

# 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 from 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[2,3].

```
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)
set.seed(1, sample.kind="Rounding")
```

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.

```
walmart_dataset <- read.csv("/Users/aditya/Documents/Coursework/Online_Courses/Machine_Learning_and_Deep_Learning/walmart_dataset.csv")
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 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 the entries of the first 5 rows of walmart\_dataset.

```
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. 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.

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

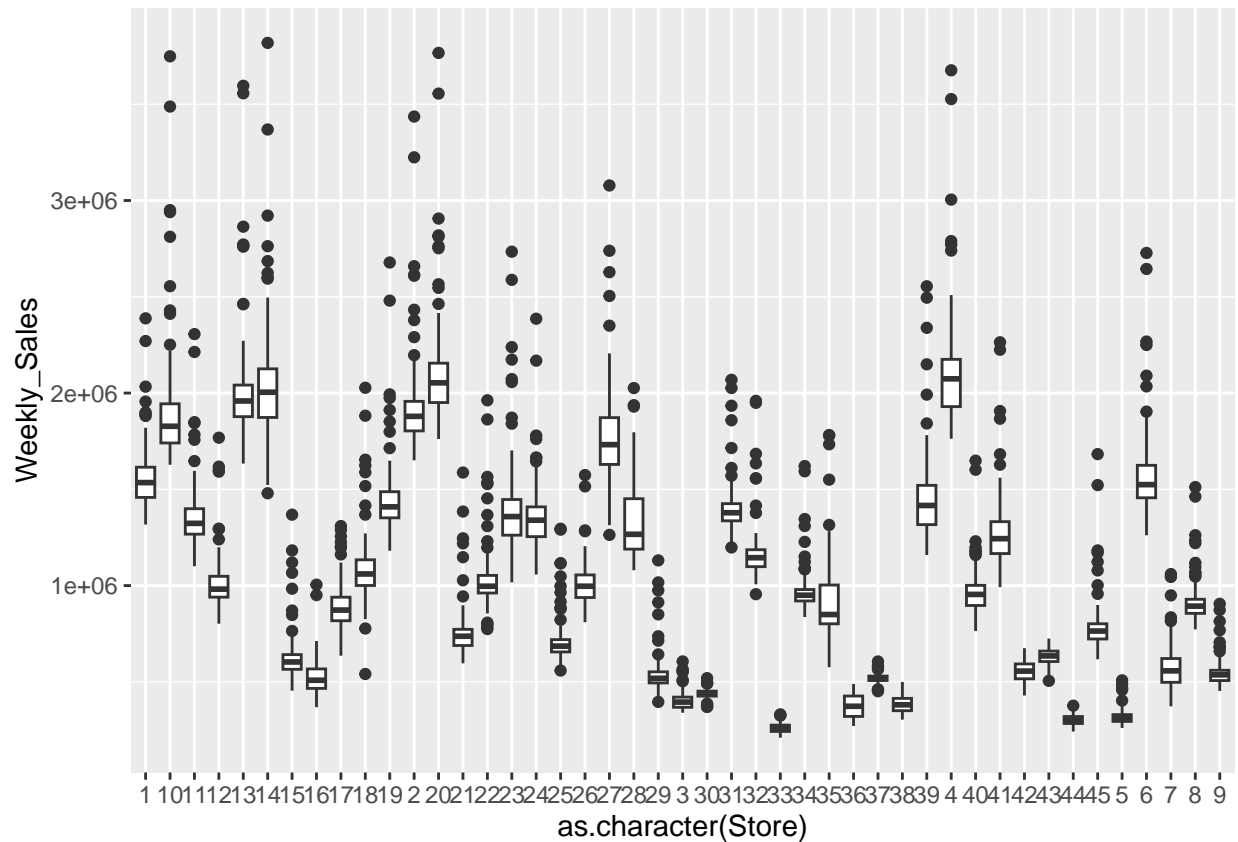
```
## [1] 209986.2 3818686.5
```

```
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
```

