Capstone Project - Walmart Store Sales Forecast

Vincenzo Grifo

20/06/2020

INTRODUCTION

Goal of the project

The goal of this project was to build a machine learning model to predict future sales of 45 Walmart stores located in different regions, given their historical sales. This project was part of the final course in the HarvardX Professional Certificate in Data Science.

Datasets description

The dataset available to train the model, called *train*, was a collection of weekly sales data for each of the 45 Walmart stores, which covered sales per department registered from the 5^{th} February 2010 to the 1^{st} November 2012. More specifically, the dataset included information about 5 variables (columns):

- **train** dataset
  - *Store* - the store number
  - *Dept* - the department number
  - *Date* - the week
  - *Weekly_Sales* - sales for the given department in the given store and department
  - *IsHoliday* - whether the week was a special holiday week

Additional data related to the store, department, and regional promotional activity for the given dates were available in the dataset *features*. The variables included in this dataset are shown below.

- **features** dataset
  - *Store* - the store number
  - *Date* - the week
  - *Temperature*- average temperature in the region
  - *Fuel Price* - cost of fuel in the region
  - *MarkDown 1-5* - anonymised data related to promotional markdowns that Walmart was running
  - *CPI* - consumer price index
  - *Unemployment* - the unemployment rate
  - *IsHoliday* - whether the week was a special holiday week

Anonymised information about the 45 stores, indicating the type and size of store, were also available in the dataset *stores*.

- **stores** dataset
  - *Store* - the store number
  - *Type* - type of the store (A, B, C)
  - *Size* - size of the store

Key steps to build the machine learning model

The supervised machine learning algorithm was implemented following 5 key steps.

1. First, the data were retrieved by downloading a database from a public GitHub repository; this dataset had been previously downloaded from the Kaggle website. Then, the the training and test datasets that would have been used in the next steps were created.
2. After that, the data in the training set were explored through visualisation and summarisation techniques to understand which features could have had predictive power and, therefore, should have been considered in the algorithm.
3. Next, the dataset were processed to include the new features that had been engineered and selected during the data exploration and visualisation stages. Furthermore, during this stage any missing values was treated and the multiple datesets were joined into one.
4. In the next phase, five different machine learning models have been implemented and compared. These included a machine learning techniques that went beyond standard linear regression:
   - a 3-steps imputational algorithm;
   - an algorithm which combined the results of thousands linear models;
   - an approach based on the k-nearest neighbours (KNN) algorithm;
   - a Random Forest model;
   - an ARIMA based algorithm.
5. Finally, the performance of those model were compared and discussed. A final model was chosen based on the evaluation of the error functions.

ANALYSIS

Retrieving the data

The first step in building the machine learning model consisted in retrieving the datasets. The data were obtained from the public Kaggle competition *Walmart Recruiting - Store Sales Forecasting* (url: https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/overview). As to download the datasets a Kaggle account is needed, to facilitate the public evaluation of the project these datasets were uploaded in a public repository in *GitHub* - thus the R script could fetch the data in a easier and more direct way.

Splitting the data into *training* and *test* sets

After having retrieved the data, the dataset containing the weekly sales was split into two complementary datasets - the *training* and *test* sets. The former was the subset that was going to be used to train the algorithms (tuning the models' parameters), while the latter was the subset that was going to be used to measure the accuracy of the models and therefore select the final machine learning model.

When splitting the training and test sets, the proportion of the data split is usually arbitrary. In most cases, the larger the *training set* the better, because generally adding more examples adds diversity within the data, thus improving the fit and generalisation of the model. However, at the same time, the *test set* in order to to yield statistically meaningful results must be large enough and representative of the *train* data set as a whole. For this reason the *train* dataset was split randomly with a 90-10 train-test ratio.

```r
# --- Splitting the edx dataset into TEST and TRAINING set ---

set.seed(17, sample.kind="Rounding")

# The test set will be 10% of the training set
test_index <- createDataPartition(y = train$Weekly_Sales, times = 1, p = 0.1, list = FALSE)
# Converting train from a tibble to a dataframe
train <- as.data.frame(train)
train_set <- train[-test_index,]
test_set <- train[test_index,]
# Removing temporary files from the working environment
rm(test_index)
```

Data Exploration

The newly created test dataset *test_set* was used for the exploratory data analysis. This stage consisted of the initial investigation on the data so as to discover relationships and attributes present in the dataset, to formulate and check any assumptions, with the help of summary statistics and graphical representations. This was a key process as it helped gain precious insights on the predictive power of the variables and therefore enabled the formulation of the models that were built and compared in the subsequent stages.

In order to get a first grasp of the data, the initial investigations involved the analysis of the structure of the datasets and the visualisation of the first few rows.

```r
# --- Features dataset ---
# Visualising the structure of the dataset
str(features)
```

```
## tibble [8,190 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Store       : num [1:8190] 1 1 1 1 1 1 1 1 1 1 ...
##  $ Date        : Date[1:8190], format: "2010-02-05" "2010-02-12" ...
##  $ Temperature : num [1:8190] 42.3 38.5 39.9 46.6 46.5 ...
##  $ Fuel_Price  : num [1:8190] 2.57 2.55 2.51 2.56 2.62 ...
##  $ MarkDown1   : num [1:8190] NA NA NA NA NA NA NA NA NA NA ...
##  $ MarkDown2   : num [1:8190] NA NA NA NA NA NA NA NA NA NA ...
##  $ MarkDown3   : num [1:8190] NA NA NA NA NA NA NA NA NA NA ...
##  $ MarkDown4   : num [1:8190] NA NA NA NA NA NA NA NA NA NA ...
##  $ MarkDown5   : num [1:8190] NA NA NA NA NA NA NA NA NA NA ...
##  $ CPI         : num [1:8190] 211 211 211 211 211 ...
##  $ Unemployment: num [1:8190] 8.11 8.11 8.11 8.11 8.11 ...
##  $ IsHoliday   : logi [1:8190] FALSE TRUE FALSE FALSE FALSE FALSE ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Store = col_double(),
##   ..   Date = col_date(format = ""),
##   ..   Temperature = col_double(),
##   ..   Fuel_Price = col_double(),
##   ..   MarkDown1 = col_double(),
##   ..   MarkDown2 = col_double(),
##   ..   MarkDown3 = col_double(),
##   ..   MarkDown4 = col_double(),
##   ..   MarkDown5 = col_double(),
##   ..   CPI = col_double(),
##   ..   Unemployment = col_double(),
##   ..   IsHoliday = col_logical()
##   .. )
```

```
# Visualising the first (and last) few rows of the dataset
head(features)
```

```
## # A tibble: 6 x 12
##   Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##   <dbl> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
## 1     1 2010-02-05        42.3       2.57        NA        NA        NA
## 2     1 2010-02-12        38.5       2.55        NA        NA        NA
## 3     1 2010-02-19        39.9       2.51        NA        NA        NA
## 4     1 2010-02-26        46.6       2.56        NA        NA        NA
## 5     1 2010-03-05        46.5       2.62        NA        NA        NA
## 6     1 2010-03-12        57.8       2.67        NA        NA        NA
## # ... with 5 more variables: MarkDown4 <dbl>, MarkDown5 <dbl>, CPI <dbl>,
## #   Unemployment <dbl>, IsHoliday <lgl>
```

```
tail(features)
```

```
## # A tibble: 6 x 12
##   Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##   <dbl> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
## 1    45 2013-06-21        70.1       3.63     4989.      385.    179.
## 2    45 2013-06-28        76.0       3.64     4842.      975.      3
## 3    45 2013-07-05        77.5       3.61     9090.     2269.    583.
## 4    45 2013-07-12        79.4       3.61     3790.     1827.     85.7
## 5    45 2013-07-19        82.8       3.74     2961.     1047.    204.
## 6    45 2013-07-26        76.1       3.80      212.      852.      2.06
## # ... with 5 more variables: MarkDown4 <dbl>, MarkDown5 <dbl>, CPI <dbl>,
## #   Unemployment <dbl>, IsHoliday <lgl>
```

```
# Checking if there are any NA data
sum(is.na(features))
```

```
## [1] 24040
```

```
# Counting NA values due to missing MarkData
sum(is.na(features[,c(5:9)]))
```

```
## [1] 22870
```

```
# We can verify this with the following code
sum(is.na(features[,c(5:11)]))
```

```
## [1] 24040
```

```r
# Identifying CPI NAs
na_cpi <-features[(is.na(features$CPI)),]
na_cpi %>%
  group_by(Date) %>%
  summarise(n())
```

```
## # A tibble: 13 x 2
##    Date       `n()`
##    <date>     <int>
##  1 2013-05-03    45
##  2 2013-05-10    45
##  3 2013-05-17    45
##  4 2013-05-24    45
##  5 2013-05-31    45
##  6 2013-06-07    45
##  7 2013-06-14    45
##  8 2013-06-21    45
##  9 2013-06-28    45
## 10 2013-07-05    45
## 11 2013-07-12    45
## 12 2013-07-19    45
## 13 2013-07-26    45
```

```r
# Identifying Unemployment NAs
na_unmp <- features[(is.na(features$Unemployment)),]
na_unmp %>%
  group_by(Date) %>%
  summarise(n())
```

```
## # A tibble: 13 x 2
##    Date       `n()`
##    <date>     <int>
##  1 2013-05-03    45
##  2 2013-05-10    45
##  3 2013-05-17    45
##  4 2013-05-24    45
##  5 2013-05-31    45
##  6 2013-06-07    45
##  7 2013-06-14    45
##  8 2013-06-21    45
##  9 2013-06-28    45
## 10 2013-07-05    45
## 11 2013-07-12    45
## 12 2013-07-19    45
## 13 2013-07-26    45
```

```r
# -> The dates that are have NAs values are the weeks from 03rd May 2013 to 26th July 2013

# Checking that CPI NAs and Unemployment NAs are in the exact same rows
sum(na_cpi[,c(1:2)]!=na_unmp[,c(1:2)])
```

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

```r
# -> the dates where there are NAs in CPI also got NAs in Unemployment

# Removing temporary datasets
rm(na_cpi,na_unmp)

# --- Stores dataset ---
# Visualising the structure of the dataset
str(stores)
```

```
## tibble [45 x 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Store: num [1:45] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Type : chr [1:45] "A" "A" "B" "A" ...
##  $ Size : num [1:45] 151315 202307 37392 205863 34875 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Store = col_double(),
##   ..   Type = col_character(),
##   ..   Size = col_double()
##   .. )
```

```r
# Visualising the first (and last) few rows of the dataset
head(stores)
```

```
## # A tibble: 6 x 3
##   Store Type    Size
##   <dbl> <chr>  <dbl>
## 1     1 A     151315
## 2     2 A     202307
## 3     3 B      37392
## 4     4 A     205863
## 5     5 B      34875
## 6     6 A     202505
```

```r
tail(stores)
```

```
## # A tibble: 6 x 3
##   Store Type    Size
##   <dbl> <chr>  <dbl>
## 1    40 A     155083
## 2    41 A     196321
## 3    42 C      39690
## 4    43 C      41062
## 5    44 C      39910
## 6    45 B     118221
```

```r
# Checking if there are any NA data
sum(is.na(stores))
```

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

```r
# Checking store type distibution
summary(stores$Type)
```

```
##    Length     Class      Mode 
##        45 character character 
```

```r
# --- Train set ---
# Visualising the structure of the dataset
str(train_set)
```

```
## 'data.frame':    379410 obs. of  5 variables:
##  $ Store       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Dept        : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Date        : Date, format: "2010-02-05" "2010-02-12" ...
##  $ Weekly_Sales: num  24924 46039 41596 19404 21043 ...
##  $ IsHoliday   : logi  FALSE TRUE FALSE FALSE FALSE FALSE ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Store = col_double(),
##   ..   Dept = col_double(),
##   ..   Date = col_date(format = ""),
##   ..   Weekly_Sales = col_double(),
##   ..   IsHoliday = col_logical()
##   .. )
```

```r
# Visualising the first (and last) few rows of the dataset
head(train_set)
```

```
##   Store Dept       Date Weekly_Sales IsHoliday
## 1     1    1 2010-02-05     24924.50     FALSE
## 2     1    1 2010-02-12     46039.49      TRUE
## 3     1    1 2010-02-19     41595.55     FALSE
## 4     1    1 2010-02-26     19403.54     FALSE
## 6     1    1 2010-03-12     21043.39     FALSE
## 7     1    1 2010-03-19     22136.64     FALSE
```

```r
tail(train_set)
```

```
##        Store Dept       Date Weekly_Sales IsHoliday
## 421565    45   98 2012-09-21       467.30     FALSE
## 421566    45   98 2012-09-28       508.37     FALSE
## 421567    45   98 2012-10-05       628.10     FALSE
## 421568    45   98 2012-10-12      1061.02     FALSE
## 421569    45   98 2012-10-19       760.01     FALSE
## 421570    45   98 2012-10-26      1076.80     FALSE
```

```r
# Checking if there are any NA data
sum(is.na(train_set))
```

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

```r
# -- Test set ---
# -> same structure of training set
# Checking if there are any NA data
sum(is.na(test_set))
```

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

Some key insights could be drawn already from the results of just these few lines of code:

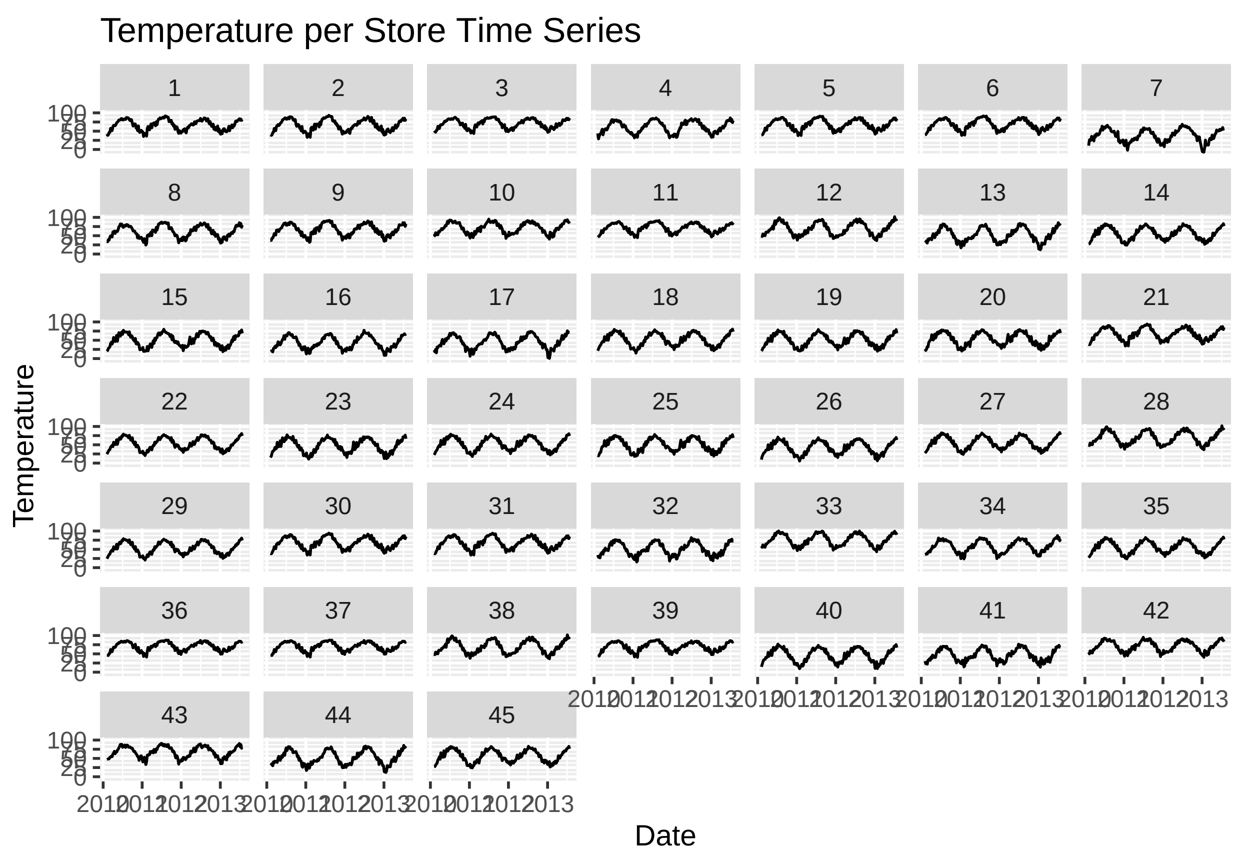
- Same features were present in multiple datasets: The features *Store*, for instance, could be found in both *stores* and *train_set* datasets. *Date*, instead, could be found in both *train_set* and *features*. Knowing this information was key because it meant that all the datasets could be merged into one by using the join function and the above features as primary keys.
- Some variables were not stored in the most appropriate format. The feature *Store*, for example, was stored as *numeric* and the feature *Type* as a *Character*, while both should have been stored as *factors* - as they were categorical variables that could assume just a limited number of values.
- Most of the stores were mainly either A or B type.
- The dataset *features* included some NAs values in multiple columns, specifically *Markdown 1-5*, *CPI*, *Unemployment*. MarkDown data were only available after Nov 2011 (and were not available for all stores all the time), while CPI and Unemployment data were not available in the weeks from the 03rd May 2013 until the 26th July 2013. Handling missing data is one of the most common problems in data science and can be approached in different ways. More detail about how NAs were managed will be provided in the section about data preprocessing.

The code above helped identify the available features that could be used in the model; thus these became the focus of the next phase of the data exploration analysis.

