store22 <dbl>, Store23 <dbl>, Store24 <dbl>, Store25 <dbl>,
## #   Store26 <dbl>, Store27 <dbl>, Store28 <dbl>, Store29 <dbl>, Store30 <dbl>,
## #   Store31 <dbl>, Store32 <dbl>, Store33 <dbl>, Store34 <dbl>, ...
```

Correlation between features and Weekly_Sales

To understand how each of the feature variable impacts the weekly sales, we can calculate the correlation between the data of that feature variable with Weeekly_Sales.

```r
cor_data<-cor(walmart_dataset$Weekly_Sales,walmart_dataset%>%select(where(is.numeric)))
barplot(cor_data)
```



We see that different feature variables have different correlations with Weekly sales ranging from -0.2102702149 for Store33 to 0.2833633 for Store20. ## Building the Machine Learning Model

Machine Learning Model

Partitioning the dataset into training and testing data A machine learning model is build to get trained on some dataset and be used to predict a quantity of interest in some unknown dataset which may not even exist at the present. Therefore, it is necessary to test the performance of the model by training it over a subset of the dataset(training_set) and test its performance over the other subset of the dataset (testing_set). The performance is evaluated by predicting the output for the target variable over the testing set and compare to the actual values of the target variable in in

First we find out indices of 15% of rows smapled randomly from the walmart_dataset (using Weekly_Sales column) using the createDataPartition function in R. Then we use the test index to create the testing_set which will be used for testing the final model performance after fitting it over the training set which is the subset of the dataset excluding the testing set

```
test_index <- createDataPartition(y = walmart_dataset$Weekly_Sales, times = 1,
                                  p = 0.15, list = FALSE)
testing_set <- walmart_dataset[test_index,]
training_set <- walmart_dataset[-test_index,]
```

We can see the number of observations in each set using the nrow function and see the first 5 rows of the training set using the head(training_set,5) function.

```r
nrow(training_set)
```

```
## [1] 5467
```

```r
nrow(testing_set)
```

```
## [1] 968
```

```r
head(training_set,5)
```

```
## # A tibble: 5 x 66
##   Weekly_Sales Holiday_Flag Temperature Fuel_Price   CPI Unemployment Store1
##          <dbl>        <dbl>       <dbl>      <dbl> <dbl>        <dbl>  <dbl>
## 1     1643691.            0       0.434     0.0501 0.840        0.405      1
## 2     1641957.            1       0.397     0.0381 0.842        0.405      1
## 3     1611968.            0       0.411     0.0210 0.842        0.405      1
## 4     1409728.            0       0.476     0.0446 0.843        0.405      1
## 5     1554807.            0       0.475     0.0767 0.843        0.405      1
## # i 59 more variables: Store2 <dbl>, Store3 <dbl>, Store4 <dbl>, Store5 <dbl>,
## #   Store6 <dbl>, Store7 <dbl>, Store8 <dbl>, Store9 <dbl>, Store10 <dbl>,
## #   Store11 <dbl>, Store12 <dbl>, Store13 <dbl>, Store14 <dbl>, Store15 <dbl>,
## #   Store16 <dbl>, Store17 <dbl>, Store18 <dbl>, Store19 <dbl>, Store20 <dbl>,
## #   Store21 <dbl>, Store22 <dbl>, Store23 <dbl>, Store24 <dbl>, Store25 <dbl>,
## #   Store26 <dbl>, Store27 <dbl>, Store28 <dbl>, Store29 <dbl>, Store30 <dbl>,
## #   Store31 <dbl>, Store32 <dbl>, Store33 <dbl>, Store34 <dbl>, ...
```

Linear Regression Model In linear regression we fit the data to a function to the form $y = \Sigma c_i f_i$

where t is the target variable and f_i are the feature variables. The coefficients c_i are the machine learning parameters which need to be optimized from the training data. I use the caret package to train a linear model to the training_data.