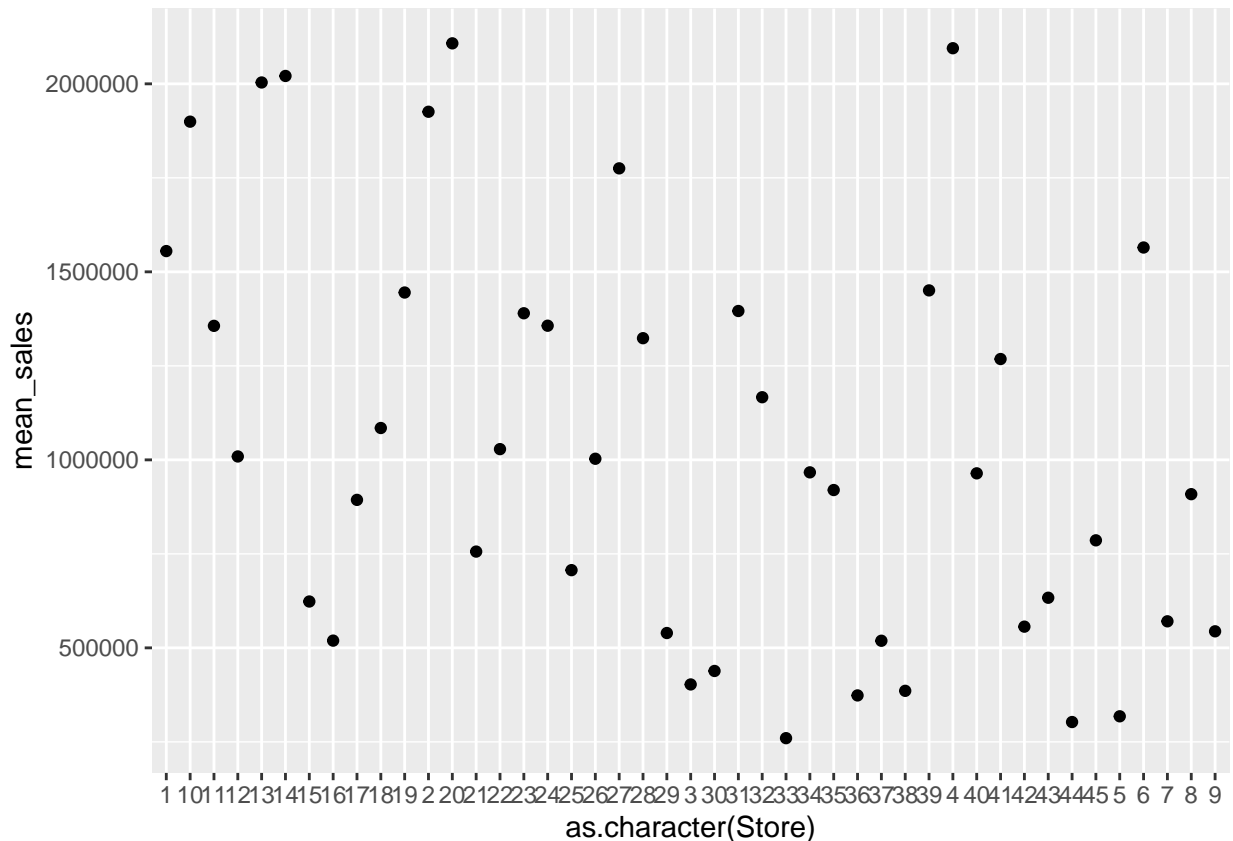
We can notice that there are 45 stores in the dataset with the store numbers ranging from 1 to 45. Some stores can have more sales than others due to location, total volume of products and other factors. Therefore, its useful to visualize the distribution of Weekly sales for different stores. For that, 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 hierarchy, we should use one-hot-encoding[4] 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 function to encode 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 = length)
```

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 use 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.

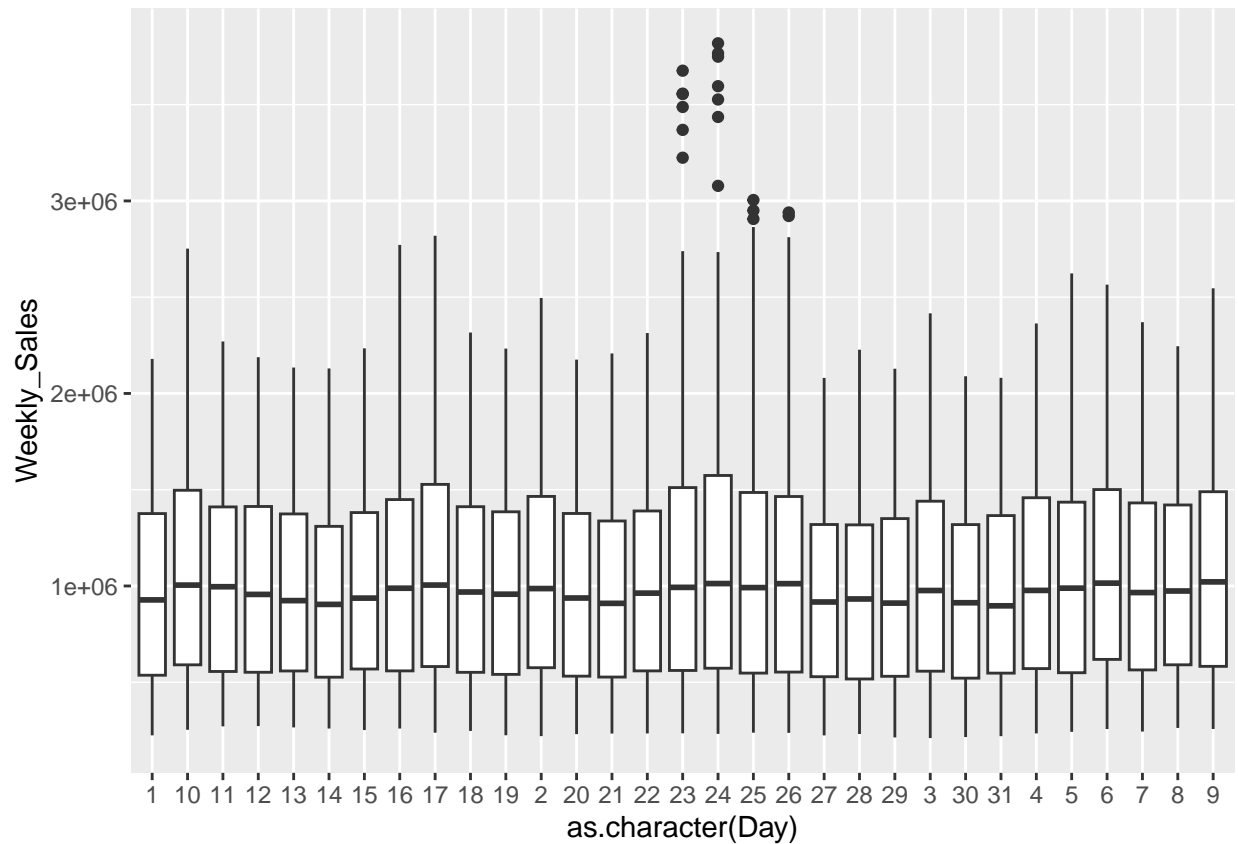
```
walmart_dataset<-walmart_dataset%>%mutate(Day=day(dmy(Date)),Month=month(dmy(Date)), Year=year(dmy(Date)))
head(walmart_dataset%>%select(c("Day", "Month", "Year")),5)
```