Temperature feature exploration

The first feature that was explored was **temperature**, which was available in the *features* dataset. The analysis started by having a look at how the temperature varied across time and regions (because each store was located in a different region). From the time series that was plotted, the cyclicity of the 4 seasons was quite noticeable. A significant variation of temperatures across regions was also quite evident and was confirmed by summarising the minimum and maximum temperature per store . This could indicate that the 45 stores could have been located in any area of the United States, possibly some in the South, other in the North and so forth.

```r
# Plotting the time series for temperature by stores.
features %>%
  ggplot(aes(x= Date, y=Temperature)) +
  geom_line() +
  facet_wrap(~Store) +
  ggtitle("Temperature per Store Time Series") +
  xlab("Date") +
  ylab("Temperature")
```



```r
# -> Average temperature by store
features %>%
  group_by(Store) %>%
  summarise(avg_temperature = mean(Temperature))
```

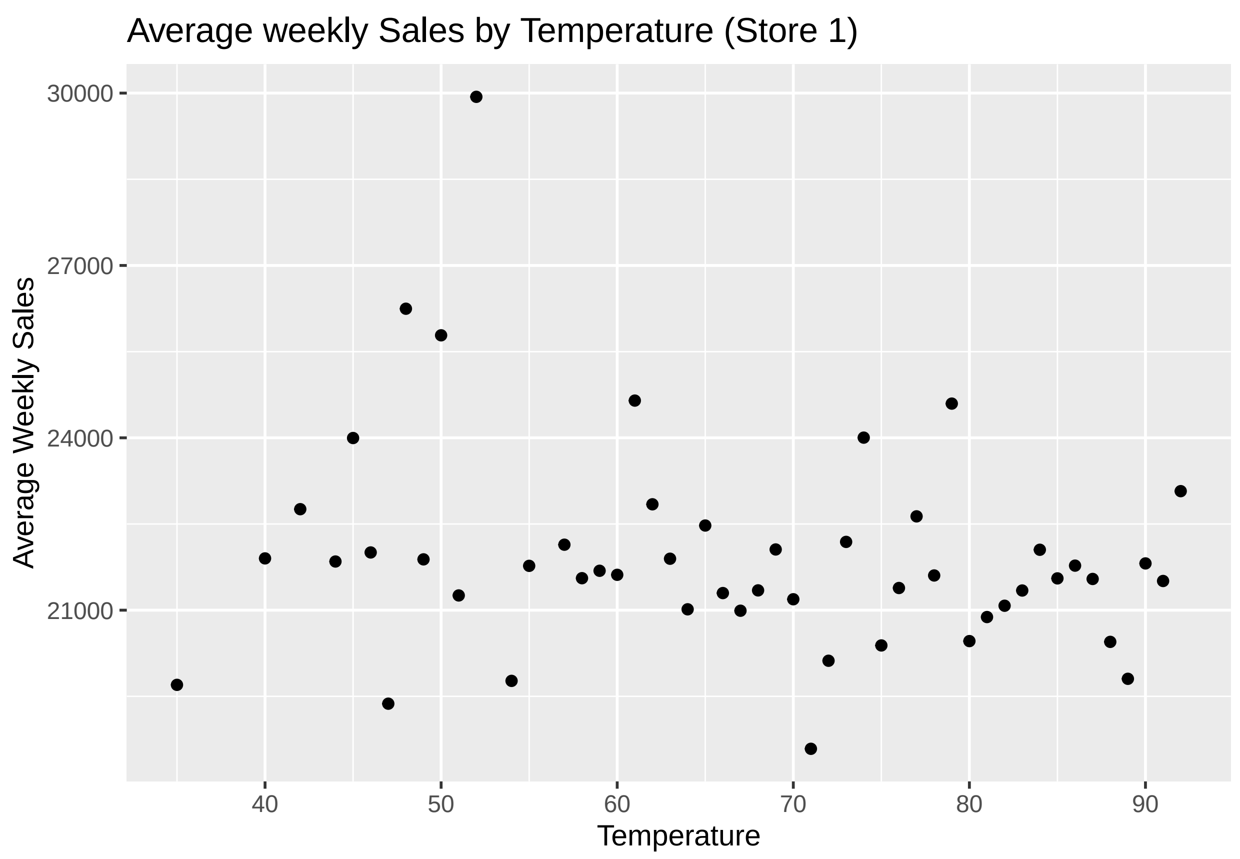
```
## # A tibble: 45 x 2
##    Store avg_temperature
##    <dbl>           <dbl>
##  1     1            66.9
##  2     2            66.7
##  3     3            70.4
##  4     4            61.4
##  5     5            68.2
##  6     6            68.5
##  7     7            37.9
##  8     8            61.2
##  9     9            66.3
## 10    10            71.3
## # ... with 35 more rows
```

```r
# Min & Max avg temperature by store
features %>%
  group_by(Store) %>%
  summarise(avg_temperature = mean(Temperature)) %>%
  arrange(avg_temperature)  %>%
  slice(c(1,45))
```

```
## # A tibble: 2 x 2
##   Store avg_temperature
##   <dbl>           <dbl>
## 1     7            37.9
## 2    33            75.4
```

Next, the possibility that a variation of *temperature* could significantly influence the sales was assessed through the use of a scatter plot. To isolate the effects of the feature, only one store was considered in this instance. Unfortunately, the result was inconclusive, as the chart did not show any clear pattern in the data. However, this did not necessarily mean that the *temperature* did not have any predictive power, but rather that this graphic representation was not suitable in this situation. By taking the average of sales across weeks that belonged to different seasons and promotional periods, other features and effects could come to play, thus masking the effect of changes in temperature.

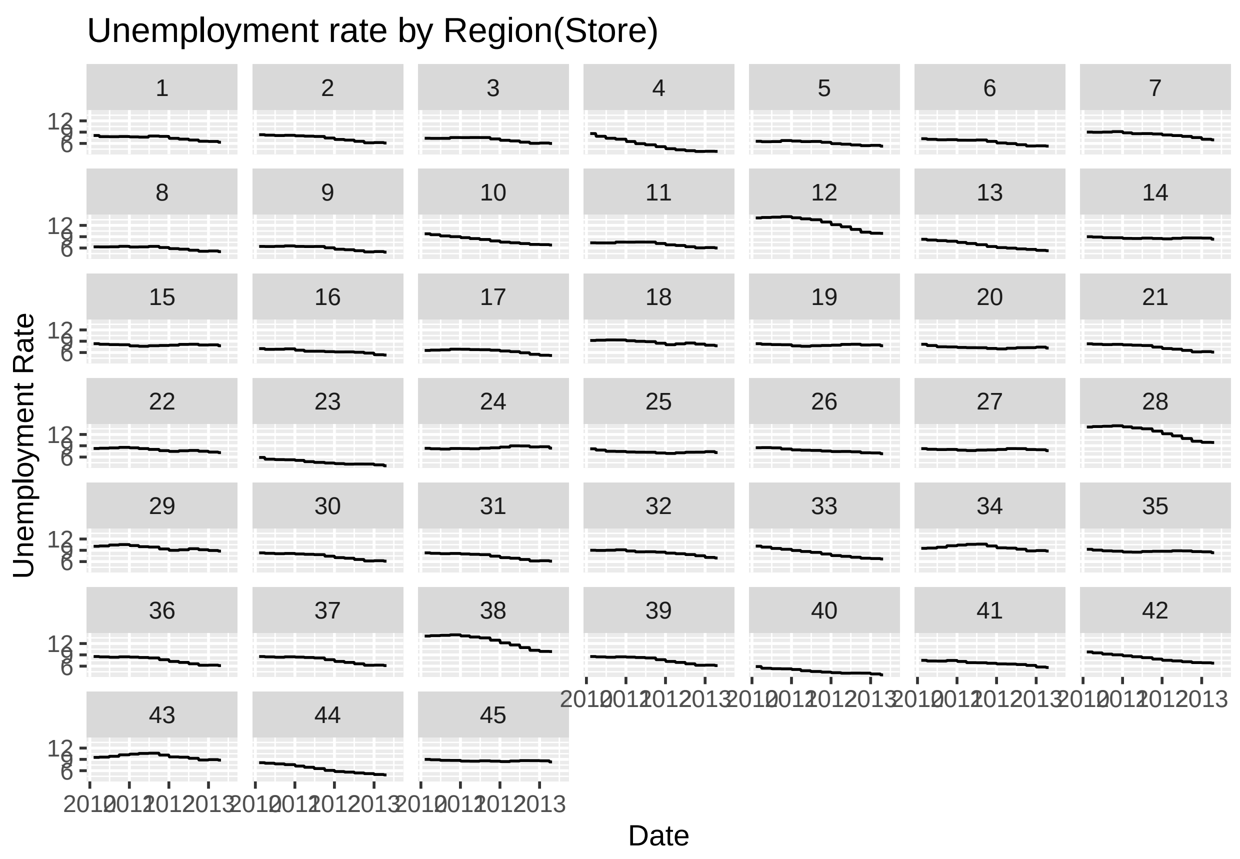
```r
# Determining how the Temperature influences the sales
train_set %>%
  left_join(stores, by ="Store") %>%
  left_join(features, by = c("Store","Date")) %>%
  filter(Store==1 & IsHoliday.x == FALSE) %>%
  mutate(round_temp = round(Temperature, digits=0)) %>%
  # rounding the temperature
  group_by(Store, round_temp) %>%
  summarise(avg_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=round_temp,y=avg_sales)) +
  geom_point() +
  ggtitle("Average weekly Sales by Temperature (Store 1)") +
  xlab("Temperature") +
  ylab("Average Weekly Sales")
```



Unemployment feature exploration

The second feature that was analysed was **Unemployment** rate. First, how the unemployment rate varied across stores was explored. As in the case of the temperature, unemployment rate varied considerably across areas/stores - most likely some stores were located in area wealthier than others.

```r
# Plotting the time series for unemployement rate by stores
features %>%
  ggplot(aes(x= Date, y= Unemployment)) +
  geom_line() +
  facet_wrap(~Store) +
  ggtitle("Unemployment rate by Region(Store)") +
  xlab("Date") +
  ylab("Unemployment Rate")
```



```r
# Average unemployement by store
features %>% filter(Unemployment>=0) %>%
  group_by(Store) %>%
  summarise(avg_unemployement = mean(Unemployment))
```

```
## # A tibble: 45 x 2
##    Store avg_unemployement
##    <dbl>             <dbl>
##  1     1              7.44
##  2     2              7.40
##  3     3              7.01
##  4     4              5.65
##  5     5              6.16
##  6     6              6.41
##  7     7              8.38
##  8     8              5.95
##  9     9              5.93
## 10    10              8.14
## # ... with 35 more rows
```

```r
# Min & Max avg unemployement by store
features %>%
  filter(Unemployment>=0)  %>%
  group_by(Store) %>%
  summarise(avg_unemployement = mean(Unemployment)) %>%
  arrange(avg_unemployement)  %>%
  slice(c(1,45))
```

```
## # A tibble: 2 x 2
##   Store avg_unemployement
##   <dbl>             <dbl>
## 1    23              4.67
## 2    38             12.6
```

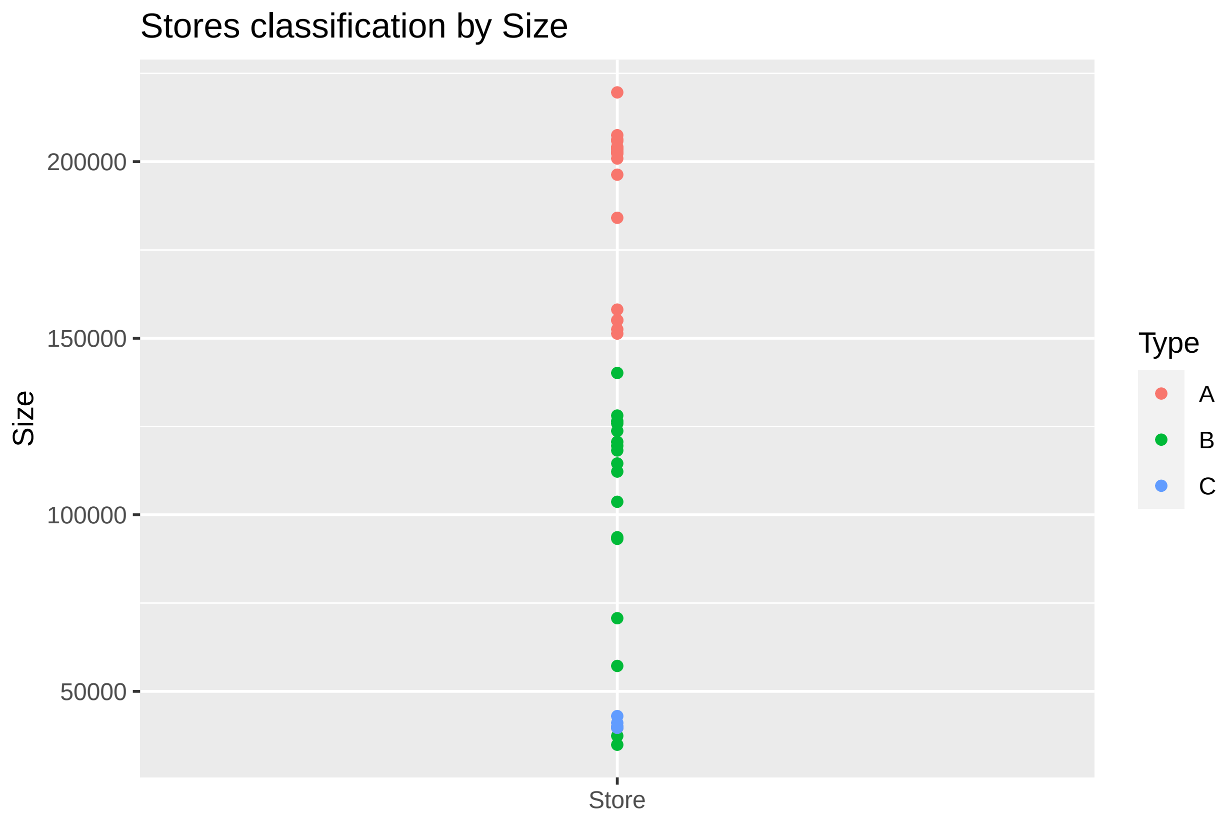
Store Type feature exploration

The next variable that was explored was **Store Type**. The first assumption that was verified was whether the classification of a store into a certain *type* depended on its *size*. As expected, the results confirmed that hypothesis - more specifically, the largest stores belonged to *Type A*, *Type B* were mostly medium stores, while *Type C* included the smallest stores. However, by plotting the *size* of the stores vs the designated *type*, it was also possible to classify the stores into 4 groups:

- *Small Stores* - size $< 7{,}500$
- *Medium Stores* - $7{,}500 <$ size $< 15{,}000$
- *Large Stores* - $15{,}000 <$ size $< 17{,}500$
- *XL Stores* - size $> 17{,}500$

For this reason, the feature *size_type* was then engineered and stored in the dataset to test in future whether using this classification instead of the feature *Type* would improve the accuracy of the model.

```r
# Plotting Store Type vs Size to understand if the type of stores depends on size and how
stores %>%
  ggplot(aes(x="Store", y=Size)) +
  geom_point(aes(color=Type)) +
  ggtitle("Stores classification by Size") +
  xlab("") +
  ylab("Size")
```



```r
# Classifying the stores based on the size
stores <- stores %>%
  mutate(size_type = ifelse(Size < 75000, "Small",
                            ifelse(Size < 150000, "Medium",
                                   ifelse(Size < 175000, "Large", "X-Large"))))
```

Next, the influence that the size of the store could have on sales was studied. By plotting the average weekly sales across the years by stores, it was noticed that sales in certain store where on average higher than in other stores and that sales appeared less seasonal from stores to stores.

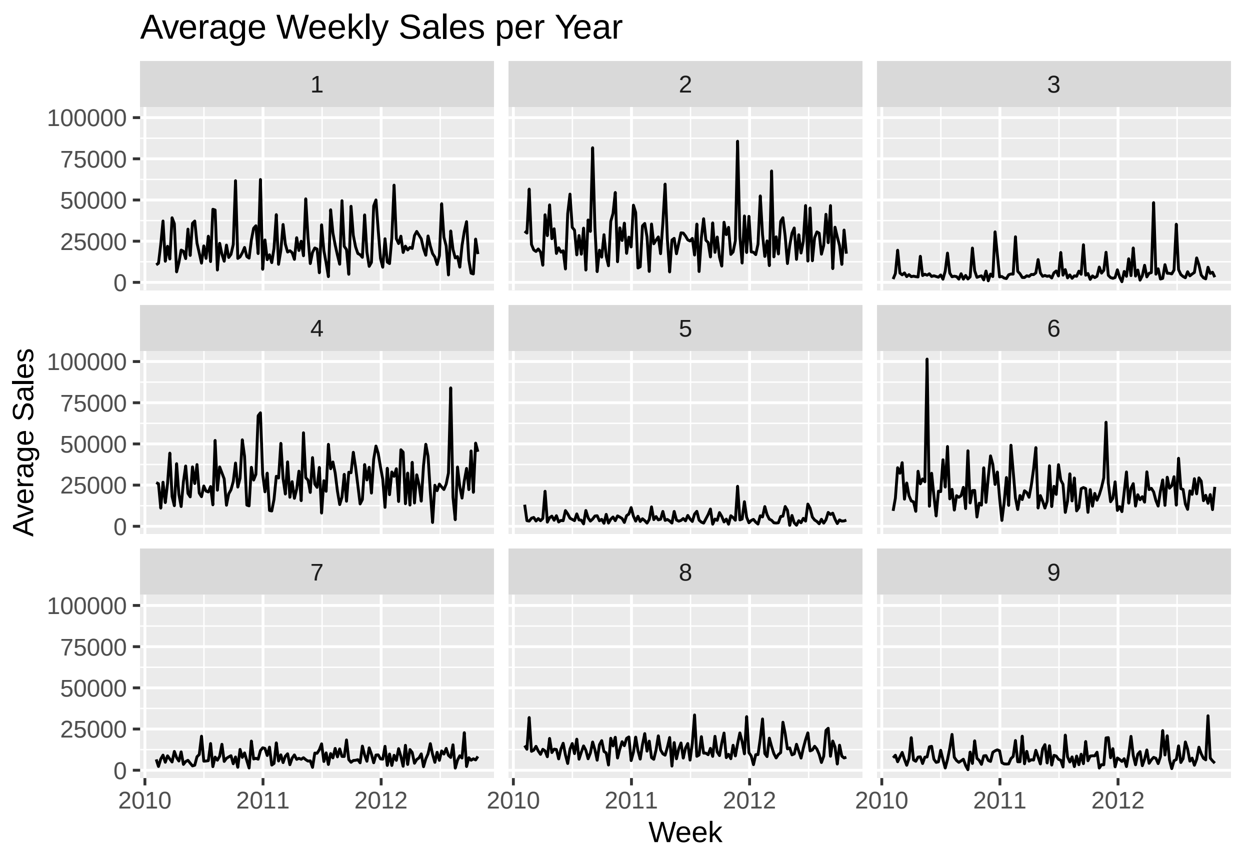
This led to explore two hypotheses:

- the magnitude of the weekly sales could depend on the *Type* of store;
- the seasonality of the weekly sales could depend on the *Type* of store