```r
fit_lm<-train(Weekly_Sales~.,method="lm",data=training_set)
summary(fit_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -642877  -63520   -3955   44432 1626560
##
## Coefficients: (3 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    552688     107349   5.149 2.72e-07 ***
## Holiday_Flag    18409       8431   2.184  0.02904 *
## Temperature     74831      38172   1.960  0.05000 .
## Fuel_Price     -28539      27745  -1.029  0.30371
## CPI            418563     170312   2.458  0.01402 *
## Unemployment  -367388      51306  -7.161 9.10e-13 ***
## Store1          598541      54493  10.984  < 2e-16 ***
## Store2          975626      53866  18.112  < 2e-16 ***
## Store3         -587087      60580  -9.691  < 2e-16 ***
## Store4         1448934      97736  14.825  < 2e-16 ***
## Store5         -687821      57457 -11.971  < 2e-16 ***
## Store6          547582      58609   9.343  < 2e-16 ***
## Store7         -236448      23492 -10.065  < 2e-16 ***
## Store8         -120727      61898  -1.950  0.05118 .
## Store9         -482968      62371  -7.743 1.15e-14 ***
## Store10        1322312      98427  13.434  < 2e-16 ***
## Store11         366031      60458   6.054 1.51e-09 ***
## Store12         609172     104462   5.832 5.81e-09 ***
## Store13        1396085      97802  14.275  < 2e-16 ***
## Store14        1227433      18736  65.513  < 2e-16 ***
## Store15          32738      87551   0.374  0.70847
## Store16        -367930      26100 -14.097  < 2e-16 ***
## Store17         283608      97976   2.895  0.00381 **
## Store18         521866      88365   5.906 3.72e-09 ***
## Store19         853147      87654   9.733  < 2e-16 ***
## Store20        1187923      44170  26.894  < 2e-16 ***
## Store21        -200631      53903  -3.722  0.00020 ***
## Store22         417993      81275   5.143 2.80e-07 ***
## Store23         692836      87201   7.945 2.34e-15 ***
## Store24         782445      87820   8.910  < 2e-16 ***
## Store25        -216295      44042  -4.911 9.32e-07 ***
## Store26         402612      87751   4.588 4.57e-06 ***
## Store27        1164080      81077  14.358  < 2e-16 ***
## Store28         922152     104424   8.831  < 2e-16 ***
## Store29           8618      89441   0.096  0.92324
## Store30        -518526      54073  -9.589  < 2e-16 ***
## Store31         439250      53875   8.153 4.37e-16 ***
## Store32         354175      22688  15.611  < 2e-16 ***
## Store33        -300989      98575  -3.053  0.00227 **
## Store34         463491     100530   4.610 4.11e-06 ***
## Store35         329315      81807   4.026 5.76e-05 ***
## Store36        -571438      52558 -10.873  < 2e-16 ***
## Store37        -426334      52456  -8.127 5.39e-16 ***
## Store38          -9999     104443  -0.096  0.92373
## Store39         507948      52420   9.690  < 2e-16 ***
## Store40         264435      87217   3.032  0.00244 **
## Store41         387333      24800  15.618  < 2e-16 ***
## Store42         -10618      98337  -0.108  0.91402
## Store43        -210362      40494  -5.195 2.12e-07 ***
## Store44        -310053      97844  -3.169  0.00154 **
## Store45             NA         NA      NA       NA
## Month2          121261      11374  10.661  < 2e-16 ***
## Month3           81784      13213   6.190 6.48e-10 ***
## Month4           89625      15432   5.808 6.69e-09 ***
## Month5           87857      17514   5.016 5.44e-07 ***
## Month6          107386      18769   5.721 1.11e-08 ***
## Month7           68150      19998   3.408  0.00066 ***
## Month8           84161      20123   4.182 2.93e-05 ***
## Month9           25976      18802   1.382  0.16716
## Month10          41764      16369   2.551  0.01075 *
## Month11        197837      15756  12.556  < 2e-16 ***
## Month12        350446      14558  24.072  < 2e-16 ***
## Month1             NA         NA      NA       NA
## Year2010        41303      21858   1.890  0.05886 .
## Year2011        22898      10409   2.200  0.02786 *
## Year2012           NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 144200 on 5404 degrees of freedom
## Multiple R-squared:  0.9359, Adjusted R-squared:  0.9351
## F-statistic:  1272 on 62 and 5404 DF,  p-value: < 2.2e-16
```

The summary function displays the optimized fit coefficients after the training.with their standard errors and t values. The t-statistic is the ratio of the estimate of the parameter to its standard error (assuming the null-hypothesis is model parameter = 0). As the t-statistic folows a student-t distribution, we can estimate the probablity that such a parameter could occur due to statistical fluctuations and it is mentioned in the Pr(>|t|) column. As lower Pr(>|t|) means that the non-zero value of the parameter is less likely to have arisen due to statistical fluctuations.The residual standard error is an estimate of the errors made by asssuming the functional form of Weekly_Sales as a linear function of the feature variables evaluated on the training set.

Now, that we have fitted the linear model on the training set we can use it to predict the Weekly_Sales from the the other variables in the validation set using the predict function