```
## # 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
```

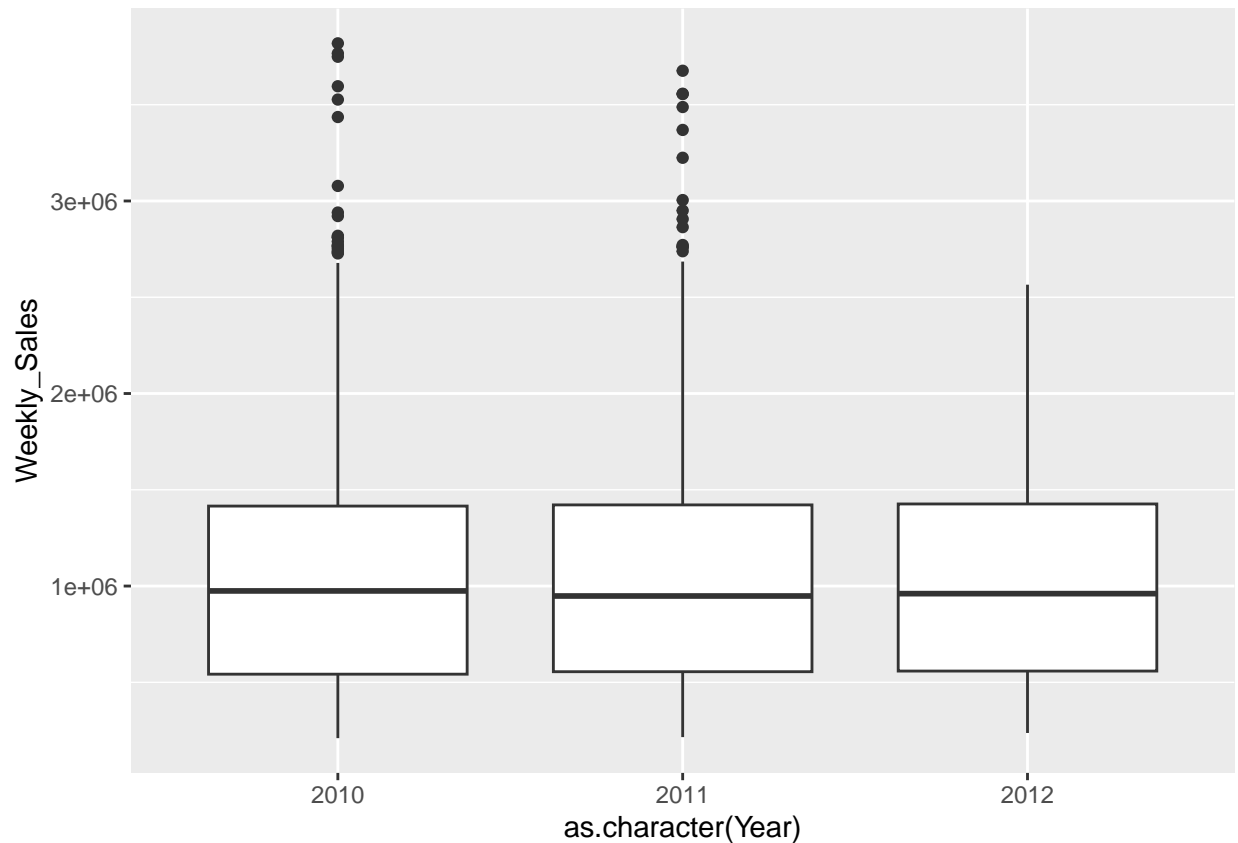
Now, we try to visualize the distribution of weekly sales accross day, month and year by creating 3 different boxplots corresponding to each of them.

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



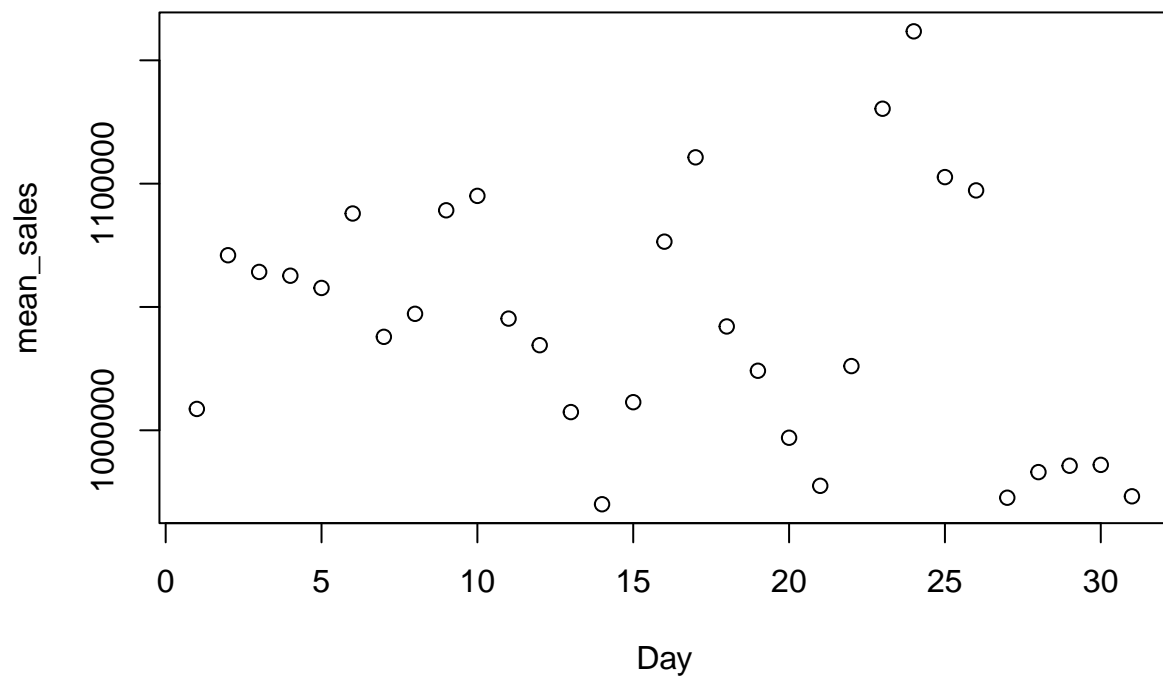
```
walmart_dataset%>% ggplot(aes(as.character(Month), Weekly_Sales)) + geom_boxplot()