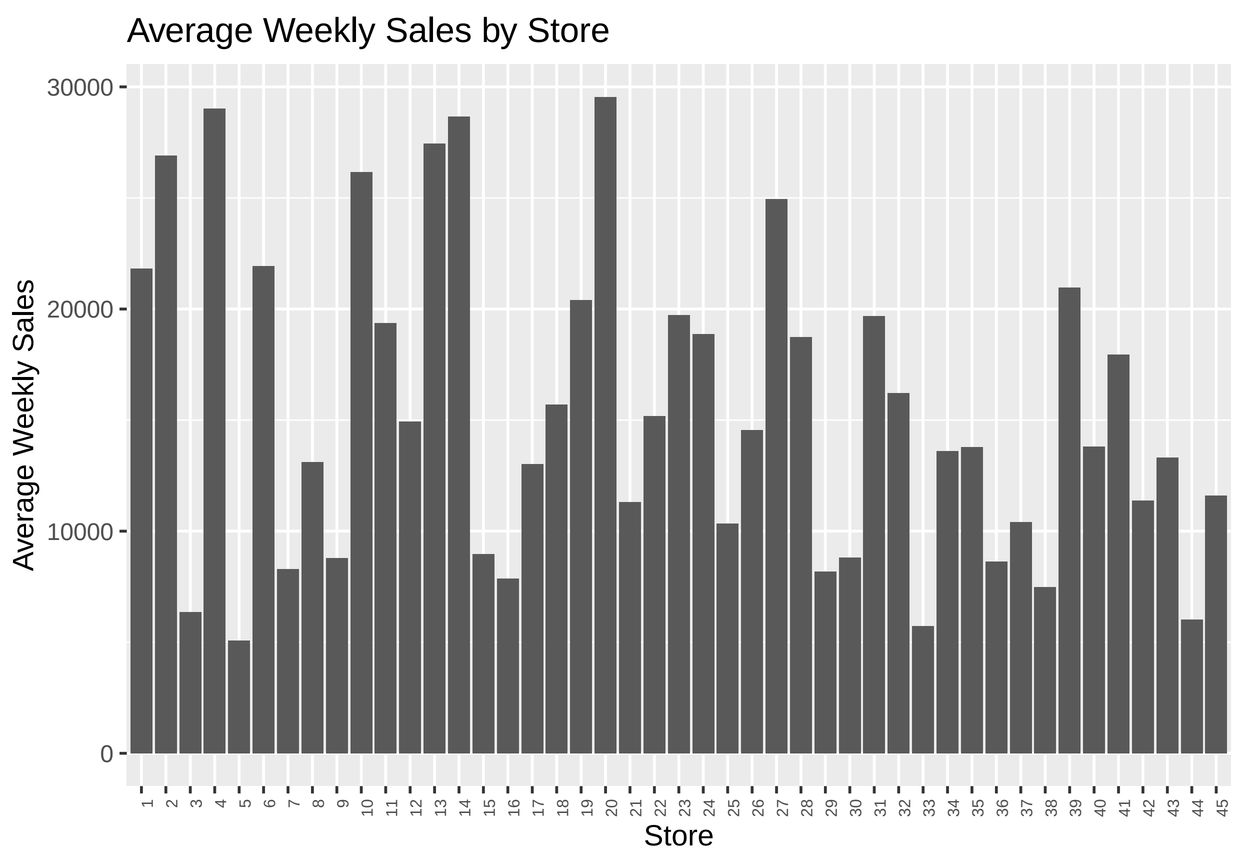
By plotting the time series of the average weekly sales by *Type* of store, it was possible to notice that:

- larger stores have on average more sales
- smallest stores (Type C) suffered less of seasonality fluctuations in sales

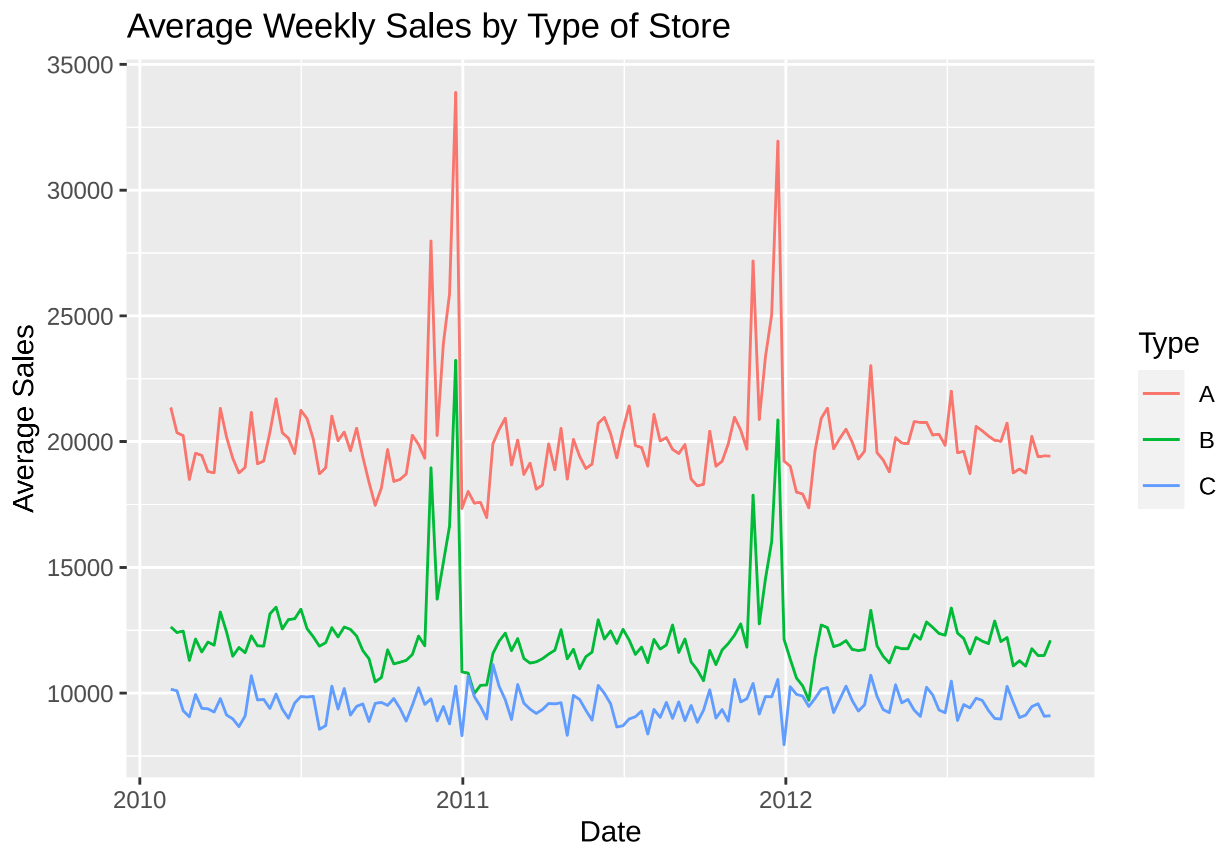
```r
# Average weekly sales  (Stores 1:9)
train_set %>%
  filter(Store==c(1,2,3,4,5,6,7,8,9)) %>%
  group_by(Store,Date) %>%
  mutate(avg_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=Date, y=avg_sales)) +
  geom_line() +
  facet_wrap(~Store) +
  ggtitle("Average Weekly Sales per Year") +
  xlab("Week") +
  ylab("Average Sales")
```



```
# Average weekly Sales per Store
train_set %>%
  left_join(stores, by = "Store") %>%
  group_by(Store) %>%
  summarise(average_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=as.factor(Store), y=average_sales)) +
  geom_bar(stat = "identity") +
  ggtitle("Average Weekly Sales by Store") +
  xlab("Store") +
  ylab("Average Weekly Sales") +
  theme(axis.text.x=element_text(angle=90,hjust=1, size = 6))
```

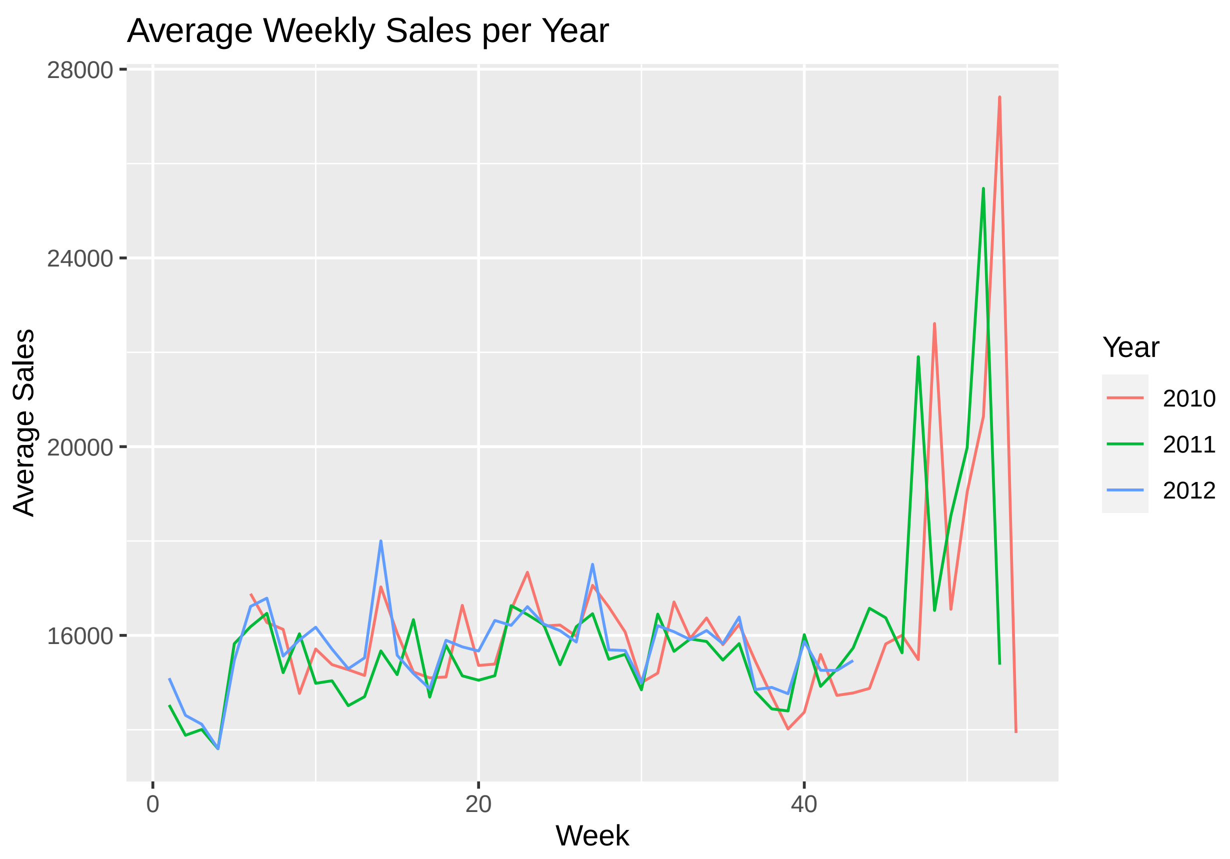


```
# Average sales by Type of store
train_set %>%
  left_join(stores, by ="Store") %>%
  group_by(Type,Date) %>%
  mutate(avg_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=Date, y=avg_sales)) +
  geom_line(aes(colour= Type)) +
  ggtitle("Average Weekly Sales by Type of Store") +
  xlab("Date") +
  ylab("Average Sales")
```



Finally, how the average weekly sales varied across the year was explored. Plotting the average weekly sales per year allowed to appreciate once again the presence of seasonalities - average weekly sales followed a similar pattern each year. In addition, the last weeks of the year were always the ones with significantly more sales than others. This did not come as a surprise because Christmas and New Year weeks always bring high volume of sales in retailing.

```
# Average weekly sales per year
train_set %>%
  mutate(Year = as.factor(year(Date)),
         Week=week(Date)) %>%
  group_by(Date) %>%
  mutate(avg_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=Week, y=avg_sales, color=Year)) +
  geom_line() +
  ggtitle("Average Weekly Sales per Year") +
  xlab("Week") +
  ylab("Average Sales")
```



```r
train_set %>%
  mutate(Year = as.factor(year(Date)), Week=week(Date))%>%
  filter(Week<=45) %>% # data for Xmas 2012 are missing
  group_by(Date) %>%
  mutate(avg_sales = mean(Weekly_Sales)) %>%
  group_by(Year)%>%
  summarise(average_weekly_sales =mean(avg_sales))
```

```
## # A tibble: 3 x 2
##   Year  average_weekly_sales
##   <fct>                <dbl>
## 1 2010                15725.
## 2 2011                15443.
## 3 2012                15704.
```

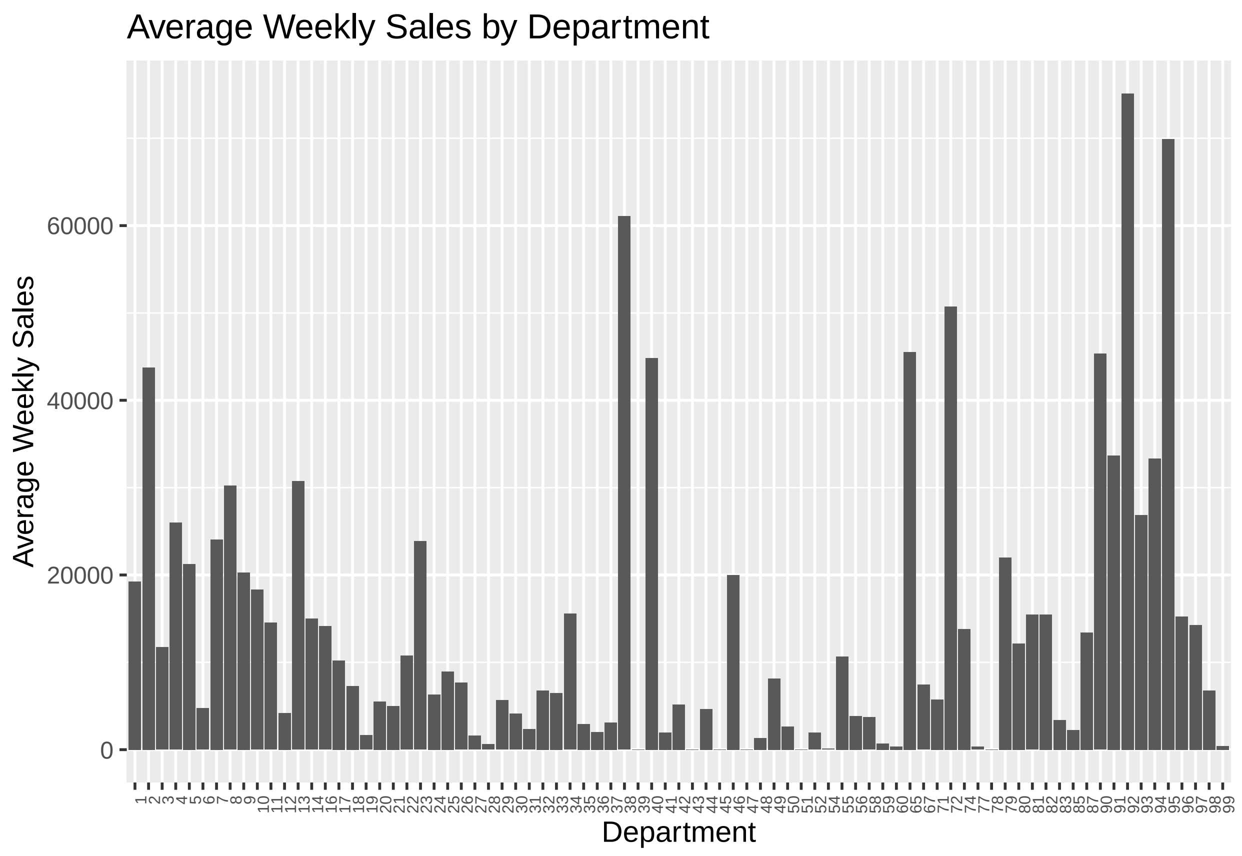
Department feature exploration

The next feature within the *train_set* that was explored was **Dept** (department). Similarly to the previous analyses, it was analysed how the weekly sales varied across departments. Subsequently, it was also explored whether the *Type* of store had an impact on the sales of each department. As expected, not all the departments were born equal - some departments saw, on average, larger sales than others. Breaking down the sales by Type of stores showed some interesting results:

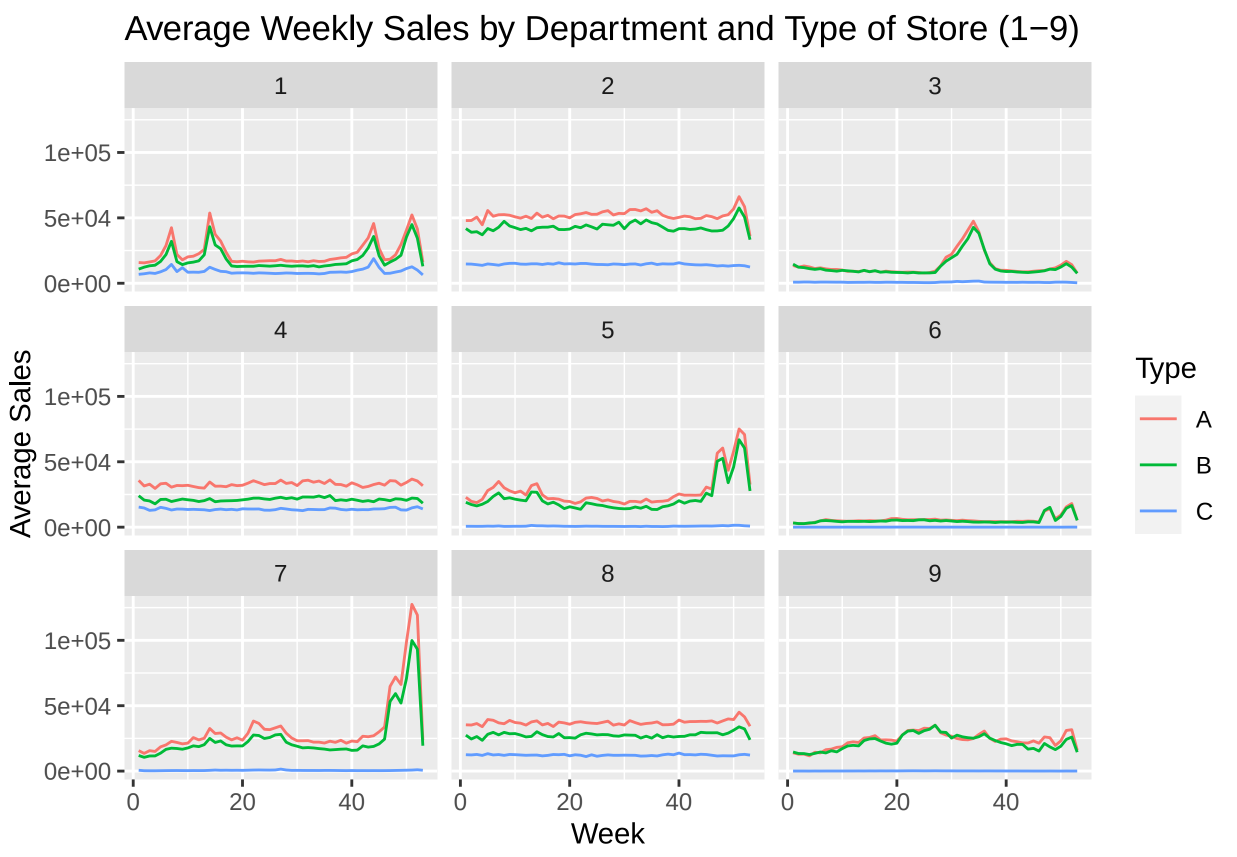
- generally, larger store saw larger sales across all the departments