```
y_hat<-predict(fit_lm,testing_set)
```

We can find out the mean absolute error betweent the predicted and actual Weekly_Sales in the validation_set by using the MAE function (ref Introduction to Data Science by Rafael A. Irizarry)

```
MAE<-function(true_values,predicted_values){
  mean(abs(predicted_values - true_values))
}
```

```
mae_lm<-MAE(testing_set$Weekly_Sales,y_hat)
```

We see that the mean absolute error (MAE) is 81893.94. Similarly the root mean square deviation can be found by defining the RMSE function

```
RMSE <- function(true_values, predicted_values){
  sqrt(mean((true_values - predicted_values)^2))}
```

and using it to calulate the RMSE as

```
rmse_lm<-RMSE(testing_set$Weekly_Sales,y_hat)
```

which is calulated to be 133537.1. However, to get a sense of the model performance, we should calculate the ratio of the error to the mean of the Weekly_Sales, as scaling the Weekly_Sales would also scale up the errors in the prediction. This is known as the relative errors and can relative_mae and relative_rmse

```
rmse_lm<-RMSE(testing_set$Weekly_Sales,y_hat)
mean_weeklysales_val<-mean(testing_set$Weekly_Sales)
relative_mae_lm<-mae_lm/mean_weeklysales_val
relative_rmse_lm<-rmse_lm/mean_weeklysales_val

print(mae_lm)
```

```
## [1] 82794.47
```

```
print(relative_mae_lm)
```

```
## [1] 0.07893735
```

```
print(rmse_lm)
```

```
## [1] 128814
```

```
print(relative_rmse_lm)
```

```
## [1] 0.122813
```

We find the relative_mae is 0.07844218 and the relative_rmse is 0.1279086 which are both around 10%. Thus, we find that using the linear model and just a handful of information about the store and environment, we can predict the weekly sales to an accuracy of around 10% relative error which is quite amazing.

K-Nearest Neighbours Model Usually the data points which are close in the feature space also have the same target value, for example the phones having similar features have similar price. We can utilize this information to make a prediction using the k-nearest neightbors (knn) model which uses knnreg for continuous variables (https://github.com/topepo/caret/blob/master/models/files/knn.R). In this model, for a given featueset, k observations in the training data are found which have the smallest distance to the given feature set. Then the predicted value is given by the average of the target value of these k-data points in the training set. However, we do not know which value of k should provide the best result, therefore I train the knn model using k values form 1-30 in steps of 2 and obtain the RMSE and print the RMSE vs k (#Neighbors) in a line plot.