```





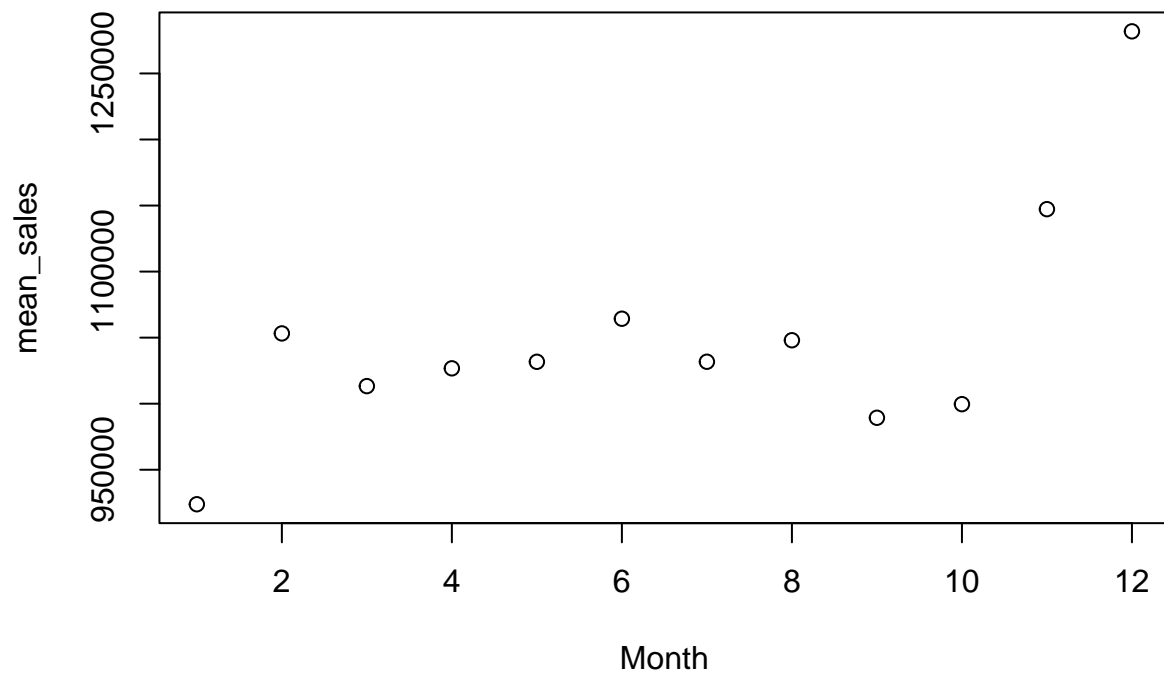
We see that the distribution of sales shows slight variation with day, month and year. Especially, the average sales is higher around december and 2010 had higher mean weekly sales than 2011 which was higher than that in 2012. To visualize the distribution of mean weekly sales, we can plot the scatter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

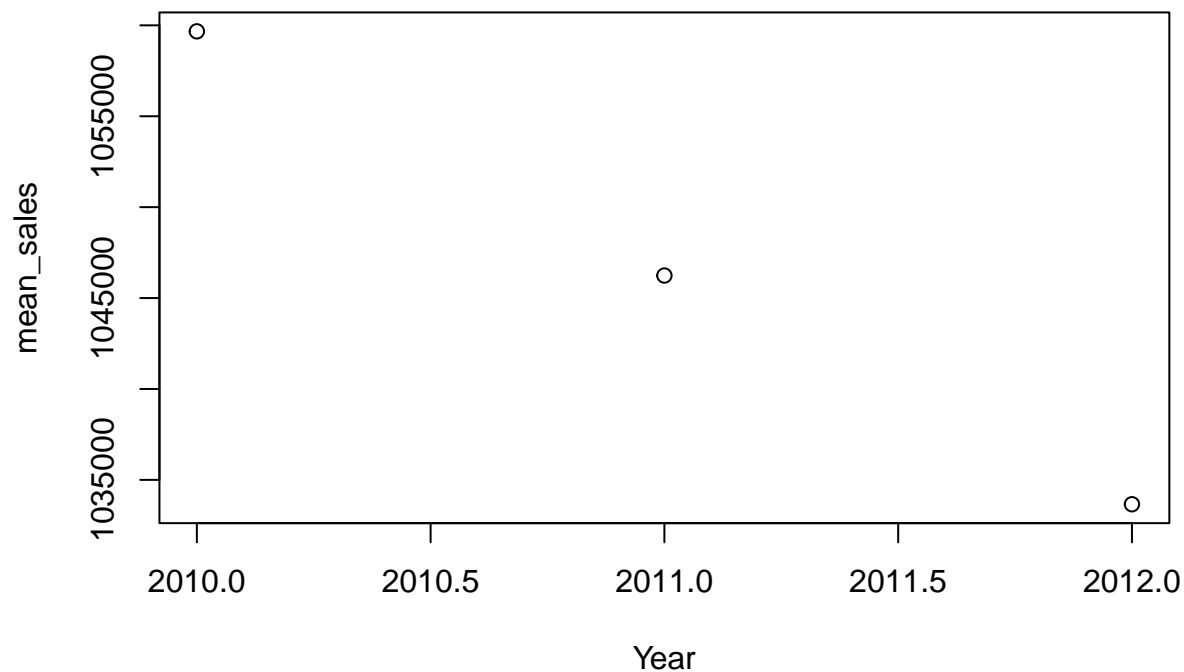