- sales in certain departments were less influenced by the type of store than others
- sales in some departments were less seasonal than others

```
# Average weekly sales by Department
train_set %>%
  left_join(stores, by = "Store") %>%
  group_by(Dept) %>%
  summarise(average_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=as.factor(Dept), y=average_sales)) +
  geom_bar(stat = "identity") +
  ggtitle("Average Weekly Sales by Department") +
  xlab("Department") +
  ylab("Average Weekly Sales") +
  theme(axis.text.x=element_text(angle=90,hjust=1, size = 6))
```



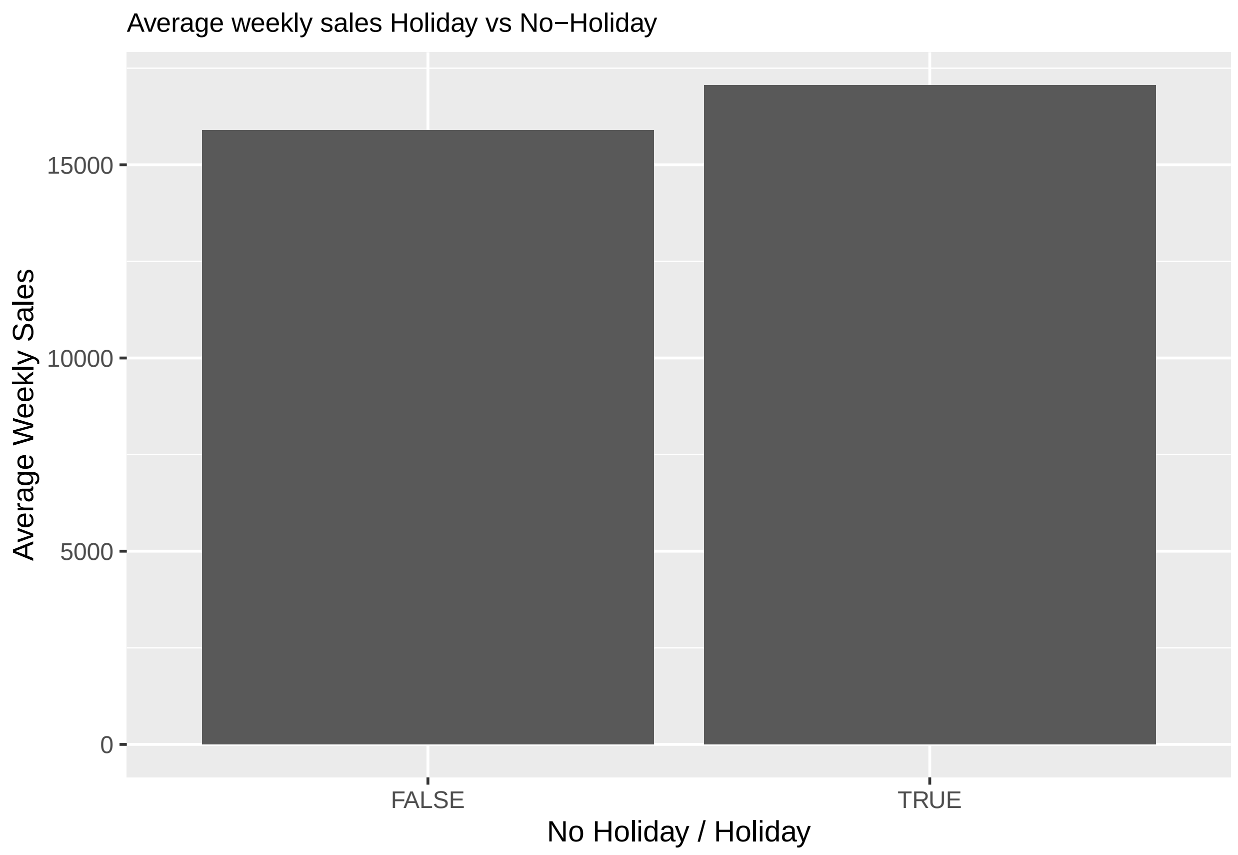
```
# Average weekly sales by Department and Type of store
train_set %>%
  mutate(Year = as.factor(year(Date)), Week=week(Date))%>%
  left_join(stores, by = "Store") %>%
  group_by(Dept, Type, Week) %>%
  summarise(average_sales = mean(Weekly_Sales)) %>%
  filter(Dept %in% seq(1,9)) %>%
  ggplot(aes(x=Week, y=average_sales)) +
  geom_line(aes(color =Type)) +
  facet_wrap(~Dept) +
  ggtitle("Average Weekly Sales by Department and Type of Store (1-9)") +
  xlab("Week") +
  ylab("Average Sales")
```



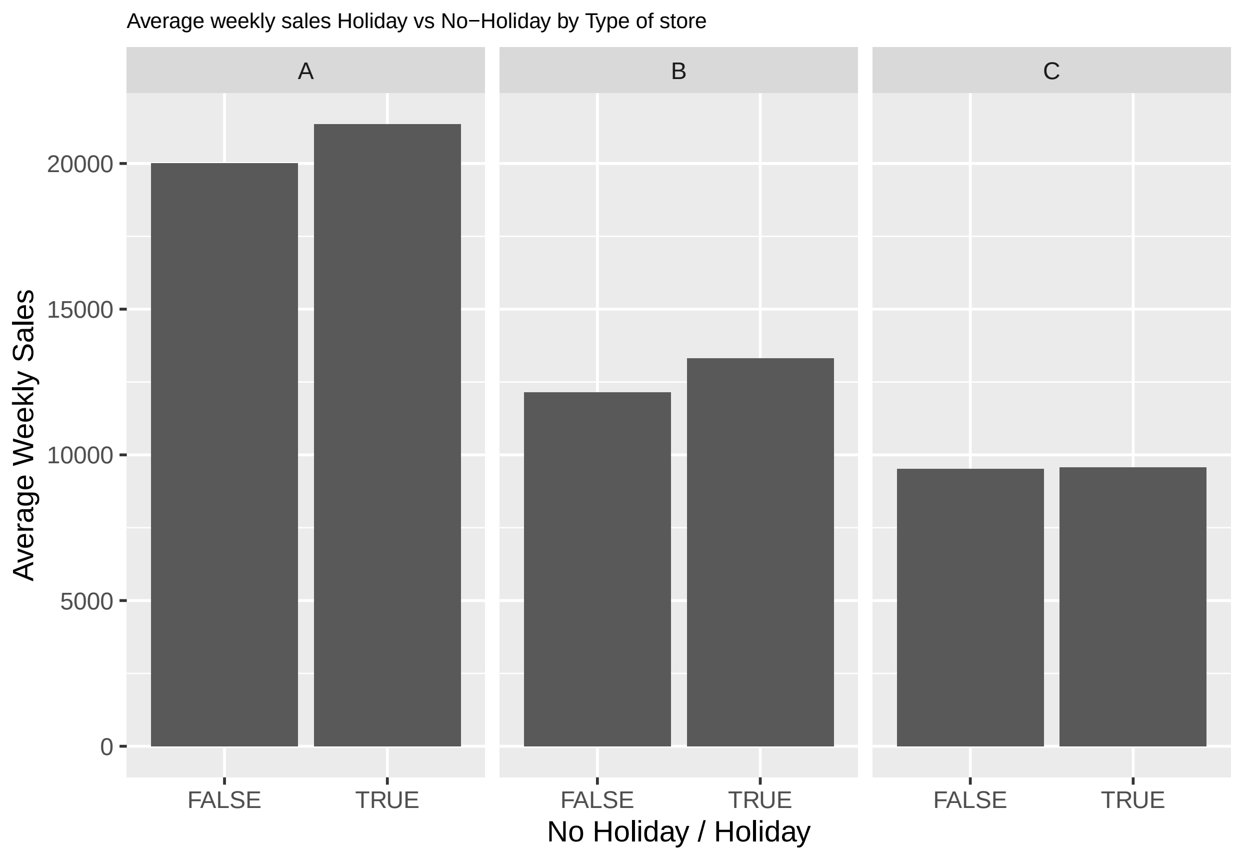
Holiday feature exploration

The feature **is.Holiday** was the last variable to be explored. The seasonal nature of the weekly sales had been highlighted in every analysis undertaken so far, thus it made sense to have a deep look into the impact of holidays on sales. Firstly, it was explored whether weekly sales were on average higher during holiday periods. After that this assumption was verified, it was assessed whether this condition was the same across store type. As expected following on the previous analyses, weekly sales in smaller stores (Type C) stores seemed less impacted by the holiday vs no-holiday periods factor.

```
# Impact of holiday on sales in general
train_set %>%
  left_join(features, by= c("Store", "Date")) %>%
  group_by(IsHoliday.x) %>%
  summarise(avg_weekly_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=IsHoliday.x, y=avg_weekly_sales))+
  geom_bar(stat = "identity") +
  xlab("No Holiday / Holiday") +
  ylab("Average Weekly Sales") +
  ggtitle("Average weekly sales Holiday vs No-Holiday") +
  theme(plot.title = element_text(size=10))
```



```r
# -> The weekly sales are on average higher during holiday periods

# Verifing if this pattern is the same across store 'type'
    train_set %>%
    left_join(features, by= c("Store", "Date")) %>%
    left_join(stores, by=c("Store")) %>%
    group_by(IsHoliday.x, Type ) %>%
    summarise(avg_weekly_sales = mean(Weekly_Sales)) %>%
    ggplot(aes(x=IsHoliday.x, y=avg_weekly_sales)) +
    geom_bar(stat = "identity") +
    facet_wrap(~Type) +
    ylab("Average Weekly Sales") +
    xlab("No Holiday / Holiday")+
    ggtitle("Average weekly sales Holiday vs No-Holiday by Type of store") +
    theme(plot.title = element_text(size=8))
```



In general, one of the reasons as to why holiday periods drive sales is because retailers tend to run promotional activities around these dates to capitalise on the festive activities undertaken by the shopper. Specifically, in the case of Walmart,several promotional events, known as *markdown*, are run throughout the year. These markdowns usually precede prominent holidays, the four largest of which are the Super Bowl, Labor Day, Thanksgiving, and Christmas.

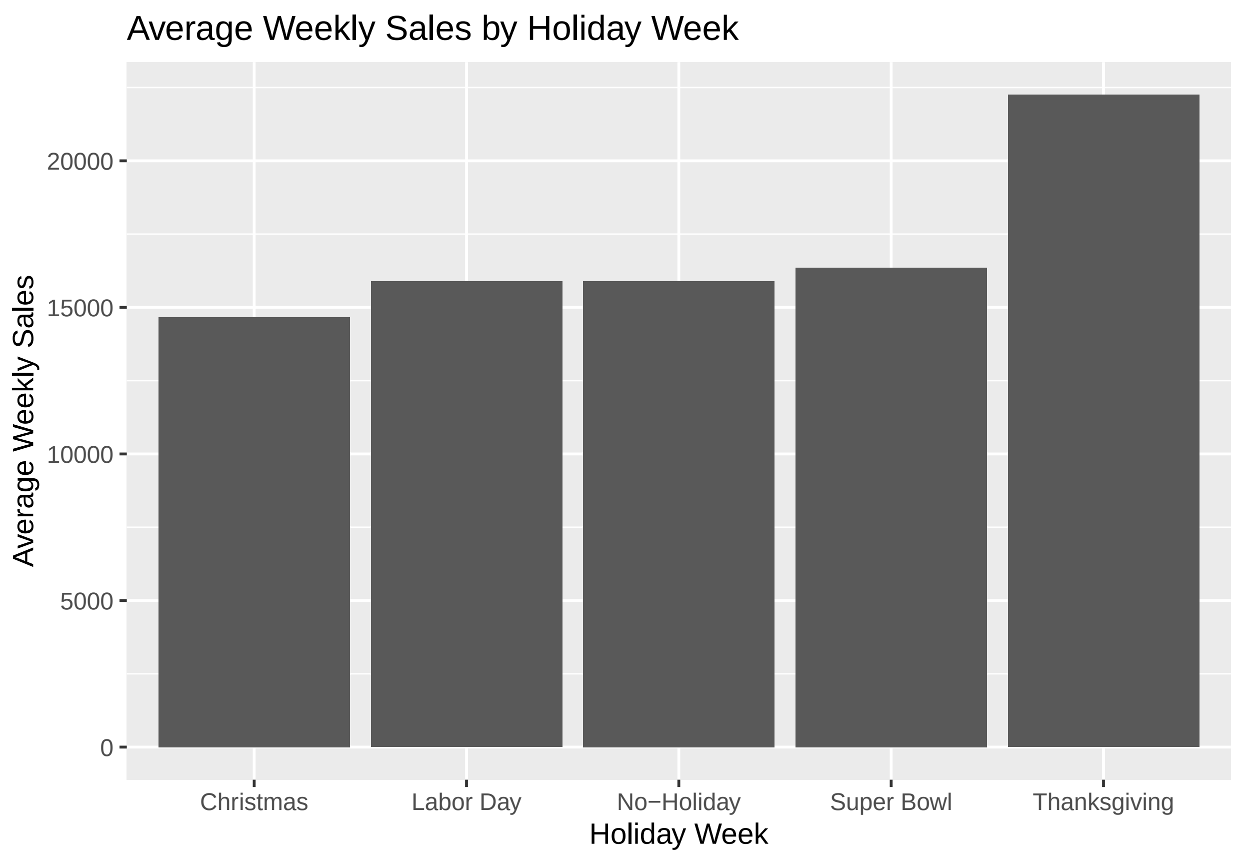
In the specific years included in the datasets, these four holidays fall within the following weeks (although not all holidays were in the data):

- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 08-Feb-13
- Labor Day: 10-Sep-10, 09-Sep-11, 07-Sep-12, 06-Sep-13
- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

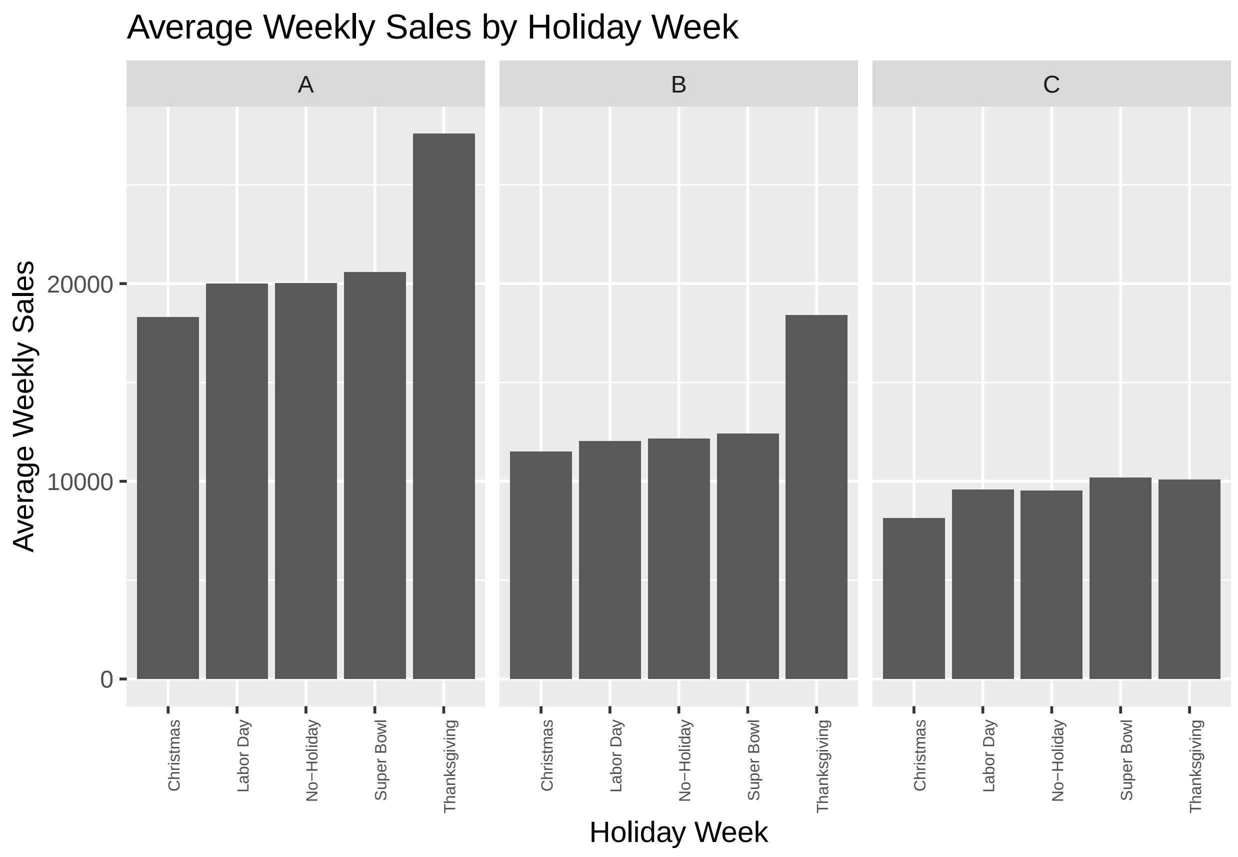
A new feature called **holiday_week** was created - a categorical variable that indicated the specific holiday, from the ones listed above, that a week was belonging to. The value of this variable in any non-festive week was *"No-Holiday"*. Having created this variable, it was possible to analyse the average sales on those weeks. What came out from this analysis was:

- The week leading to *Thanksgiving* drove on average significantly more sales than any other *festive* week
- Sales in the Christmas week were actually even lower than the average sales in *Non-holiday* weeks. This was not totally unexpected, because shoppers tend to buy (and stock-buy) in the first weeks of December while spending Christmas Eve and Christmas day at home with their families
- As expected sales in smaller stores (type C) in holiday periods followed a different pattern than the one present in larger stores
- Thanksgiving holiday seemed to have a larger impact on type A and B stores

```
# Creating the vectors with the dates for each holiday
superbowl    <- c("2010-02-12", "2011-02-11", "2012-02-10", "2013-02-08")
laborday     <- c("2010-09-10", "2011-09-09","2012-09-07", "2013-09-06")
thanksgiving <- c("2010-11-26", "2011-11-25", "2012-11-23", "2013-11-29")
christmas    <- c("2010-12-31", "2011-12-30", "2012-12-28", "2013-12-27")

# Inserting manually a flag that identify the specific holiday
features <- features %>%
  mutate(Date= as.Date(Date, format="%B %d %Y"),
         holiday_week = ifelse(Date %in% as.Date(superbowl), "Super Bowl",
                          ifelse(Date %in% as.Date(laborday), "Labor Day",
                            ifelse(Date %in% as.Date(thanksgiving),"Thanksgiving",
                              ifelse(Date %in% as.Date(christmas),"Christmas","No-Holiday")))))

# Understanding impact of specific holidays on Sales
train_set %>%
  left_join(features, by= c("Store", "Date")) %>%
  group_by(holiday_week) %>%
  summarise(average_week_sales = mean(Weekly_Sales)) %>%
  arrange(average_week_sales) %>%
  ggplot(aes(x=holiday_week, y=average_week_sales))+
  geom_bar(stat = "identity") +
  ggtitle("Average Weekly Sales by Holiday Week") +
  xlab("Holiday Week") +
  ylab("Average Weekly Sales")
```



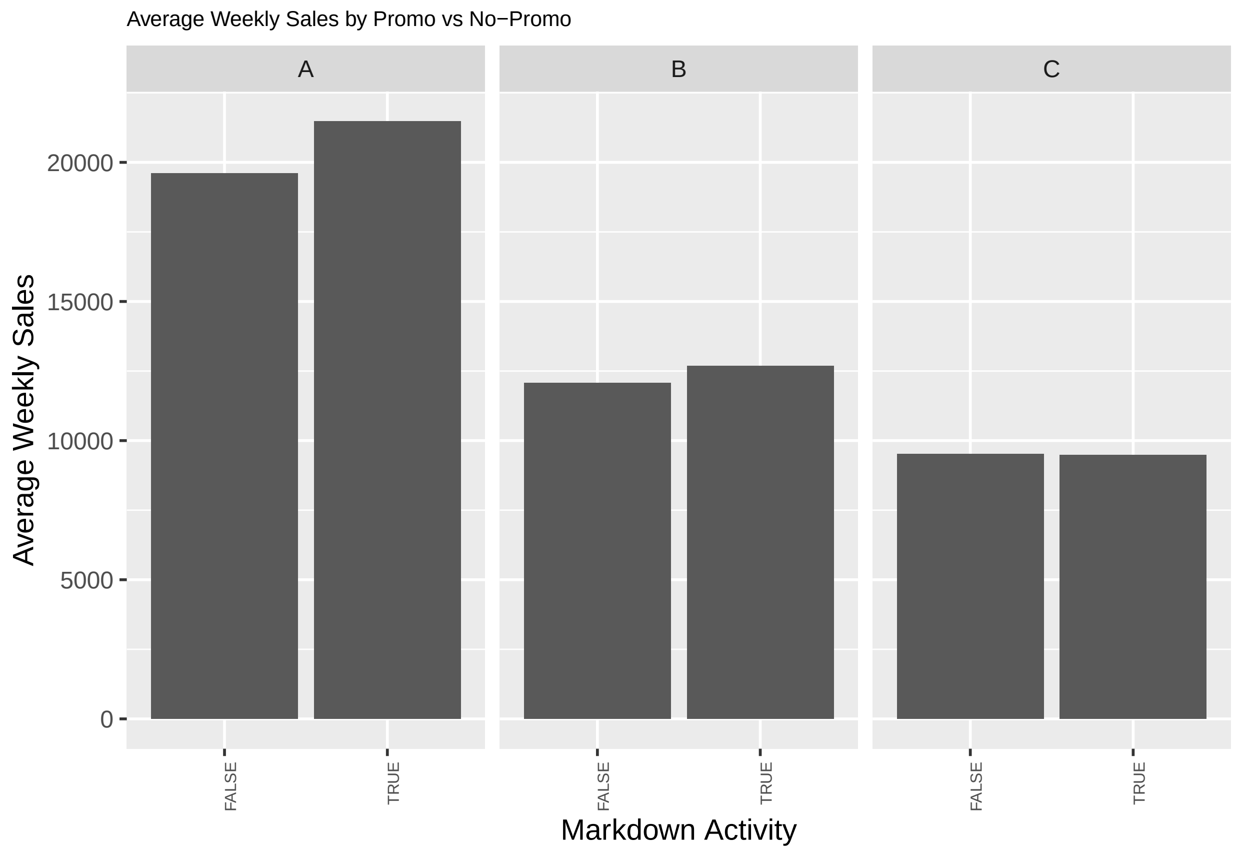
```r
# Understanding if this pattern is present across the store types
train_set %>%
  left_join(features, by= c("Store", "Date")) %>%
  left_join(stores, by=c("Store")) %>%
  group_by(holiday_week, Type) %>%
  summarise(average_week_sales = mean(Weekly_Sales)) %>%
  ggplot(aes(x=holiday_week, y=average_week_sales))+
  geom_bar(stat = "identity") +
  facet_wrap(~Type)+
  ggtitle("Average Weekly Sales by Holiday Week") +
  xlab("Holiday Week") +
  ylab("Average Weekly Sales") +
  theme(axis.text.x=element_text(angle=90,hjust=1, size = 6))
```



Finally, the impact of markdown activities on sales was examined. In line with expectations, the weeks that included any sort of *MarkDown* promotional activities had more sales; furthermore, *Markdown* activities seemed to be driving more sales in larger stores than in smaller ones.

```r
# Exploring the markdown feature

# Average sales when markdown promo events are running (by type of store)
train_set %>%
  left_join(features, by= c("Store", "Date")) %>%
  left_join(stores, by=c("Store")) %>%

# Creating a flag to identify weeks with MarkDown activites
```

```r
mutate(tot_markdown = MarkDown1 + MarkDown2 + MarkDown3 + MarkDown4 + MarkDown5,
       markdown_flag = ifelse(is.na(tot_markdown)== TRUE,"FALSE","TRUE")) %>%
group_by(Type, markdown_flag) %>%
summarise(avg_weekly_sales= mean(Weekly_Sales)) %>%
ggplot(aes(x=markdown_flag,
           y=avg_weekly_sales)) +
geom_bar(stat = "identity") +
facet_wrap(~Type)+
ggtitle("Average Weekly Sales by Promo vs No-Promo") +
xlab("Markdown Activity") +
ylab("Average Weekly Sales") +
theme(axis.text.x=element_text(angle=90,hjust=1, size = 6),
      plot.title = element_text(size=8))
```



Unfortunately, it was not possible to continue with any further data exploration that regarded Markdown activities, due to the anonymous nature of those data - it was not specified what these numbers actually represented (i.e. quantities, sales, etc.); in addition, these data were grouped by store, therefore it was not possible to reallocate the values by department.

Key insights

The key insights that were gained from the data exploration analysis could be summarised as follow:

- store *type* seemed to be the feature that affected the sales the most. This variable depended on the size of the store. Larger stores tended to have larger sales but these are usually also more affected by seasonalities and promotional activities

- Certain features such as temperature and unemployment rate varied considerably across the stores, however their impact on sales was not possible to be identified and quantified through the data visualisation techniques that were used.
- The features *Markdown* could not be directly used within the models (due anonymised to the anonymised nature of the data and consequent lack of key information)
- data needed substantial preprocessing before implementing the machine learning models in order to:
- replace missing values (*CPI*, *Unemployment*)
- engineer new features (*holiday_week*, *size_type*, *Week*, *Month* and *Year*)
- transform the class of certain variables (*Store*, *Type*, *Dept*, *size_type* and *holiday_week*)

Data pre-processing

During the data exploration analysis it was noticed that the data might have required some pre-processing before being fed into the machine learning models.

Dealing with NA values

Handling missing data is one of the most common problem in data science. Unfortunately, there is no perfect way to solve this issue - there are different solutions depending on the kind and nature of the problem. Whilst sometimes it is safe to remove the data with missing values (entire rows), when this is limited to a small number of random observations, in most cases using a deletion method would be disadvantageous because it might lead to drop features that are perfectly filled and informative.

As seen in the exploratory data analysis, the dataset *features* contained 3 variables with missing data:

- Markdown 1-5
- CPI
- Unemployment

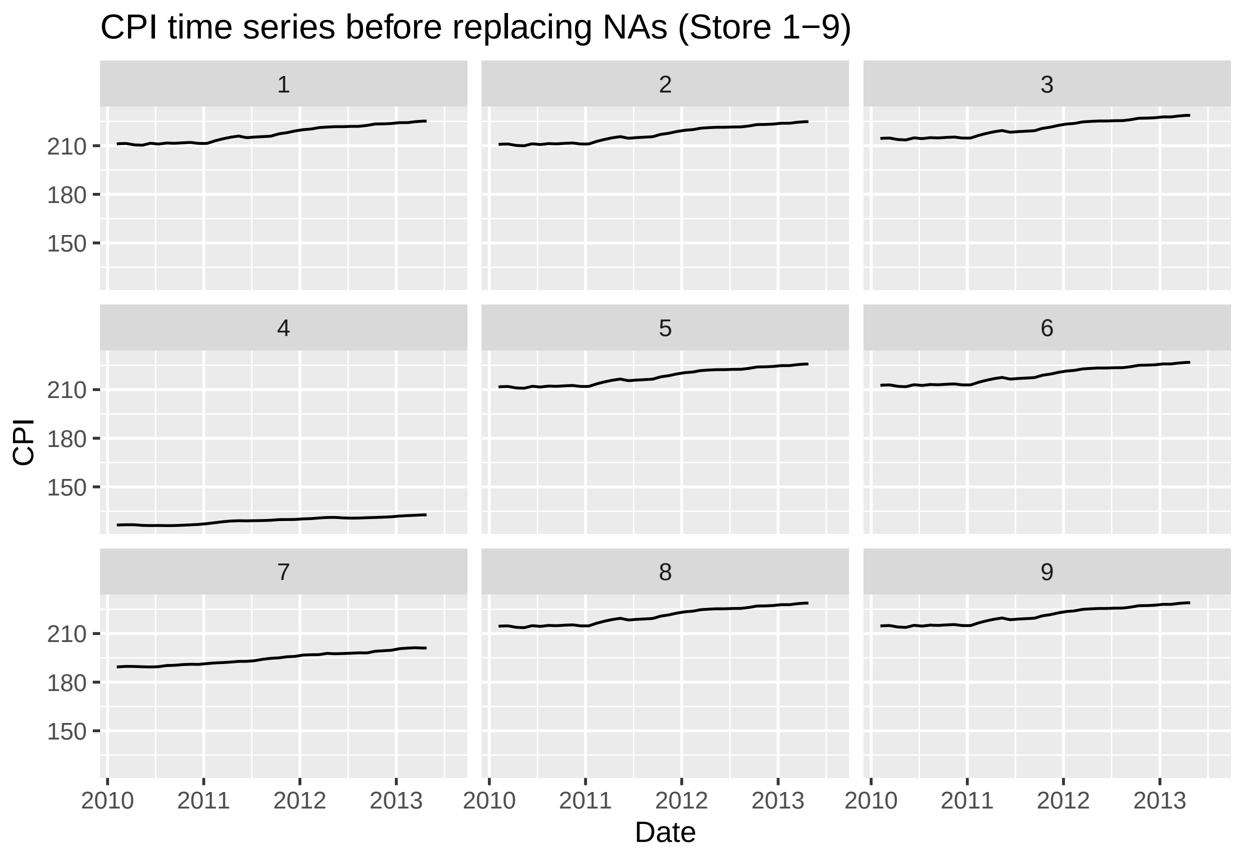
To identify the best approach to the problem, it was first identified whether a pattern was present within the missing data. MarkDown data were only available after Nov 2011 and were not available for all stores all the time, however due to the anonymous nature of the data it had been decided to discard these features and use the new engineered feature *holiday_week* instead.

Both CPI and Unemployment data were not available in a specific time interval - the weeks from the 03rd May 2013 until the 26th July 2013 - basically the last 13 weeks of the whole time period considered. This was a key insight because it had significant impact on the methodology chosen for the data imputation. First of all, it was decided not to delete the rows with missing data mainly for two reasons:

- from the exploratory data analysis, CPI and Unemployment did not appear to be key features
- as the missing data belonged to a specific time interval, due to the time dependent nature of the output variable (Weekly Sales was basically a time-series) deleting all the data from such time interval could have led to miss important information about trend, seasonalities, cycles, etc.

By plotting the time-series related to variables with missing data (ie CPI and Unemployment), it was possible to identify the most appropriate imputational method for each feature. As the time-series for CPI showed somewhat a trend and no seasonalities, a regression model was used for a linear interpolation of the missing data.

```r
# Matrix with the indexes of rows with NAs
na_index <- is.na(features$CPI)

# Inizialising dataset that will contain the data from features after the pre-processing
features_post_pp <- features[,]

# Plotting CPI (Store 1)
features %>%
  filter(Store %in% c(1:9)) %>%
  group_by(Store) %>%
  ggplot(aes(x=Date, y=CPI)) +
  geom_line() +
  facet_wrap(~Store) +
  ggtitle("CPI time series before replacing NAs (Store 1-9)")
```



The code used to predict the values that were used to replace the NAs is shown below. First, a *for* loop was implemented to:

1. fit a linear regression model for CPI for each of the 45 stores/regions
2. predict the missing CPI values using the fitted regression model
3. store those predicted values in a dataframe called *predicted_cpi*.

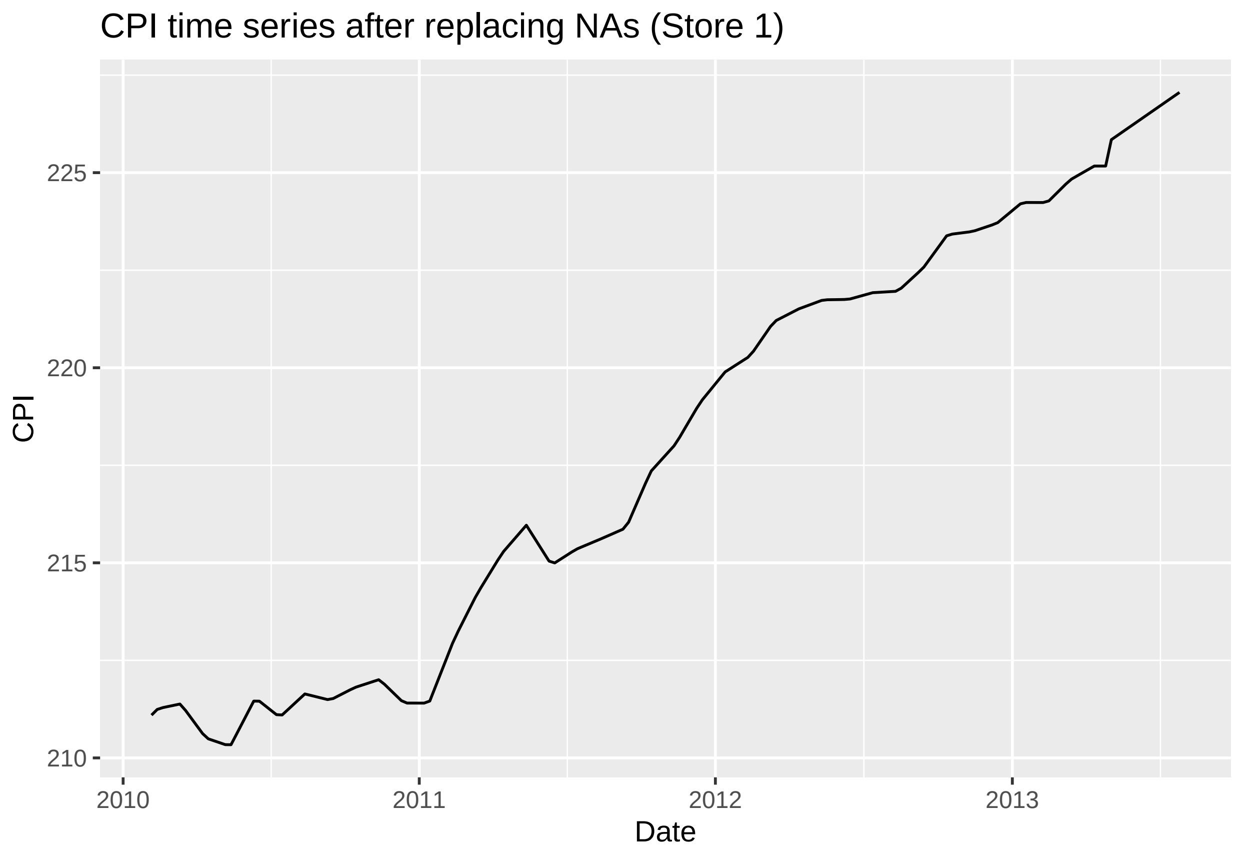
The resulting *predicted_cpi* dataframe was therefore a 13x45 dataframe, where each column represented one of the 45 stores, while the rows were representing each of the 13 weeks where the data were missing.

A second *for* loop was then used to replace the NAs values. As mentioned, the exploratory data analysis showed a pattern in the missing CPI data: each store was missing the values for CPI for the exact same weeks, which were the last 13 weeks of the time period. This insight was pivotal when building the second *for* loop, because it allowed to know exactly the rows with the missing data. Basically, the missing data were periodical - there were 13 rows with NAs for every 169 (*rows with filled CPI data for each store*) + 13 (*rows with NA CPI data for each store*) rows of the *features* dataset.

```
# Initialising dataframe that will contain predictions/ NA replacements
predicted_cpi <- data.frame(matrix(ncol = 45, nrow = 13))

# Loop to estimate values that will replace NAs with predictions from linear regression
i=1
for(i in 1:45) {
  fit_cpi <- features[-na_index,c(1,2,10)] %>%
  filter(Store==i) %>%
  lm(CPI ~  Date, data=.) # excluding NAs

  # populating the dataframe that will contains the predictions
  predicted_cpi[,i] <- features[na_index,c(1,2,10)] %>%
    filter(Store==i) %>%
    predict(fit_cpi, newdata=.)
  i=i+1
}

# Loop to replace NAs
i=1
start =170
for(i in 1:45) {
  loop_index <- seq(start,start+12)
  features_post_pp$CPI[loop_index] <- predicted_cpi[,i]
  start = start + 169 + 13
  i=i+1
}

# Checking if CPI NAs have been replaced
sum(is.na(features_post_pp[,10]))
```

Once the NA values were replaced, to check whether chosen imputation method was appropriate and correctly implemented, the values of the CPI after the data-preprocessing were plotted.

```
# Plotting CPI (Store 1)
features_post_pp %>%
  filter(Store==1) %>%
  ggplot(aes(x=Date, y=CPI)) +
  geom_line() +
  ggtitle("CPI time series after replacing NAs (Store 1)")
```



A similar procedure was followed when replacing the NA values for the feature *Unemployment*. However, from the plot it was clear that using a linear regression method would have not been appropriate in this case. The unemployment rate seemed to follow a floor/ceiling type of function, therefore a *Last Observation Carried Forward* (LOCF) method appeared to be more appropriate. Basically, after looking at the charts, it seemed safe to assume that the unemployment rate stayed constant throughout the time interval with NA values. Therefore, the last observed value of the CPI for each specific store was used to replace the missing values. The *for* loops coded in this stage followed a similar logic explained in the section dedicated to the CPI case. The only difference was the coding of the LOFC method instead of a linear regression.