```
fit_knnreg<-train(Weekly_Sales~.,method="knn",data=training_set,metric="RMSE",tuneGrid = data.frame(k =
summary(fit_knnreg)
```

```
##             Length Class      Mode     
## learn        2     -none-     list     
## k            1     -none-     numeric  
## theDots      0     -none-     list     
## xNames      65     -none-     character
## problemType  1     -none-     character
## tuneValue    1     data.frame list     
## obsLevels    1     -none-     logical  
## param        0     -none-     list     
```

```r
print(fit_knnreg)
```

```
## k-Nearest Neighbors
##
## 5467 samples
##   65 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5467, 5467, 5467, 5467, 5467, 5467, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   386037.1  0.5620022  289933.2
##
## Tuning parameter 'k' was held constant at a value of 13
```

```r
fit_knnreg$bestTune
```

```
##    k
## 1 13
```

The k-value giving the lowest RMSE can be found using fit_knn$bestTune and is found to be 1 although there is a local minima of the RMSE at k=15. Comparing to the true values of the Weekly_Sales in the dataset the mae,relative_mae,rmse and relative_rmse can be found.

```r
y_hat<-predict(fit_knnreg,testing_set)
mae_knnreg<-MAE(testing_set$Weekly_Sales,y_hat)
relative_mae_knnreg<-mae_knnreg/mean_weeklysales_val
rmse_knnreg<-RMSE(testing_set$Weekly_Sales,y_hat)
relative_rmse_knnreg<-rmse_knnreg/mean_weeklysales_val

print(mae_knnreg)
```

```
## [1] 280701
```

```r
print(relative_mae_knnreg)
```

```
## [1] 0.2676241
```

```r
print(rmse_knnreg)
```

```
## [1] 363462.1
```

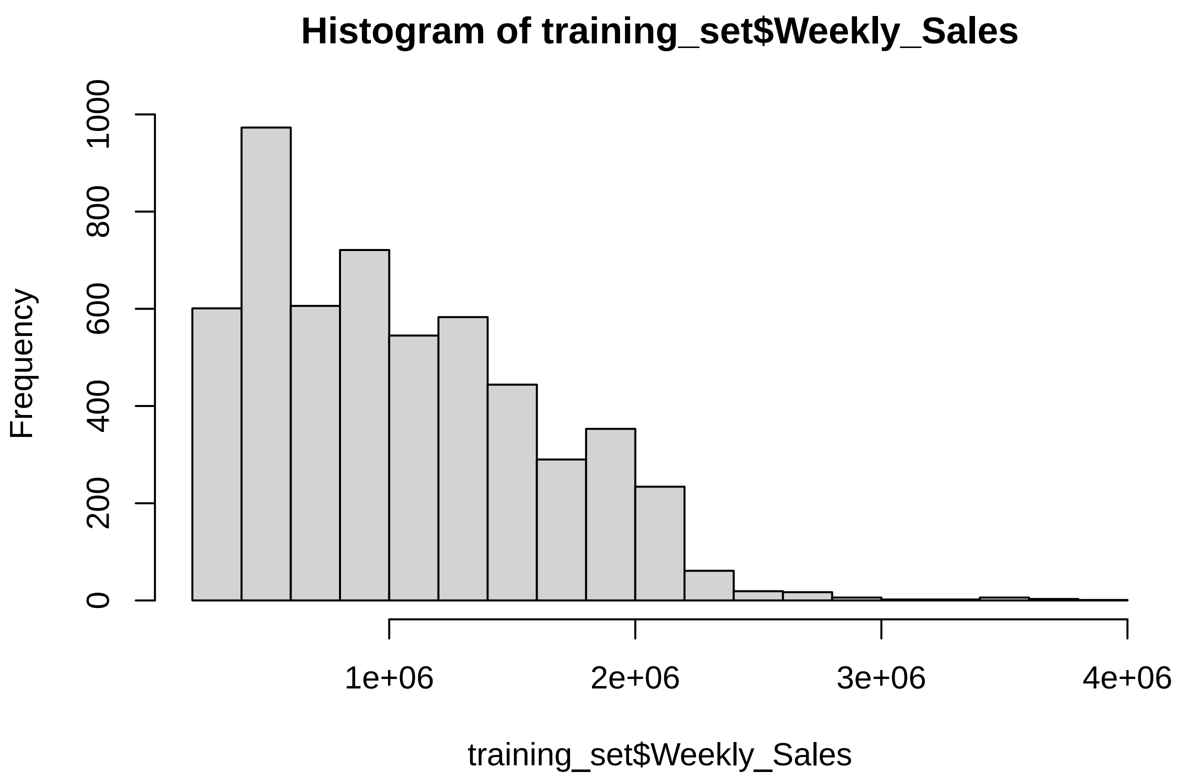
```r
print(relative_rmse_knnreg)
```