```
plot(walmart_dataset%>%group_by(Month)%>%summarise(mean_sales=mean(Weekly_Sales)))
```





```
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 corresponding 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 lowest in 2012.

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

```
## # 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_sales))
```

```
## # 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_sales))
```

```
## # 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 dependence 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 [4]. However, as we have 31 days, including each day as a feature can over-complicate 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 = Weekly_Sales)
walmart_dataset <- walmart_dataset %>% pivot_wider(names_from = Year, names_prefix="Year", values_from = Weekly_Sales)
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.

```
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 feature 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 deviation. 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[5] 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.

```
#Scaling the dataset
fMinMaxSclaer <- function(x) (
  (x - min(x)) / (max(x) - min(x))
)

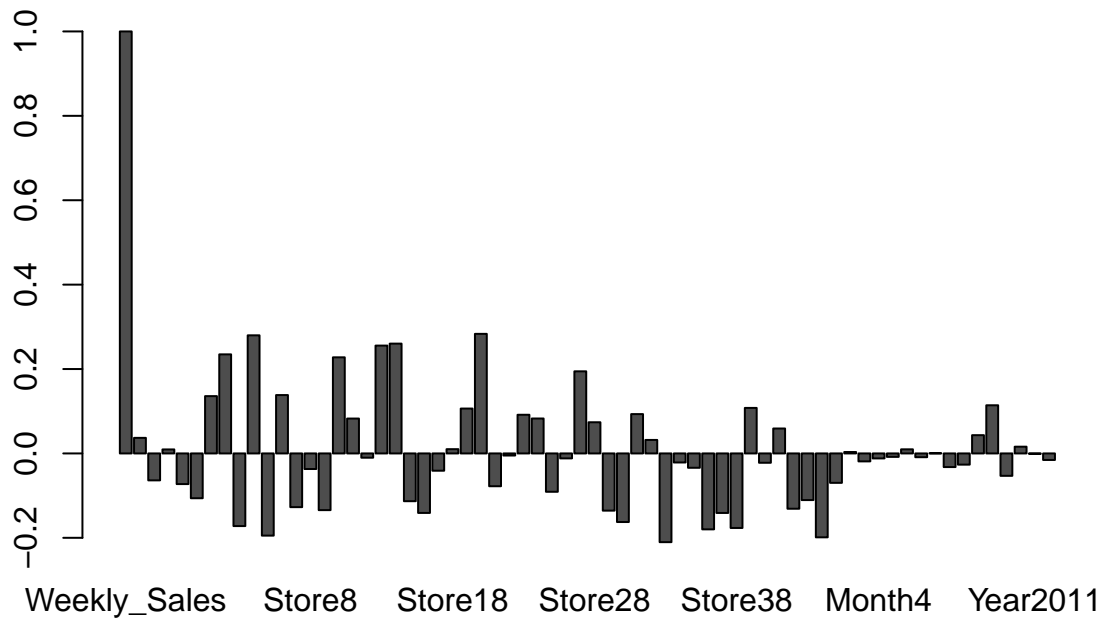
walmart_dataset[-1]<-as.data.frame(lapply(walmart_dataset[-1],fMinMaxSclaer))
head(walmart_dataset)
```

```
## # 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 [6] between the data of that feature variable with Weekly\_Sales.

```
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.