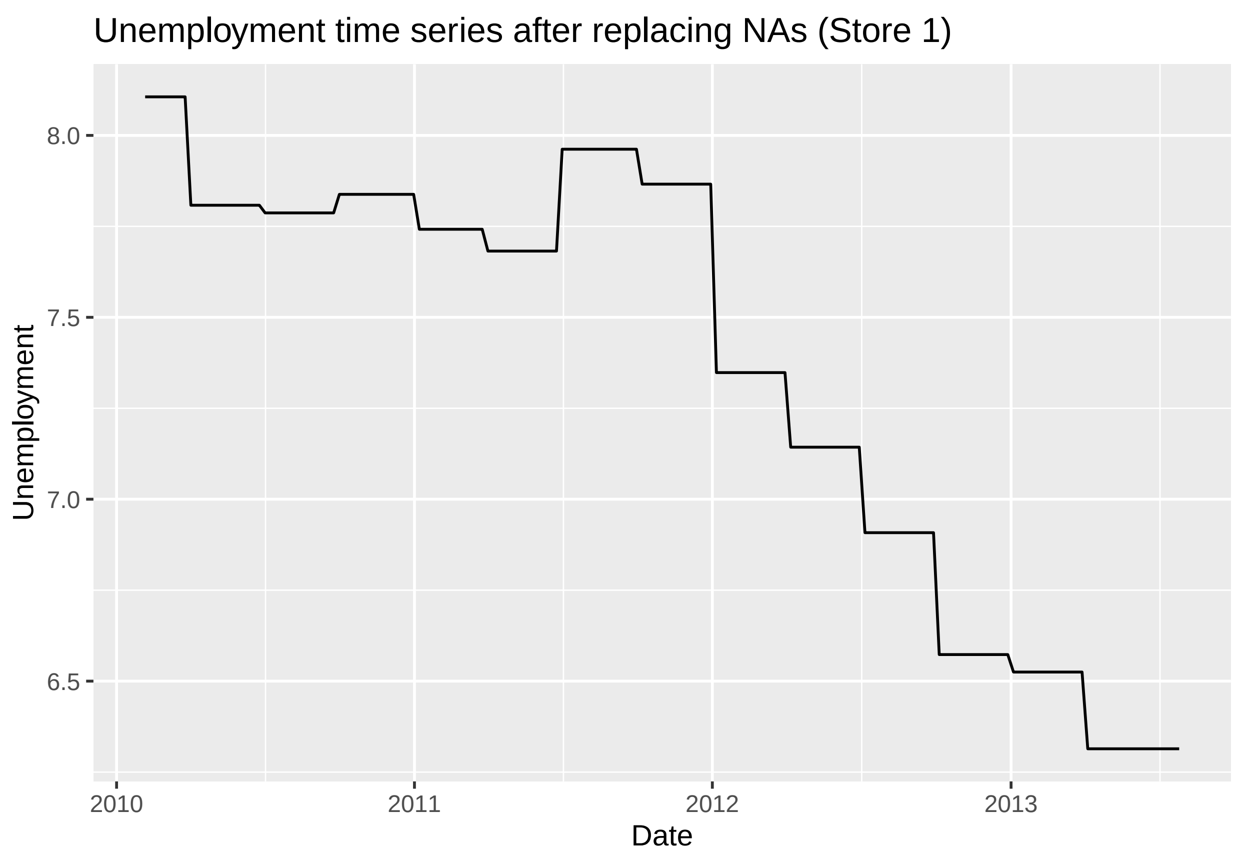
```
# Initialising dataframe that will contain predictions/ NA replacements
predicted_unmp <- data.frame(matrix(ncol = 45, nrow = 13))

# Loop to estimate values that will replace NAs
i=1
for(i in 1:45) {

  # Storing last recorded value of Unemployment for store i
  last_unm <- features %>%
    filter(Store==i,
           is.na(Unemployment)==FALSE) %>%
    select(Unemployment) %>%
    tail(1) %>%
    pull

  # Populate the dataframe with the replacement for the NAs
  predicted_unmp[,i] <- rep(last_unm,13)
```

```r
  i=i+1
}

# Loop to replace NAs
i=1
start =170
for(i in 1:45) {
  loop_index <- seq(start,start+12)
  features_post_pp$Unemployment[loop_index] <- predicted_unmp[,i]
  start = start + 12 + 170
  i=i+1
}

# Checking if Unemployment NAs have been replaced
sum(is.na(features_post_pp[,11]))
```

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

The plotted Unemployment time-series after that replacement of the missing value, gave solid indications that the chosen method was appropriate and correctly implemented.

```r
# Plotting Unemployment (Store 1)
features_post_pp %>%
  filter(Store==1) %>%
  ggplot(aes(x=Date, y=Unemployment)) +
  geom_line() +
  ggtitle("Unemployment time series after replacing NAs (Store 1)")
```



```
# -> The results from the replacements of the NA values appears to be satisfactory

# Removing temporary files
rm(i, last_unm, loop_index, start, predicted_cpi, predicted_unmp, fit_cpi )
```

Joining the datasets

The information included within the *stores* and *features* dataframes were included into the *test* and *training* sets through the use of the *left_join* function.

```
# Adding the information from the datasets 'stores' and 'features' to the 'train' and 'test' sets
# Train set
train_set <- train_set %>%
  left_join(stores, by ="Store",
            suffix = c("", "_new")) %>%
  left_join(features, by = c("Store","Date"),
            suffix = c("", "_new"))


# Test set
test_set <- test_set %>%
  left_join(stores, by ="Store",
            suffix = c("", "_new")) %>%
  left_join(features, by = c("Store","Date"),
            suffix = c("", "_new"))
```

After that, the features that were engineered during the data exploration phase were added to the *test* and *training* sets. Whilst the feature **size_type** had been directly added onto those datasets while being explored, the variables corresponding to the **Week**, **Month**, and **Year** were added during this stage.

```r
# Adding the features engineered in the data exploration
# Week, Month and Year of the date
train_set <- train_set %>%
  mutate(week_date = week(Date),
         month_date = month(Date),
         year_date = year(Date)
         )

test_set <- test_set %>%
  mutate(week_date = week(Date),
         month_date = month(Date),
         year_date = year(Date)
  )
```

Dealing with categorical variables

The exploratory analysis also highlighted that some features were not stored in the most appropriate format. Therefore, during the data pre-processing the variables **Store**, **Type**, **Dept**, **size_type** and **holiday_week** were transformed into factors. This operation was executed on both *train* and *test* sets.

```r
# Transforming 'Store', 'Type', 'Dept', size_type, holiday_week into factors

# Train set
train_set <- train_set %>%
  mutate(Store = as.factor(Store) ,
         Dept = as.factor(Dept),
         Type = as.factor(Type),
         size_type = as.factor(size_type),
         holiday_week = as.factor(holiday_week),
        )

# Test set
test_set <- test_set %>%
  mutate(Store = as.factor(Store) ,
         Dept = as.factor(Dept),
         Type = as.factor(Type),
         size_type = as.factor(size_type),
         holiday_week = as.factor(holiday_week),
  )

# Features post data pre-processing
features_post_pp <- features_post_pp %>%
  mutate(Store = as.factor(Store),
         holiday_week = as.factor(holiday_week))
```

Building the machine learning models

Defining the loss functions - RMSE and WMAE

After having prepared the *training* and *test* sets, the first step toward the development of the machine learning model was choosing the appropriate *loss function*.

Two loss functions were used:

- the *weighted mean absolute error* (**WMAE**)
- the *residual mean squared error* (**RMSE**).

Defined y_i as the actual sales in week i and $\hat{y}_i$ as the predicted sales in week i, the WMAE is defined as:

$$WMAE = \frac{1}{\sum w_i} \sum_i w_i |\hat{y}_i - y_i|$$

with $w_{i} weights - w = 5 if the week is a holiday week, 1 otherwise.

The RMSE is instead defined as

$$RMSE = \sqrt{\frac{1}{N} \sum_i (\hat{y}_i - y_i)^2}$$

where N is the total number of observations.

```r
# Weighted mean absolute error (WMAE)
WMAE <- function(flag, actuals, predictions) {
  w_i <- ifelse(flag ==TRUE, 5,1)
  (1/sum(w_i))*  sum(w_i* abs(actuals - predictions))
}

# Root mean squared error (RMSE)
RMSE <- function(actuals, predictions){
  sqrt(mean((actuals - predictions)^2))
}

# Creating the vector that will be used to calculate the weights
# that indicates which weeks are holidays
holiday_flag <- test_set$IsHoliday
```

Both functions represent the error that could occur when making a prediction. However, while the WMAE gives a very high weight to errors made when predicting sales during holiday weeks, the RMSE gives a relatively high weight to any large errors regardless of the type of week - this is because no weights w_i are assigned and the errors are squared before they are averaged.

The reason as to why both functions were considered, instead of only one, was because a model could have a good RMSE but a less performing WMAE (and viceversa). For instance, a model could be good at predicting the sales in general (good RMSE) but then fail with the predictions in the crucial holiday weeks, thus affecting its WMAE. As holiday weeks have often a strategic meaning in retailing (i.e. higher investments in promotional plan, higher pressure on market share etc.) the WMAE could not have been overlooked, while the RMSE gave an indication of the performance in general.

First model: baseline prediction of mean of the sales across the years

The first model that was created was a simple algorithm that predicted the sales in week number x in the department d of store s by assigning the average sales in week number x across the years in that specific store-department comination. This was a rather elementary model that was clearly not adapted for forecasting future sales, as it ignored key elements of a time-series such as **long-term trends**. For example, if there was an upward trend (the sales in a specific week number were gradually increasing throughout the years), this model would most likely forecast a future value of sales smaller that the one recorded in the last years. This is because by taking the average values, lower sales recorded in the previous years would still be taken into consideration and would offset the increasing trend of the latest years.

Furthermore, the way that this model was designed completely disregarded most of the additional features available in the dataset *features*, such as *Temperature*, *CPI* and *Unemployment*.

Another caveat was that the training set could not contain historical weekly data for each week, due to the randomised sampling method. For this reason, multiple steps were added to predict the sales when there was not enough data for that week. The whole algorithm could be summarised as follow:

1. Sales in *week number* x in department d of store s are predicted to be equal to the **average sales in that specific week number**, store and department across the years;
2. For any week with missing values, sales will be predicted to be equal to the average weekly sales across the **month** of that specific week number, in that store and department;
3. For any missing values, weekly sales will be predicted to be equal to the average weekly sales across the **year** in that specific store and department;
4. Any remaining missing predictions will be replaced with the average weekly sales in that specific store and department **across the whole training set**.

It is clear that at each additional step the predictions lost accuracy, because the resulting calculation derived from more aggregated data (ie. monthly and yearly sales).

Despite all the highlighted flaws, this model was extremely useful to determine a baseline value for the WMAE and RME to use a benchmark for future models.

```r
# Creating the dataset that will contain only the features considered by the model
train_df <- train_set %>%
  select(Store, Dept, Date, Weekly_Sales, Type, holiday_week, year_date, week_date, month_date)

test_df <- test_set %>%
  select(Store, Dept, Date, Weekly_Sales, Type, holiday_week, year_date, week_date, month_date)

# Creating the dataframe with the weekly sales averages
# (1st round of predictions - average weekly sales)
fit_avg_week <- train_df %>%
  group_by(Store, Dept, week_date) %>%
  summarise(predicted_sales = mean(Weekly_Sales))

fit_avg_week <- as.data.frame(fit_avg_week)

# Creating the matrix with the predictions
avg_predictions <- test_df %>%
  left_join(fit_avg_week,
            by = c('Store','Dept','week_date'),
            suffix = c("", "_new")) %>%
  select(Store,Dept,Type, Date, year_date, month_date, week_date,holiday_week, predicted_sales)
```

```
# Checking NAs values
sum(is.na(avg_predictions$predicted_sales))
```

```
## [1] 2344
```

```
# Storing the index
avg_na <- is.na(avg_predictions$predicted_sales)

# -> There a few weeks in the test set whose week number was never included in the training set
# -> For those weeks we will assign the average monthly sales

# Creating the matrix with the predictions
# (2nd round of predictions - average monthly sales)
fit_avg_month <- train_df %>%
  group_by(Store, Dept, month_date) %>%
  summarise(predicted_sales = mean(Weekly_Sales))

# Replacing NAs
avg_predictions[avg_na, "predicted_sales"] <- avg_predictions[avg_na,] %>%
                                              left_join(fit_avg_month,
                                                        by = c('Store','Dept','month_date'),
                                               suffix = c("", "_new")) %>%
                                                select(predicted_sales_new) %>%
                                                 pull()

# Checking NAs values
sum(is.na(avg_predictions$predicted_sales))
```

```
## [1] 77
```

```
# Storing the indexes
avg_na <- is.na(avg_predictions$predicted_sales)
# -> There are still some NAs left.
# -> For those weeks the predicted values will be the average yearly sales
#    in that specific store-department

# Creating the matrix with the predictions
# (3rd round of predictions - average yearly sales )
fit_avg_year <- train_df %>%
  group_by(Store, Dept, year_date) %>%
  summarise(predicted_sales = mean(Weekly_Sales))

# Replacing NAs
avg_predictions[avg_na, "predicted_sales"] <- avg_predictions[avg_na,] %>%
  left_join(fit_avg_year, by = c('Store','Dept','year_date'),
            suffix = c("", "_new")) %>%
  select(predicted_sales_new) %>%
  pull()

# Checking NAs values
sum(is.na(avg_predictions$predicted_sales))
```

```
## [1] 6
```

```r
# Storing the indexes
avg_na <- is.na(avg_predictions$predicted_sales)
# -> few NAs left
# -> These resilient values will be replaced with the average sales
#    in that department store across the whole train_set

# Creating the matrix with the predictions
# (4th round of predictions - average sales across the train_Set)
fit_avg_tot <- train_df %>%
  group_by(Store, Dept,) %>%
  summarise(predicted_sales = mean(Weekly_Sales))

# Replacing NAs
avg_predictions[avg_na, "predicted_sales"] <- avg_predictions[avg_na,] %>%
  left_join(fit_avg_tot, by = c('Store','Dept'),
            suffix = c("", "_new")) %>%
  select(predicted_sales_new) %>%
  pull()

# Checking NAs values
sum(is.na(avg_predictions$predicted_sales))
```

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

```r
# -> No NA values left
```

After the predictions were made, the errors were calculated.

```r
# Calculating the error
avg_wmae <- WMAE(holiday_flag,test_set$Weekly_Sales,avg_predictions$predicted_sales)
avg_wmae
```

```
## [1] 2841.446
```

```r
avg_rmse <- RMSE(test_set$Weekly_Sales, avg_predictions$predicted_sales)
avg_rmse
```

```
## [1] 7113.528
```

```r
# Building a table where we are going to store the WMAEs and RMSEs
# of all the algorithms that we will be created to facilitate a comparison between models
error_results <- tibble(Model = c("avg model"),
                        WMAE = c(avg_wmae),
                        RMSE = c(avg_rmse))
error_results
```

```
## # A tibble: 1 x 3
##   Model      WMAE  RMSE
##   <chr>     <dbl> <dbl>
## 1 avg model 2841. 7114.
```

Second model: linear regression model

The second model that was developed was based on a **linear regression model** approach. Usually, linear regression models are not the most suitable machine learning methods to use for this type of sales forecasting problem for a few reasons:

1. the standard requirements for simple linear regression are not usually met. Specifically, the dependent variable weekly sales does not usually have a linear relationship to the independent variables (dates, temperature, etc.)
2. in regression analysis categorical variables (such as stores, departments, holiday_week, etc) require special attention because, unlike dichotomous or continuous variables, they cannot by entered into the regression equation just as they are

For this reasons, the linear model was initially implemented for illustrative purpose to show a possible workaround to deal with some of the categorical variables and to compare the linear method performance vs more appropriate models.

When implementing the linear model, these features were considered: Store, Dept, Weekly_Sales, year_date, week_date, Fuel_Price, Unemployment, Temperature.

As mentioned in the data exploration analysis, Store and Dept were categorical variables as they could take on only a limited fixed number of possible values - *Store* numbers from 1 to 45 and *Department* numbers from 1 to 99. The workaround used to deal with those categorical variables was to implement a linear model for each department *d* in each store *s*, through the use of *for* loops. Basically, the idea was to decompose one single linear model that use *Store* and *Department* as categorical variables, into *s* x *d* linear models that did not use these variables.