```
## [1] 0.3465297
```

We find the relative mae is 0.2638343 and the relative rmse is 0.3386724 which is slightly more than that from the linear model. It suggests that a linear model is not able to capture the effect of the feature variables on weekly sales but the average of 15 nearest neighbors provides a better estimate of the Weekly_Sales in this dataset.

Logistic Regression of the category of Weekly Sales Many a times, it is important to categorize the target variable into different classes ranging from its minimum to maximum value and use the Machine Learning model to output the cateogry into which the target variables should fall given its set of feature values. For example, the store might be interested in making its overall strategy based on weather a given set of conditions would tend to produce high, medium or low weekly_sales and optimize the conditions based on that. Therefore, in this section, I create 3 classes of the weekly sales using its maximum and minimum values and mutate it as the Category_Weekly_Sales column.

```
hist(training_set$Weekly_Sales)
```



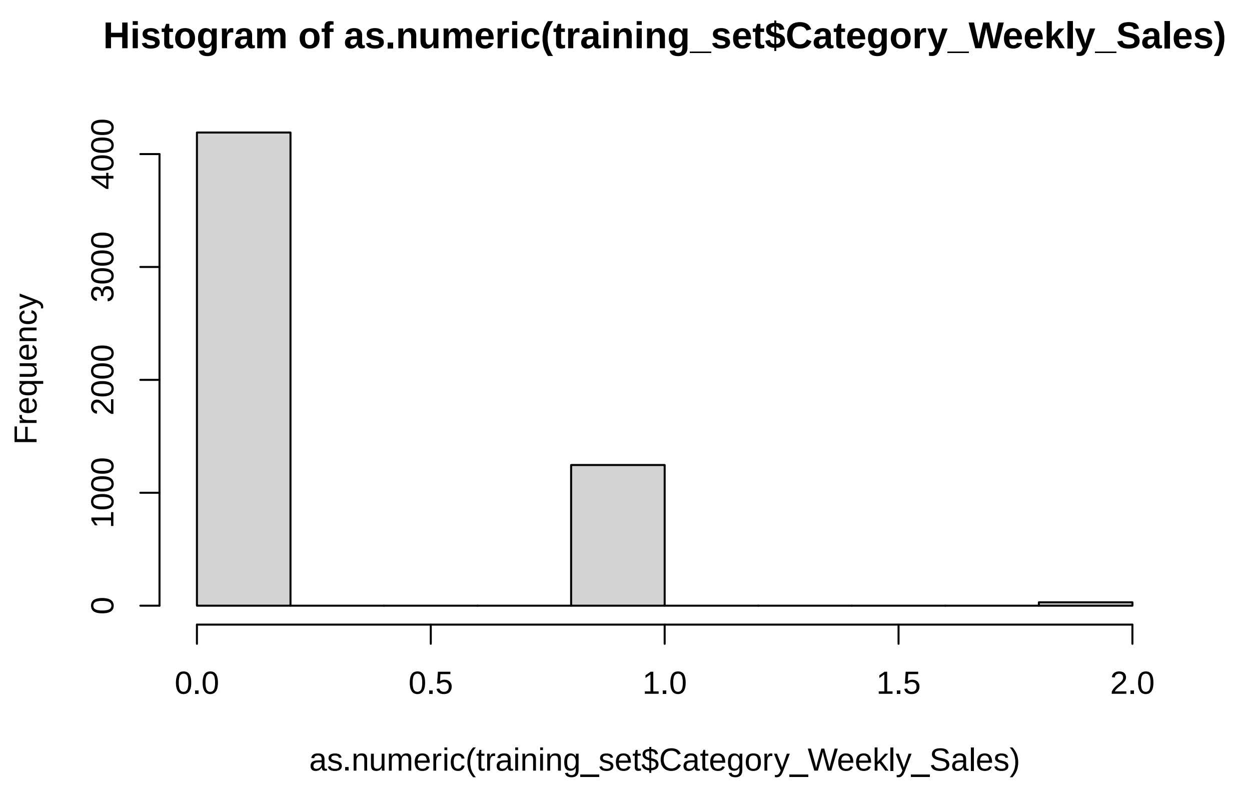
```
range(training_set$Weekly_Sales)
```