## 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 it.

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 [1,2]. 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.

```
nrow(training_set)
```

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

```
nrow(testing_set)
```

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

```
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    1611968.           0         0.411     0.0210 0.842         0.405     1
## 3    1409728.           0         0.476     0.0446 0.843         0.405     1
## 4    1554807.           0         0.475     0.0767 0.843         0.405     1
## 5    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>, ...
```

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

where  $y$  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 [7] to train a linear model to the training\_data.

```
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
## -675097  -61415   -3875   45598 1610769
##
## Coefficients: (3 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  561736.5    105915.5   5.304 1.18e-07 ***
## Holiday_Flag  43720.1      8401.7   5.204 2.03e-07 ***
## Temperature  77738.1     37482.6   2.074 0.038128 *
## Fuel_Price   -770.3       27289.2  -0.028 0.977483
## CPI          404042.9    168184.3   2.402 0.016322 *
## Unemployment -398625.2     50148.2  -7.949 2.27e-15 ***
```

## Store1	596899.8	53843.8	11.086	< 2e-16	***
## Store2	977376.0	53070.2	18.417	< 2e-16	***
## Store3	-584930.6	59448.7	-9.839	< 2e-16	***
## Store4	1427481.7	96614.0	14.775	< 2e-16	***
## Store5	-690654.8	56495.3	-12.225	< 2e-16	***
## Store6	558052.4	57459.7	9.712	< 2e-16	***
## Store7	-233821.3	23044.6	-10.146	< 2e-16	***
## Store8	-118660.8	61191.2	-1.939	0.052531	.
## Store9	-485018.9	61678.9	-7.864	4.47e-15	***
## Store10	1323906.0	97325.6	13.603	< 2e-16	***
## Store11	370194.0	59597.6	6.212	5.64e-10	***
## Store12	611223.5	103072.3	5.930	3.22e-09	***
## Store13	1383298.6	96754.7	14.297	< 2e-16	***
## Store14	1228860.5	18484.2	66.482	< 2e-16	***
## Store15	16223.9	86352.8	0.188	0.850979	
## Store16	-372659.8	25688.2	-14.507	< 2e-16	***
## Store17	266967.6	96641.8	2.762	0.005756	**
## Store18	507257.4	87188.7	5.818	6.30e-09	***
## Store19	839403.2	86247.7	9.732	< 2e-16	***
## Store20	1183078.5	43591.0	27.140	< 2e-16	***
## Store21	-192415.2	53250.4	-3.613	0.000305	***
## Store22	410639.6	80340.5	5.111	3.31e-07	***
## Store23	668384.3	86057.3	7.767	9.57e-15	***
## Store24	760882.6	86648.6	8.781	< 2e-16	***
## Store25	-218545.7	43444.8	-5.030	5.05e-07	***
## Store26	402200.4	86595.8	4.645	3.49e-06	***
## Store27	1154210.3	80082.8	14.413	< 2e-16	***
## Store28	931119.5	103146.5	9.027	< 2e-16	***
## Store29	-470.3	88190.5	-0.005	0.995745	
## Store30	-516198.0	53374.8	-9.671	< 2e-16	***
## Store31	435997.1	53245.3	8.188	3.27e-16	***
## Store32	354125.9	22294.7	15.884	< 2e-16	***
## Store33	-317576.8	97446.2	-3.259	0.001125	**
## Store34	456534.8	99141.5	4.605	4.22e-06	***
## Store35	314702.4	80761.1	3.897	9.87e-05	***
## Store36	-568359.2	51564.5	-11.022	< 2e-16	***
## Store37	-422490.7	51469.7	-8.209	2.77e-16	***
## Store38	-10083.7	103064.7	-0.098	0.922064	
## Store39	514155.8	51583.0	9.968	< 2e-16	***
## Store40	241492.0	85907.7	2.811	0.004956	**
## Store41	399747.6	24581.3	16.262	< 2e-16	***
## Store42	-23378.3	97087.1	-0.241	0.809721	
## Store43	-196799.6	39820.0	-4.942	7.96e-07	***
## Store44	-325291.8	96734.0	-3.363	0.000777	***
## Store45	NA	NA	NA	NA	
## Month2	116742.5	11279.4	10.350	< 2e-16	***
## Month3	77420.4	13008.0	5.952	2.82e-09	***
## Month4	81232.0	15267.5	5.321	1.08e-07	***
## Month5	77455.5	17238.1	4.493	7.16e-06	***
## Month6	103913.3	18522.5	5.610	2.12e-08	***
## Month7	64315.3	19675.1	3.269	0.001087	**
## Month8	78554.3	19780.6	3.971	7.24e-05	***
## Month9	15774.2	18430.2	0.856	0.392099	
## Month10	36154.0	16102.5	2.245	0.024793	*

```
## Month11      197201.0    15612.9   12.631 < 2e-16 ***
## Month12      343262.9    14287.5   24.025 < 2e-16 ***
## Month1        NA         NA        NA        NA
## Year2010      56795.3    21422.2    2.651 0.008043 **
## Year2011      23136.4    10212.2    2.266 0.023517 *
## Year2012        NA         NA        NA        NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 142400 on 5404 degrees of freedom
## Multiple R-squared:  0.9373, Adjusted R-squared:  0.9366
## F-statistic: 1303 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 follows a student-t distribution, we can estimate the probability that such a parameter could occur due to statistical fluctuations and it is mentioned in the  $\text{Pr}(>|t|)$  column. As lower  $\text{Pr}(>|t|)$  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 assuming the functional form of Weekly\_Sales as a linear function of the feature variables.

Now, that we have fitted the linear model on the training set we can use it to predict the Weekly\_Sales from 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 between the predicted and actual Weekly\_Sales in the validation\_set by using the MAE function[3]