The first loop was needed to consider each store *s* one by one, by filtering the train dataset and consider only the rows of the training set related to store *s*. The second loop was nested within the first loop to filter based on the department, thus considering only one department *d* at each iteration. Once the linear model for the department *d* in store *s* was fitted, the predictions were made within the second loop and added onto a dataset that was containing all the predictions made thus far (through the *rbind* function).

```r
# Creating the dataframe that contains only the predictors
# that will be the data source of the train function
train_df <- train_set %>%
  select(Store, Dept, Weekly_Sales, Date, Fuel_Price, Unemployment,
         Temperature, CPI, year_date, week_date )

# Setting up the test test in the same way
test_df  <- test_set %>%
  select(Store, Dept, Weekly_Sales, Date, Fuel_Price, Unemployment,
         Temperature, CPI, year_date, week_date)

# Initialising the indexes for the loops
s=1 # Index for the stores
d=1 # Index for the department

# Initialising dataframe that will contain all predictions
predictions_outcome <- data.frame(matrix(nrow=0, ncol=5))
colnames(predictions_outcome) = c("Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales")

# ncol = 5 as the columns will be "Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales"
# nrow = nrow(test_df) because for each entry of test_df a prediction will be calculated
```

```r
for (s in 1:45) {

  # Retrieving the list of deparments of store s
  dept <- train_df %>%
    filter(Store==s) %>%
    select(Dept) %>%
    distinct() %>%
    pull()

  # Loop to fit a linear regression model for each department d in store s
  for (d in dept) {
    set.seed(87, sample.kind="Rounding")

    fit_lm <- train_df %>%
      filter(Store==s,
             Dept==d) %>%
      train(Weekly_Sales ~ year_date + week_date + Fuel_Price + Unemployment + Temperature + CPI,
            data=.,
            method="lm")

    store_s_dept_d_df <- test_df %>%
      filter(Store==s,
             Dept==d)

    predict_lm <- store_s_dept_d_df %>%
      predict(fit_lm,
              newdata=.)

    # Populating  dataset with predictions and features
    predictions_lm_store_s_dept_d <- cbind(store_s_dept_d_df[,c("Store", "Dept",
                                                             "Date", "Weekly_Sales")],
                                         predict_lm)
    # -> we are including Weekly sales for the time being
    #    to facilitate the calculation of the error

    # Storing the predictions
    predictions_outcome <- rbind(predictions_outcome,predictions_lm_store_s_dept_d)
  }
}
```

```
## Error in check_dims(x = x, y = y): nrow(x) > 1 is not TRUE
```

After having calculated all the predicted sales, the errors were calculated. Prior to the calculation of the WMAE, the flag highlighting whether a date was a holiday week was retrieved - as it was critical to the calculation of the weights.

```r
# Calculating the errors

# Looking up the relevant holiday_week flag
predictions_outcome_plus <- predictions_outcome %>%
  left_join(features_post_pp, by = c("Store", "Date"))
```

```r
# Checking NAs
sum(is.na(predictions_outcome_plus$isHoliday))
```

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

```r
holiday_flag_lm <- predictions_outcome_plus$IsHoliday

lm_wmae <- WMAE(holiday_flag_lm,predictions_outcome_plus$Weekly_Sales,
                predictions_outcome_plus$predict_lm)
lm_wmae
```

```
## [1] 2759.53
```

```r
lm_rmse <- RMSE(predictions_outcome_plus$Weekly_Sales,
                predictions_outcome_plus$predict_lm )
lm_rmse
```

```
## [1] 5964.497
```

```r
# Adding this WMAE to the table where we are storing all WMAE previously calculated
error_results <- tibble(Model = c("Avg model", "Linear model"),
                        WMAE = c(avg_wmae,lm_wmae),
                        RMSE = c(avg_rmse, lm_rmse))
error_results
```

```
## # A tibble: 2 x 3
##   Model         WMAE  RMSE
##   <chr>        <dbl> <dbl>
## 1 Avg model    2841. 7114.
## 2 Linear model 2760. 5964.
```

As expected the performance of this model were better than the first model. It is worth noting that improved more the RMSE rather then the WMAE - 13% improvement vs the 2.6% of the latter. This meant that the linear model improved predictions in general, whilst still struggling with predicting weeks within the holiday periods.

Third model: KNN model

The third model that was implemented was a regression based on the k-nearest neighbors (KNN) method. An approach based on the KNN regression is in general more suitable than the linear model when there is no linear relationship between the variables. However, as in the case of the linear regression, the KNN model tend to struggle when handling categorical variables.

The same *loop-based* approach used for the regression model allowed to consider the features *Store* and *Department* without adding these directly within the knn train function. These features were considered: Store, Department, year_date, week_date.

Within the train function, a *trainControl* function was used to determine the re-sampling method - this was set to use a 10-fold cross validation process. The *tuneLength*, instead, was set to 5 to tell the train algorithm to try 5 different default values of the main parameter k. Finally the *preProcess* function was used to center and scale the predictors.

```r
# Creating the dataframe that contains only the predictors
# that will be the data source of the train function
train_df <- train_set %>%
  select(Store, Dept, Weekly_Sales, Date, Fuel_Price,
         Unemployment, Temperature, CPI, year_date, week_date )

# Setting up the test test in the same way
test_df  <- test_set %>%
  select(Store, Dept, Weekly_Sales, Date, Fuel_Price,
         Unemployment, Temperature, CPI, year_date, week_date )

# Initialising the indexes for the loops
s=1 # Index for the stores
d=1 # Index for the department

# Initialising dataframe that will contain all predictions
predictions_outcome <- data.frame(matrix(nrow=0, ncol=5))
colnames(predictions_outcome) = c("Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales")

# ncol = 5 as the columns will be "Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales"
# nrow = nrow(test_df) because for each entry of test_df a prediction will be calculated


# Initialising the indexes for the loops
s=1 # Index for the stores
d=1 # Index for the department

# Initialising dataframe that will contain all predictions
predictions_outcome <- data.frame(matrix(nrow=0, ncol=5))
colnames(predictions_outcome) = c("Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales")

# ncol = 5 as the columns will be "Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales"
# nrow = nrow(test_df) because for each entry of test_df a prediction will be calculated


for (s in 1:45) {

  # Retrieving the list of deparments of store s
  dept <- train_df %>%
    filter(Store==s) %>%
    select(Dept) %>%
    distinct() %>%
    pull()

  # Loop to fit a knn regression model for each department d in store s
  for (d in dept) {
    set.seed(87, sample.kind="Rounding")

    fit_knn_dept <- train_df %>%
      filter(Store==s,
             Dept==d) %>%
      train(Weekly_Sales ~ year_date + week_date + Fuel_Price + Unemployment + Temperature + CPI,
            data=.,
            method="knn",
            trControl = trainControl("cv", number = 10),
            preProcess = c("center","scale"),
            tuneLength = 10)

    store_s_dept_d_df <- test_df %>%
      filter(Store==s,
             Dept==d)

    predict_knn_dept <- store_s_dept_d_df %>%
      predict(fit_knn_dept,
              newdata=.)

    # Populating  dataset with predictions and features
    predictions_store_s_dept_d <- cbind(store_s_dept_d_df[,c("Store", "Dept",
                                                              "Date", "Weekly_Sales")],
                                        predict_knn_dept)
    # -> we are including Weekly sales for the time being
    #    to facilitate the calculation of the error

    # Storing the predictions
    predictions_outcome <- rbind(predictions_outcome,
                                 predictions_store_s_dept_d)
  }
}
```

```
## Error in cut.default(y, breaks, include.lowest = TRUE): invalid number of intervals
```

```
# Calculating the errors

# To be able to calculate the wmae we first need to lookup the relevant holiday_flag
predictions_outcome_plus <- predictions_outcome %>%
  left_join(features_post_pp, by = c("Store", "Date"))

# Checking NAs
sum(is.na(predictions_outcome_plus$isHoliday))
```

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

```
holiday_flag_knn <- predictions_outcome_plus$IsHoliday

knn_wmae <- WMAE(holiday_flag_knn,
                 predictions_outcome_plus$Weekly_Sales,
                 predictions_outcome_plus$predict_knn_dept)
knn_wmae
```

```
## [1] 3161.386
```

```
knn_rmse <- RMSE(predictions_outcome_plus$Weekly_Sales,
                 predictions_outcome_plus$predict_knn_dept )
knn_rmse
```

```
## [1] 6701.977
```

```
# Adding this WMAE to the table where we are storing all WMAE previously calculated
error_results <- tibble(Model = c("Avg model", "Linear model", "KNN model"),
                        WMAE = c(avg_wmae,lm_wmae, knn_wmae),
                        RMSE = c(avg_rmse, lm_rmse, knn_rmse))
error_results
```

```
## # A tibble: 3 x 3
##   Model         WMAE  RMSE
##   <chr>        <dbl> <dbl>
## 1 Avg model    2841. 7114.
## 2 Linear model 2760. 5964.
## 3 KNN model    3161. 6702.
```

Surprisingly, the values of both WMAE and RMSE resulted higher than the case of the linear model. KNN should have theoretically performed better as there was no linear relationship between the variables. Subsequently, other variants of this KNN model were tested by dropping some of the features and increasing the number of values of the parameter k that were evaluated. The best performing model was given by a simpler model that included only the engineered features *year_date* and *week_date* and a *tuneLength* equal to 20.

```
# Initialising the indexes for the loops
s=1 # Index for the stores
d=1 # Index for the department

# Initialising dataframe that will contain all predictions
predictions_outcome <- data.frame(matrix(nrow=0, ncol=5))
colnames(predictions_outcome) = c("Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales")

for (s in 1:45) {

  # Retrieving the list of deparments of store s
  dept <- train_df %>%
    filter(Store==s) %>%
    select(Dept) %>%
    distinct() %>%
    pull()

  # Loop to fit a knn regression model for each department d in store s
  for (d in dept) {
    set.seed(87, sample.kind="Rounding")

    fit_knn_dept <- train_df %>%
                    filter(Store==s,
                           Dept==d) %>%
                    train(Weekly_Sales ~ year_date + week_date ,
                    data=.,
                    method="knn",
                    trControl = trainControl("cv", number = 10),
                    preProcess = c("center","scale"),
                    tuneLength = 20)
```

```r
    store_s_dept_d_df <- test_df %>%
                         filter(Store==s,
                         Dept==d)

    predict_knn_dept <- store_s_dept_d_df %>%
                       predict(fit_knn_dept,
                               newdata=.)

    # Populating  dataset with predictions and features
    predictions_store_s_dept_d <- cbind(store_s_dept_d_df[,c("Store", "Dept",
                                                             "Date", "Weekly_Sales")],
                                        predict_knn_dept)
    # -> we are including Weekly sales for the time being
    #    to facilitate the calculation of the error

    # Storing the predictions
    predictions_outcome <- rbind(predictions_outcome,
                                 predictions_store_s_dept_d)
  }
}
```

```
## Error in cut.default(y, breaks, include.lowest = TRUE): invalid number of intervals
```

```r
# Calculating the errors

# To be able to calculate the wmae we first need to lookup the relevant holiday_flag
predictions_outcome_plus <- predictions_outcome %>%
                            left_join(features_post_pp,
                                      by = c("Store", "Date"))

# Checking NAs
sum(is.na(predictions_outcome_plus$isHoliday))
```

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

```r
holiday_flag_knn <- predictions_outcome_plus$IsHoliday

knn_wmae_imp <- WMAE(holiday_flag_knn,
                     predictions_outcome_plus$Weekly_Sales,
                     predictions_outcome_plus$predict_knn_dept)
knn_wmae_imp
```

```
## [1] 2978.306
```

```r
knn_rmse_imp <- RMSE(predictions_outcome_plus$Weekly_Sales,
                     predictions_outcome_plus$predict_knn_dept )
knn_rmse_imp
```

```
## [1] 6525.378
```

```r
# Adding this WMAE to the table where we are storing all WMAE previously calculated
error_results <- tibble(Model = c("Avg model", "Linear model", "KNN model", "KNN improved model"),
                        WMAE = c(avg_wmae,lm_wmae, knn_wmae, knn_wmae_imp),
                        RMSE = c(avg_rmse, lm_rmse, knn_rmse, knn_rmse_imp))
error_results
```

```
## # A tibble: 4 x 3
##   Model               WMAE  RMSE
##   <chr>              <dbl> <dbl>
## 1 Avg model          2841. 7114.
## 2 Linear model       2760. 5964.
## 3 KNN model          3161. 6702.
## 4 KNN improved model 2978. 6525.
```

After those adjustments, both the WMAE and RMSE improved slightly but were still higher than anticipated. One of the possible reason for this underperformance vs the linear model could be due to the absence of a significant number of observations in some departments within certain stores. In some instances the number of observations (weekly sales) available to train the model for a specific department was scarce, thus affecting the ability of the KNN algorithm to find an optimal fit.

Fourth model: Random Forest

The next model that was explored was *random forest*. Random Forest was thought to be a good model to approach a multi-variate time series problem, such as the Walmart sales forecasting project. The way that Random Forest handles categorical variables allowed to explore and include predictors such as *holiday_week*, *size_type* etc. that could not be included in the previous models. However, compared to the other regression methods, random forest required higher computational resources. As the workstation that was available for this project had rather limited resources, the study of this model remained merely theoretical.

The tuning of the model was divided into a multi-step process:

1. Default setting - A random forest model with default settings was trained
2. Tuning mtry - The model was then trained and tested for different values of the *mtry* parameter by using a *tunegrid* function. The value of mtry that gave the highest accuracy was then stored
3. Tuning maxnode - set the value of mtry equal to the value found in step 2, the accuracy of the model was then evaluated for different values of *max nodes* though the use of a *for* loop
4. Tuning ntrees - the same approach used for tuning max nodes was adopted to tune the *ntrees* parameter

```r
# Creating the dataframes that contain only the predictors
# that will be the source data of the train function
train_df <- train_set %>%
         select(Store, Dept, Weekly_Sales, Type, holiday_week,
                year_date, week_date, month_date )
 test_df  <- test_set %>%
          select(Store, Dept, Weekly_Sales, Type, holiday_week,
                 year_date, week_date, month_date )

#Defining the control (10-folds cross validation)
trControl <- trainControl(method = "cv", number = 10, search ="grid")


# Random Forest Models ----
# --  RF Model 1 - Default Settings --
# Training
set.seed(87, sample.kind="Rounding")
train_rf_def <- train(Weekly_Sales ~ .,
                        data=train_df,
                        method = "rf",
                        importance= TRUE,
                        trControl = trControl
)

# Testing
# Making predictions
predicted_sales <- predict(train_rf_def, test_df)

# Calculating the error
WMAE(holiday_flag,test_set$Weekly_Sales,predicted_sales)


# -- RF Model 2 - Tuning best mtry --
tuneGrid <- expand.grid(.mtry = c(1: 10))
# The model will be tested for values of mtry from 1 to 10

# Training
set.seed(87, sample.kind="Rounding")
train_rf_2 <- train(Weekly_Sales ~ .,
                       data=train_df,
                       method = "rf",
                       importance= TRUE,
                       trControl = trControl,
                       tuneGrid = tuneGrid
)

# Best value of mtry
train_rf_2$bestTune$mtry

# Store this value of mtry to used it in future tuning
best_mtry <- train_rf_2$bestTune$mtry


# Testing
# Making predictions
predicted_sales <- predict(train_rf_2, test_df)

# Calculating the error
WMAE(holiday_flag,test_set$Weekly_Sales,predicted_sales)


# -- RF Model 3 - Tuning best maxnodes --
# Initialising the list where the random forest models trained for each value of max node
# will be stored
store_maxnode <- list()

# Setting the tuneGrid to use the best value of mtry previously found
```

```r
tuneGrid <- expand.grid(.mtry = best_mtry)

# Training
set.seed(87, sample.kind="Rounding")

# Creating the loop which will fit a random forest model
# for each value of maxnodes from 5 to 15

for (maxnodes in c(5: 15)) {
  train_rf_maxnode <- train(Weekly_Sales ~ .,
                      data=train_df,
                      method = "rf",
                      importance= TRUE,
                      trControl = trControl,
                      tuneGrid = tuneGrid,
                      maxnodes = maxnodes)
  current_iteration <- toString(maxnodes)
  store_maxnode[[current_iteration]] <- rf_maxnode
}

results_mtry <- resamples(store_maxnode) # Arranging the result of the model
summary(results_mtry)

# From the summary the value of maxnodes that maximise the accuracy can be selected,
# stored in the varianble 'tuned_maxnodes' and used to make the predictions

train_rf_maxnode  <- train(Weekly_Sales ~ .,
                      data=train_df,
                      method = "rf",
                      importance= TRUE,
                      trControl = trControl,
                      tuneGrid = tuneGrid,
                      maxnodes = tuned_maxnodes)

# Testing
# Making predictions
predicted_sales <- predict(train_rf_maxnode, test_df)

# Calculating the error
WMAE(holiday_flag,test_set$Weekly_Sales,predicted_sales)


# -- RF Model 4 - Tuning best ntrees --
# The tuning of the max number of ntrees follow the same procedure used to tune max nodes.

# Initialising the list where the random forest models trained for each value of ntrees
# will be stored
store_maxtrees <- list()

# Setting the tuneGrid to use the best value of mtry previously found
tuneGrid <- expand.grid(.mtry = best_mtry)

# Training
```

```
set.seed(87, sample.kind="Rounding")

# Creating the loop which will fit a random forest model for each value of ntree

for (ntree in c(250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000)) {
  train_rf_maxtrees <- train(Weekly_Sales ~ .,
                      data=train_df,
                      method = "rf",
                      importance= TRUE,
                      trControl = trControl,
                      tuneGrid = tuneGrid,
                       ntrees = ntree
                      maxnodes = tuned_maxnodes)
  current_iteration <- toString(ntrees)
  store_ntrees[[current_iteration]] <- rf_maxtrees
}

# results_tree <- resamples(store_maxtrees) # Arranging the result of the model
summary(results_tree)

# From the summary the value of ntree that maximise the accuracy can be selected,
# stored in the varianble 'tuned_ntree' and used to make the predictions

train_rf_ntree  <- train(Weekly_Sales ~ .,
                      data=train_df,
                      method = "rf",
                      importance= TRUE,
                      trControl = trControl,
                      tuneGrid = tuneGrid,
                      maxnodes = tuned_maxnodes
                      ntree= tuned_ntree
)

# Testing
# Making predictions
predicted_sales <- predict(train_rf_ntree, test_df)

# Calculating the error
WMAE(holiday_flag,test_set$Weekly_Sales,predicted_sales)
```

Fifth model: ARIMA

An ARIMA based approach was the final model that was explored. ARIMA is one of the most widely used approaches to time series forecasting, therefore, it was thought to be worth to be examined even though this model was not introduced in the course.

ARIMA models aim to describe the autocorrelations in the data and are generally used to make forecasts rather than predicting sales in the past. The nature of the algorithm requires that the data are transformed into a time series format and that there are no missing observations (weeks). Thus, creating a train/test set for time-series problems becomes tricky because the time component has to be taken into account. A random sampled train-test split or k-fold validation would disrupt the pattern in the series. For this reason, when training an ARIMA model, it does not make much sense to randomly split the data into training and test set, as it was done for the previous models.

Therefore, a different training set was used to train the following ARIMA model. In order to have uninterrupted time series to train the model, the new *train set* was formed by all observations with a date earlier than 1st January 2012, while the rest of observation determined the new *test set*.

```
# Creating the dataframe that contains only the predictors
# that will be the data source of the train function
train_ts <- train %>%
  select(Store, Dept, Date, Weekly_Sales,IsHoliday)

# Splitting into train and test set.
# Observations taken in 2012 will be used as the test set
test_df <- train_ts %>%
  filter(Date >= as.Date("2012-01-01"))

train_df <- train_ts %>%
  filter(Date < as.Date("2012-01-01"))

# Calculating test / train ratio
nrow(test_df) / (nrow(test_df)+ nrow(train_df))
```

```
## [1] 0.3022938
```

```
# -> the test set is 30% of the train set
```

Due to the different samples method, the results of this model were not really comparable with the outcomes from regression and knn models. Nevertheless, the procedure and results were reported in this report to show an alternative and, most likely, most suitable approach to a time-series forecasting problem.