```
## [1]  209986.2 3818686.5
```

```
min_weekly_sales<-min(training_set$Weekly_Sales)
range_weekly_sales<-max(training_set$Weekly_Sales)-min(training_set$Weekly_Sales)

training_set<-training_set%>%mutate(Category_Weekly_Sales=floor((Weekly_Sales-min_weekly_sales)/range_we
testing_set<-testing_set%>%mutate(Category_Weekly_Sales=floor((Weekly_Sales-min_weekly_sales)/range_week
```

We can visualize the distribution of Category_Weekly_Sales by plotting its histogram and also by using the group_by function to show the number of occurences and the mean weekly sales in each of these categories. We observe that the categories 0, 1 and 2 contain 4193,1245 and 29 entries while the mean weekly sales in them are 798924, 1835041 and 3089619 respectively.

```
hist(as.numeric(training_set$Category_Weekly_Sales))
```



```
training_set%>%group_by(Category_Weekly_Sales)%>%summarise(n=n(), mean_weekly_sales=mean(Weekly_Sales))
```

```
## # A tibble: 3 x 3
##   Category_Weekly_Sales     n mean_weekly_sales
##                   <dbl> <int>             <dbl>
## 1                     0  4191           798267.
## 2                     1  1246          1832008.
## 3                     2    30          3123384.
```

Now, we can train a machine learning model to predict the category of Weekly_Sales from the other features (except Weekly_Sales as it was used to create the Category_Weekly_Sales so it should not be used as a feature variable). We can use knn, this time to be used for a classification task instead of the previous regression task. For that, we should first convert the Category_Weekly_Sales as factors an d then the knn model over it. As k=1 gave the lowest RMSE for regression, I have used k=1 although the model can be tested for different k values and the k value from the model with the lowest RMSE can be chosen for the final model.

```
training_set<-training_set%>%mutate(Category_Weekly_Sales=as.factor(Category_Weekly_Sales))
testing_set<-testing_set%>%mutate(Category_Weekly_Sales=as.factor(Category_Weekly_Sales))

fit_knncl<-train(Category_Weekly_Sales~.,method="knn",data=training_set[,-1],tuneGrid = data.frame(k =
summary(fit_knncl)
```

```
##             Length Class      Mode
## learn        2     -none-     list
## k            1     -none-     numeric
## theDots      0     -none-     list
## xNames      65     -none-     character
## problemType  1     -none-     character
## tuneValue    1     data.frame list
## obsLevels    3     -none-     character
## param        0     -none-     list
```

```
fit_knncl$bestTune
```

```
##   k
## 1 1
```

Now, with the fitted knn model with the best tune, we can test its performance over the testing set.