```
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 85775.22. Similarly the root mean square deviation can be found by defining the RMSE function[3]

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

```

and using it to calculate the RMSE as

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

which is calculated to be 139558.1. 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 be calculated using 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] 85775.22
```

```
print(relative_mae_lm)
```

```
## [1] 0.08226522
```

```
print(rmse_lm)
```

```
## [1] 139558.1
```

```
print(relative_rmse_lm)
```

```
## [1] 0.1338473
```

We find the relative\_mae is 0.08226522 and the relative\_rmse is 0.1338473 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 Neighbors 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 neighbors (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 feature set, 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 from 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
```

```
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 across tuning parameters:
##
##  k    RMSE      Rsquared    MAE
##  1 266704.5  0.7886575 122742.3
##  3 336673.3  0.6498059 195755.8
##  5 370695.1  0.5677560 247654.9
##  7 383340.3  0.5404550 271303.7
##  9 386369.9  0.5391055 280999.7
## 11 386755.4  0.5449030 285759.2
## 13 385872.4  0.5540074 288219.0
## 15 383822.8  0.5661387 288946.1
## 17 382288.2  0.5768737 289465.6
## 19 380827.0  0.5873379 289679.3
## 21 379620.5  0.5966719 289890.2
## 23 379582.6  0.6028080 290967.3
## 25 379499.0  0.6096647 292160.9
## 27 379818.1  0.6143320 293496.5
## 29 380460.2  0.6189726 295140.6
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 1.
```

```
fit_knnreg$bestTune
```

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

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  $k=25$ . This value of  $k$  is used in the knn model. Comparing to the true values of the `Weekly_Sales` in the dataset the `mae`, `relative_mae`, `rmse` and `relative_rmse` can be found.

```
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] 88463.33
```

```
print(relative_mae_knnreg)
```

```
## [1] 0.08484333
```

```
print(rmse_knnreg)
```

```
## [1] 182078.2
```

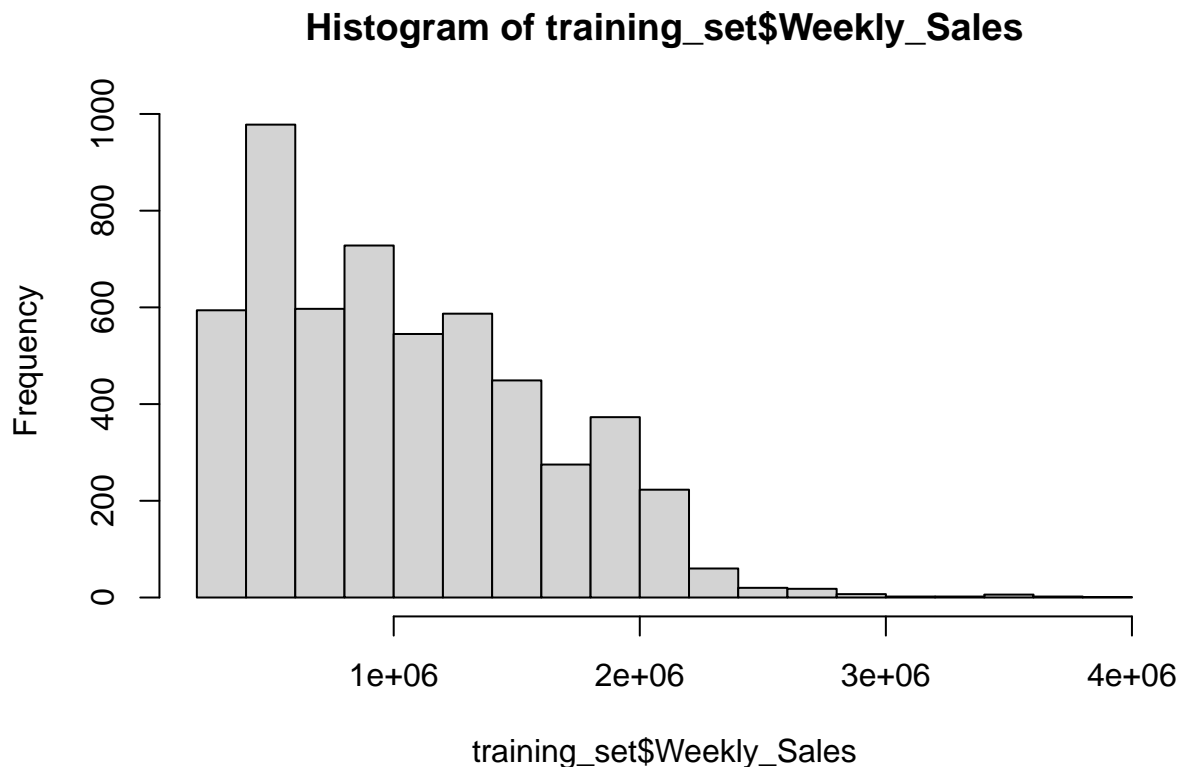
```
print(relative_rmse_knnreg)
```

```
## [1] 0.1746274
```

We find the relative mae is 0.08484333 and the relative rmse is 0.1746274 which is slightly more than that from the linear model. It suggests that a linear model is not able to capture all the effect of the feature variables on weekly sales but it still provides an estimate with similar accuracy to the linear model.

**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 category 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)
```