ARIMA is an acronym that stands for auto-regressive integrated moving average and is specified by three order parameters (p, d, q). These parameters can be identified manually, however R includes a handy **auto.arima()** function that automates the process by searching through combinations of order parameters and picking the set that optimises model fit criteria. This function is part of the *forecast* package in R. Once the model is fitted, the forecast can be made by using the *forecast()* function.

In the specific case of the Walmart forecasting problem, a distinct ARIMA model was needed to be fitted for each department d in store s. The same *2-loops* approach used in the previous models was implemented to iterate the fitting of the ARIMA model for each store-department combination. One important difference was that, within the second loop, the sales data for department d in store s were transformed into a time-series through the function *ts()*, before being fed to the auto.arima function. Note that the parameter *frequency* indicated the number of observations before the seasonal pattern repeats - as the the data were collected on a weekly basis and each seasonality repeated once a year, frequency was set equal to 52 (weeks in a year).

```
# Initialising the indexes needed for the loops
s=1
d=1
# Initialising the dataframe where the predictions will be stored
predictions_outcome_arima <- data.frame(matrix(nrow=0, ncol=6))

for (s in 1:45) {

  # Retrieving the list of deparments of store s
  dept <- train_df %>%
    filter(Store==s) %>%
    select(Dept) %>%
    distinct() %>%
    pull()

  # Loop to fit a Arima model for each department d in store s
  for (d in dept) {
    set.seed(87, sample.kind="Rounding")

    # Extracting real sales for this department d and store s
    # (to use for calculating the error)
    test_s_d <- test_df  %>%
      filter(Store==s, Dept==d) %>%
      arrange(Date) %>%
      select(Store, Dept, Date, Weekly_Sales, IsHoliday)

    # Counting number of weeks to predict
    n_weeks <- nrow(test_s_d)

    # Retrieving first available date (test set)
    min_date <- train_df %>%
      filter(Store==s, Dept==d) %>%
      select(Date) %>%
      summarise(max_date=min(Date)) %>%
      pull()

    # Extracting the weekly sales sorted by dates
    arima_rawdata <- train_df %>%
      filter(Store==s,
             Dept==d) %>%
      arrange(Date) %>%
      select( Weekly_Sales)

    # Converting the data into time series data format
    tsData <- ts(arima_rawdata,
                 start = c(year(min_date),month(min_date), day(min_date)),
                 frequency = 52) # weekly data

    # Fitting Arima model
    fit_arima <- auto.arima(tsData)

    # Making Predictions
    # (only if the department d - store s combination is included within the test set)
    if (n_weeks>0) {
    sales_forecast_model <- forecast(fit_arima, h=n_weeks)

    predicted_sales <- as.numeric(sales_forecast_model$mean)

    # Populating  dataset with predictions and features
    predictions_store_s_dept_d <- cbind(test_s_d,
                                        as.data.frame(predicted_sales))
    # -> we are including Weekly sales for the time being
    #    to facilitate the calculation of the error

    # Storing those predictions
```

```r
    predictions_outcome_arima <- rbind(predictions_outcome_arima,
                                       predictions_store_s_dept_d)}
  }
}
```

The error was calculated with the same WMAE and RMSE functions previously used.

```r
# Calculating the errors

arima_wmae <- WMAE(predictions_outcome_arima$IsHoliday,
                   predictions_outcome_arima$Weekly_Sales,
                   predictions_outcome_arima$predicted_sales)
arima_wmae
```

```
## [1] 2986.836
```

```r
arima_rmse <- RMSE(predictions_outcome_arima$Weekly_Sales,
                   predictions_outcome_arima$predicted_sales )
arima_rmse
```

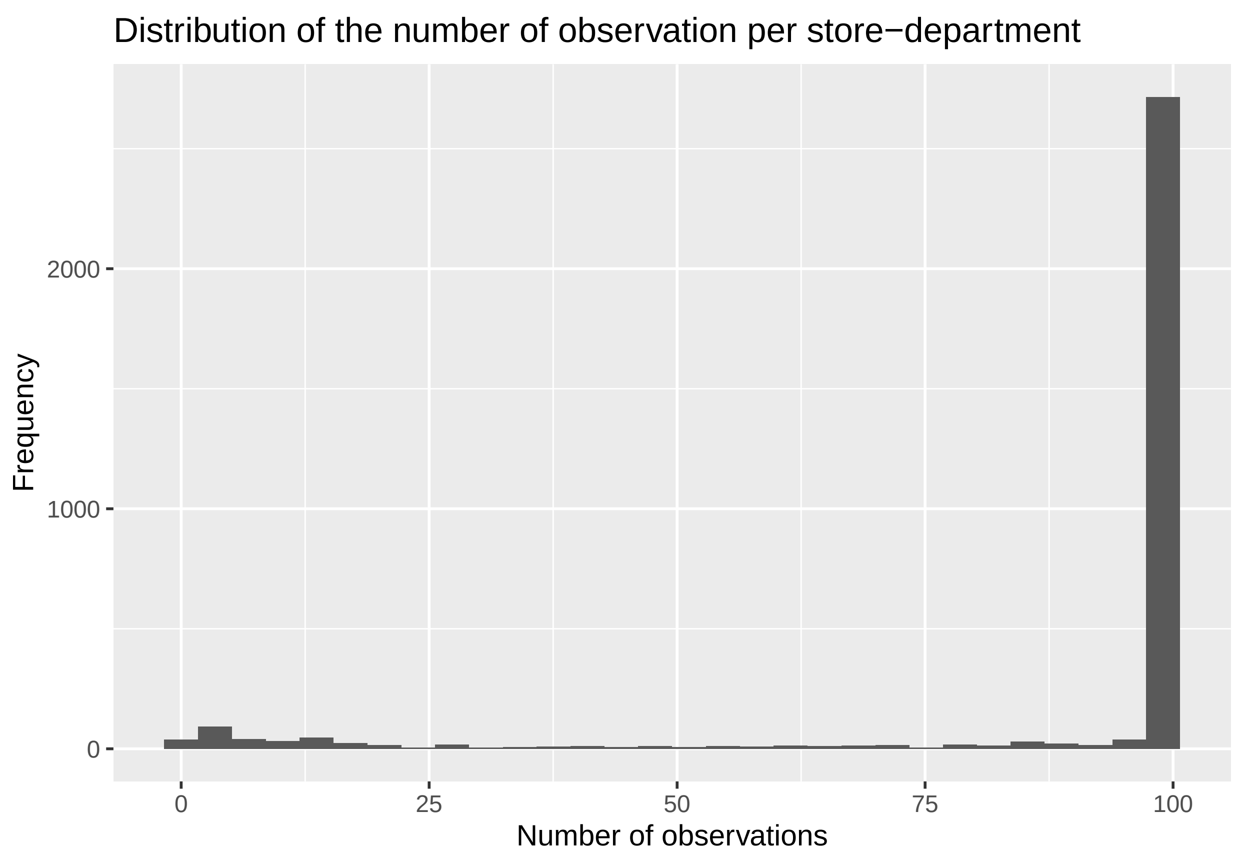
```
## [1] 6156.939
```

The value of the WMAE error was higher than the figures obtained with the knn and linear regression models, whilst the RMSE was actually the lowest found so far. While the ARIMA model was expected to perform better in terms of WMAE, it is worth noting a couple of caveats:

1. the models used different train and test sets, with different sample sizes (10/90 vs 30/70 split) and methods (Random vs Non-random). Hence, the errors should never been compared in the first place. For instance, models trained with larger datasets (as in the case of the of the knn and linear regression) tend to perform better in general.
2. certain departments-store combinations did not have the full set of observations for the whole time-period (5th Feb 2010 - 30th December 2011), thus the model fitted in those cases were skewed and consequently the predictions sub-optimal. As it could be seen from the chart, in some instances there were less than 100 weeks of sales reported for a specific department-store combination.

```r
# Check that every combination department-store have the same amount of observations
# with no missing sales for any week
train_df %>%
  group_by(Store,Dept) %>%
  summarise(n_row = n()) %>%
  ggplot(aes(x=n_row)) +
  geom_histogram() +
  ggtitle("Distribution of the number of observation per store-department") +
  xlab("Number of observations") +
  ylab("Frequency")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
train_df %>%
  group_by(Store,Dept) %>%
  summarise(n_row = n()) %>%
  arrange(-desc(n_row))
```

```
## # A tibble: 3,302 x 3
## # Groups:   Store [45]
##    Store  Dept n_row
##    <dbl> <dbl> <int>
##  1     3    78     1
##  2     4    39     1
##  3     5    78     1
##  4     6    77     1
##  5     7    78     1
##  6    10    77     1
##  7    13    43     1
##  8    13    77     1
##  9    14    43     1
## 10    14    96     1
## # ... with 3,292 more rows
```

```r
# -> sales data for certain stores are only available for the smaller time periods
```

Despite some flaws, this model could represent a base to build upon for future work. For example, an additional piece of code that deals more effectively with missing values and shorter or interrupted sales time series could be developed.

Nevertheless, in order to compare this ARIMA approach with the other models, the best model fitted so far (linear model approach) was retrained using the same train dataset of the ARIMA model. After that, the predictions were made and the errors were estimated by using the same test set used in the case of the ARIMA approach.

```r
# Creating the dataframe that contains only the predictors
# that will be the data source of the train function
train_ts <- train %>%
  left_join(stores, by ="Store",
            suffix = c("", "_new")) %>%
  left_join(features, by = c("Store","Date"),
            suffix = c("", "_new"))  %>%
  # Adding the features engineered in the data exploration
  mutate(week_date = week(Date),
         month_date = month(Date),
         year_date = year(Date)) %>%
  mutate(Store = as.factor(Store) ,
         Dept = as.factor(Dept),
         Type = as.factor(Type),
         size_type = as.factor(size_type),
         holiday_week = as.factor(holiday_week)) %>%
  select(Store, Dept, Weekly_Sales, Date, Fuel_Price, Unemployment,
         Temperature, CPI, year_date, week_date )

# Splitting into train and test set. Observations taken in 2012 will be used as the test set
# (Same split as in the case of the ARIMA model)
test_df <- train_ts %>%
  filter(Date >= as.Date("2012-01-01"))

train_df <- train_ts %>%
  filter(Date < as.Date("2012-01-01"))

# Initialising the indexes for the loops
s=1 # Index for the stores
d=1 # Index for the department

# Initialising dataframe that will contain all predictions
predictions_outcome <- data.frame(matrix(nrow=0, ncol=5))
colnames(predictions_outcome) = c("Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales")

# ncol = 5 as the columns will be "Store", "Dept", "Date", "Weekly_Sales", "Predicted Sales"
# nrow = nrow(test_df) because for each entry of test_df a prediction will be calculated

for (s in 1:45) {

  # Retrieving the list of deparments of store s
  dept <- train_df %>%
    filter(Store==s) %>%
    select(Dept) %>%
    distinct() %>%
    pull()

  # Loop to fit a linear regression model for each department d in store s
  for (d in dept) {
    set.seed(87, sample.kind="Rounding")

    fit_lm <- train_df %>%
      filter(Store==s,
             Dept==d) %>%
      train(Weekly_Sales ~ year_date + week_date + Fuel_Price +
                           Unemployment + Temperature + CPI,
            data=.,
            method="lm")

    store_s_dept_d_df <- test_df %>%
      filter(Store==s,
             Dept==d)

    predict_lm <- store_s_dept_d_df %>%
      predict(fit_lm,
              newdata=.)

    # Populating  dataset with predictions and features
    predictions_lm_store_s_dept_d <- cbind(store_s_dept_d_df[,c("Store", "Dept",
                                                              "Date", "Weekly_Sales")],
                                          predict_lm)
    # -> we are including Weekly sales to facilitate the calculation of the error

    predictions_outcome <- rbind(predictions_outcome,
                                 predictions_lm_store_s_dept_d)
  }
}
```

```
## Error in check_dims(x = x, y = y): nrow(x) > 1 is not TRUE
```

```
# Calculating the errors

# To be able to calculate the wmae we first need to lookup the relevant holiday_flag
predictions_outcome_plus <- predictions_outcome %>%
  left_join(features_post_pp, by = c("Store", "Date"))

# Checking NAs
sum(is.na(predictions_outcome_plus$isHoliday))
```

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

```
holiday_flag_lm <- predictions_outcome_plus$IsHoliday

lm_v2_wmae <- WMAE(holiday_flag_lm,
                   predictions_outcome_plus$Weekly_Sales,
                   predictions_outcome_plus$predict_lm)
lm_v2_wmae
```

```
## [1] 3476.041
```

```r
lm_v2_rmse <- RMSE(predictions_outcome_plus$Weekly_Sales,
                   predictions_outcome_plus$predict_lm )
lm_v2_rmse
```

```
## [1] 6309.626
```

```r
# Creating the tables that contains the errors of the models
# that used the second split test-train set
error_results_2 <- tibble(Model = c("ARIMA", "Linear model (revisited)"),
                          WMAE = c(arima_wmae, lm_v2_wmae),
                          RMSE = c(arima_rmse, lm_v2_rmse))

error_results_2
```

```
## # A tibble: 2 x 3
##   Model                     WMAE  RMSE
##   <chr>                    <dbl> <dbl>
## 1 ARIMA                    2987. 6157.
## 2 Linear model (revisited) 3476. 6310.
```

The new found value of the WMAEs and RMSEs showed that the ARIMA model actually performed better than the linear regression approach, when these two models where trained and tested on the same datasets. Therefore, it was demonstrated that an approach based on the ARIMA algorithm could be a more suitable model for a time-series forecasting problem.

RESULTS SUMMARY

The development of a machine learning algorithm able predict to future sales of 45 Walmart stores resulted in a considerably harder challenge than anticipated. A variety of approaches were considered:

- imputation of averages
- linear model regression
- KNN regression
- Random Forest
- ARIMA

The complex structure of the data, along with missing information (e.g. Markdown) and the lack of a consistent number of observations across all department and stores, greatly hindered the ability to find a definitive solution to the problem. As a direct consequence, for example, the results of the loss function in the ARIMA model could only be compared with a re-trained version of the linear model approach. Furthermore, the value of the loss function for the Random Forest model could not be obtained due to the extremely high computational resources that were needed.

Thus, excluding those two models, based uniquely on the value of the WMAE and RMSE estimated by using randomised sampled train-test datasets, the best performing model was an algorithm that fitted a linear model for the weekly sales of every department within every store. Thus, this model was a rather complex collection of 3331 different linear regressions models. All of these sub-models used the same set of predictors

- year_date

- week_date
- Fuel_Price
- Unemployment
- Temperature
- CPI

The use of the engineered features *week_date* and *year_date*, instead of the simpler feature *date*, helped to explain some of the variability due to seasonal and cyclical character of the sales data. For instance, higher sales in the 6^{th} week of the year could be explained by the fact that it marked the week with Valentine's day, as opposed to the date itself (as the 6^{th} week could fall in any date from the 10^{th} to the 20^{th} of February). The other features *Fuel_Price*, *Unemployment*, *Temperature* and *CPI* were also included because the data exploratory analysis had highlighted how those differed from store to store, while failing to determining with certainty their predictive power (therefore there was a mix of trial and error involved).

The results of the loss functions were the following:

- WMAE = 2766.894
- RMSE = 6195.120

This meant that the weekly sales predicted by the algorithm were on average \$6.195 higher or lower than the real sales contained in the *test* set. Given that the average weekly sales across all department and store was \$ 16.000, the final results could not be considered completely satisfactory and left wide room for improvement.

However, the are some aspects of the model performance that cannot be highlighted from the value of the WMAE and RMSE, but that deserve some attention:

- the code underlying this machine learning model is **quick to execute**. The first advantage of this property is that this forecasting model could be run by nearly anyone without heavily relying on the machine's specification. Secondly, this allows to easily *retrain* the algorithm once new data become available. Retraining is an important process in machine learning, as things change over time (e.g. disruptive events such as the Coronavirus pandemic) and the data distributions can at some point deviate significantly from those of the original training set. Therefore, the accuracy of the model can start to decline, if something changes in the context used to train the algorithm. Since the code to train the model is simple and quick to execute, the algorithm can be retrained often and without major difficulties

- the model is not directly **interpretable**. This is a disadvantage derived from the use of the nested *for* loops. What it means is that it is not very easy to understand at glance how the algorithm assigns certain sales. However, if we were to decompose the algorithm and isolate a specific store-department entry, the underlying linear model would allow an easy interpretation of the forecasted sales. The ability to interpret an erroneous prediction could help understand the cause of an error, thus delivering a direction for how to fix the algorithm should something go wrong in the future when the model is used.

Finally, it was also showed that an ARIMA based approach could actually be a more suitable solution for a time-series forecasting problem. While the performance of this last approach could not be directly compared with the findings from the other models, due to the different sampling method, it was showed that the ARIMA model did outperform the best algorithm when both were trained on the same training set.

CONCLUSION

In summary, a supervised machine learning model that was able to predict future sales of 45 Walmart stores, located in different regions, was developed in R. After having explored the datasets and identified the predictors to use in the machine learning algorithm, the data were pre-processed in order to impute missing values, engineer new predictors and join the different tables into one. After that, some modelling approaches were described. Specifically, five different algorithms were explored: a 3-steps imputational model, an algorithm which combined the results of thousands of linear models, an approach based on the k-nearest neighbours (KNN) algorithm, a Random Forest model and finally an ARIMA based algorithm. Some limitations, such us limited computational resources available and missing of key data, have hindered the ability to effectively estimate the value of the error functions for the Random Forest and ARIMA models respectively. Therefore, the choice of the final algorithm was based on the value of the error functions of the first three models. Among those, surprisingly the best performing machine learning model was the algorithm that utilised a set of thousands linear models to forecast the sales. However, it was also demonstrated how this final algorithm was actually less accurate than the model based on ARIMA, when both models were trained on the same training sets.

While the simplicity of the final model represented one of its strengths, in terms of retraining abilities and potential for interpretability, it was also one of its limitations as it precluded the possibility to take into consideration more sophisticated patterns and effects that could determine a more accurate forecast. For this reason, the possibility of using more advanced and suitable machine learning methods should be explored in future work. For instance, a mixed approached that combines the ARIMA method introduced in this study with other methodologies that could deal better with incomplete/interrupted time-series should be explored. Furthermore, it is suggested the exploration of the Random Forest model that was developed in this study, through the use of an appropriate workstation designed for machine learning purposes. Finally, given the multivariate time-series nature of the forecasting problem, the evaluation of a vector autoregressive (VAR) model and/or an AutoRegressive Moving Average Vector (VARMA) model could be explored. This could provide the opportunity to explore and include some predictors such as *holiday_week*and *size_type* that the exploratory analysis had shown to have an influence on sales, but that could not have been included in the final model due to the limitations of the selected method.