```
y_hat_knncl<-predict(fit_knncl,testing_set[,-1],type="raw")
confusionMatrix(y_hat_knncl, testing_set$Category_Weekly_Sales)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2
##          0 709  35   0
##          1  33 182   3
##          2   0   5   1
##
## Overall Statistics
##
##                Accuracy : 0.9215
##                  95% CI : (0.9027, 0.9376)
##     No Information Rate : 0.7665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7814
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity            0.9555   0.8198 0.250000
## Specificity            0.8451   0.9517 0.994813
## Pos Pred Value         0.9530   0.8349 0.166667
## Neg Pred Value         0.8527   0.9467 0.996881
## Prevalence             0.7665   0.2293 0.004132
## Detection Rate         0.7324   0.1880 0.001033
## Detection Prevalence   0.7686   0.2252 0.006198
## Balanced Accuracy      0.9003   0.8858 0.622407
```

We find that the knn model gives an accuracy of 0.9246. The confusion matrix shows the comparision between the reference and prediction values which should be a diagonal matrix in case of perfect prediction. The non-diagonal entries represent errors during the classification similar to the case when there are 2 classes in which false positive (when the prediction is positive and actual value is negative) and false_negatives (when the prediction is negative and actual value is positive) represent the errors made during the classification. Accurracy represents the ratio of number of instances which were correctly classified with the total number of instances.

Summary of Results

In this project we have successfully built a machine learning model to predict the Weekly_Sales from the store number and other features in the dataset. In the first model, a linear model was used to regress the value of weekly sales from other features which was trained over the training set. To evaluate the model perfromance two different metrics, the mae (Mean absolute error) and rmse (Root Mean Squared Error) were used which was 81893.94 and 133537.1 respectively for the linear regression model. To compare this error with the magnitude of the variable we are predicting, the relative_mae and relative_rmse which are the ratio of the mae and rmse to the mean of the Weekly_Sales were calculated and were found to be 0.07844218 and 0.1279086 respectively. Next a k-Nearest Neighbors model with regression mode was trained with different values of k to predict the weekly sales from other features. Although the RMSE showed a local minima at k=15, the lowest RMSE was found to be at k=1 which was chosen for the knn model. The mae, rmse, relative_mae and relative_rmse for the knn model were found to be 275444.1, 353575.3, 0.2638343 and 0.3386724 respectively. Next, the Weekly_Sales column was categorized into high, medium and low weekly sales and a knn model with classification mode was trained over the trainining set and used to predict the category of the Weekly_Sales in the testing set. The performance was visualized using the confusion matrix and accuracy and the accuracy was found to be 92.46%.

Conclusion

In this project, I have utilized the concepts of machine learning to build a model to predict the weekly sales of a store based on the store number and other conditions (Holiday, Temperature,Fuel_Price,CPI and Unemployment index). The dataset was first preprocessed, scaled and then partitioned into the training and testing sets. A linear model and a knn model with regression mode was then fitted over the training set and evaluated over the testing set. The linear model was found to have a better relatively mean absolute error and relative root mean squared error. The relative mean squared errors were different for different k which shows that hyperparameter tunning is important during training a machine learning model. Then, the Weekly_Sales was columns was categorized into high, medium and low weekly sales and a knn model was fit with classification to predict the cateogry of Weekly_Sales. It was found to have an accuracy of 89.26%. This model could be improved by using L1 and L2 type regularization and by using cross validation and decision trees for classification among other machine learning methods that can also be used and tested. Through this project, I have realized the importance of utilizing data to understand quantities of significance which can be used for optimizing the parameters at hand to achive the best possible value for a target variable and this model can be generalized to predict the weekly sales of other stores by preprocessing the features to the appropriate format.

References

1. https://www.kaggle.com/datasets/yasserh/walmart-dataset/data
2. Data Science: Capstone, HarvardX PH125.9x provided via the edX platform
3. Introduction to Data Science by Rafael A. Irizarry
4. https://stackoverflow.com/questions/74706268/how-to-create-dummy-variables-in-r-based-on-multiple-values-within-each-cell-in

5. https://www.geeksforgeeks.org/how-to-normalize-and-standardize-data-in-r/
6. https://stackoverflow.com/questions/62679940/geom-bar-of-named-number-vector