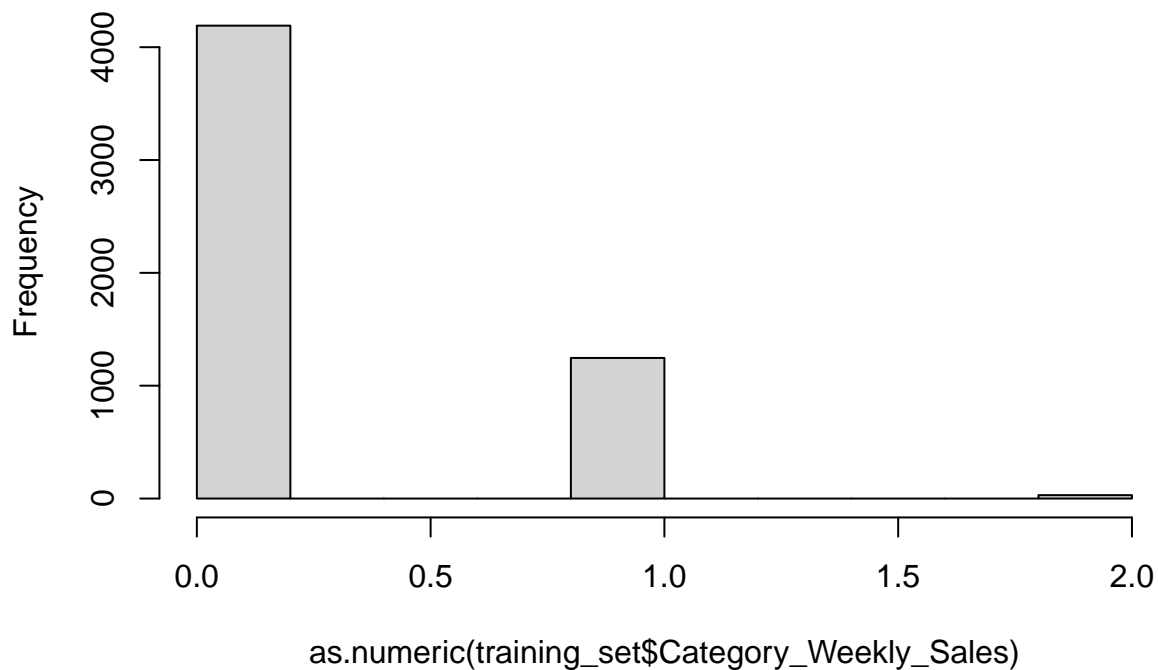
```
training_set<-training_set%>%mutate(Category_Weekly_Sales=floor((Weekly_Sales-min_weekly_sales)/range_w
```

```
testing_set<-testing_set%>%mutate(Category_Weekly_Sales=floor((Weekly_Sales-min_weekly_sales)/range_w
```

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 occurrences and the mean weekly sales in each of these categories. We observe that the categories 0, 1 and 2 contain 4191, 1246 and 30 entries while the mean weekly sales in them are 799742, 1832458 and 3098531 respectively.

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

## Histogram of 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          799742.
```

```
## 2             1   1246          1832458.
```

```
## 3             2     30          3098531.
```

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 and then train 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 = 1:10))
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
```

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
##           0 713  32    0
##           1  29 184    1
##           2   0   6    3
##
## Overall Statistics
##
##           Accuracy : 0.9298
##           95% CI : (0.9118, 0.945)
##           No Information Rate : 0.7665
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.8045
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: 0 Class: 1 Class: 2
## Sensitivity      0.9609   0.8288 0.750000
## Specificity      0.8584   0.9598 0.993776
```

## Pos Pred Value	0.9570	0.8598	0.333333
## Neg Pred Value	0.8700	0.9496	0.998957
## Prevalence	0.7665	0.2293	0.004132
## Detection Rate	0.7366	0.1901	0.003099
## Detection Prevalence	0.7696	0.2211	0.009298
## Balanced Accuracy	0.9097	0.8943	0.871888

The confusion matrix shows the comparison 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. Accuracy represents the ratio of number of instances which were correctly classified with the total number of instances. We find that the knn model gives an accuracy of 92.98%.

## Summary of Results

In this project I 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 performance two different metrics, the mae (Mean absolute error) and rmse (Root Mean Squared Error) were used which were found to be 85775.22 and 139558.1 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 0.08226522 and 0.1338473 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=25, 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 88463.33, 182078.2, 0.08484333 and 0.1746274 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 training set and used to predict the category of the Weekly\_Sales in the testing set. The performance was visualized using the confusion matrix and the accuracy of the classification model was found to be 92.98%.

## 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 were 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 in the knn model which shows that hyperparameter tuning is important during training a machine learning model. To estimate the overall nature of Weekly\_Sales, 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 category of Weekly\_Sales. It was found to have a classification accuracy of 92.98%. 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 and predict quantities of interest from other features. This can be used in many practical applications for optimizing the parameters at hand (in this case to choose the location based on the feature parameters ) to achieve the best possible value for a target variable (weekly sales of the store). This model can be generalized to predict the weekly sales of other stores by preprocessing the features of that store to the appropriate format used in the models in this project.

## 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>
7. Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. *Journal of Statistical Software*, 28(5), 1–26. <https://doi.org/10.18637/jss.v028.i05>