Project DS502

Abhishek Shah, Mahdi Alouane, Rahul Pande and Sam Longenbach

Import the training and testing datasets while converting white spaces to NAs as well

Then we first check for the presence of NA values.

```
##           Item_Identifier                Item_Weight
##                         0                       1463
##          Item_Fat_Content            Item_Visibility
##                         0                          0
##                 Item_Type                   Item_MRP
##                         0                          0
##         Outlet_Identifier Outlet_Establishment_Year
##                         0                          0
##               Outlet_Size       Outlet_Location_Type
##                      2410                          0
##               Outlet_Type          Item_Outlet_Sales
##                         0                          0
```

```
##           Item_Identifier                Item_Weight
##                         0                        976
##          Item_Fat_Content            Item_Visibility
##                         0                          0
##                 Item_Type                   Item_MRP
##                         0                          0
##         Outlet_Identifier Outlet_Establishment_Year
##                         0                          0
##               Outlet_Size       Outlet_Location_Type
##                      1606                          0
##               Outlet_Type
##                         0
```

We have missing values in `Item_Weight` and `Outlet_Size`.

Now, we check the frequencies of categorical variables.

```
## $Item_Fat_Content
##
##      LF low fat Low Fat     reg Regular
##     316     112    5089     117    2889
##
## $Item_Type
##
##          Baking Goods                Breads             Breakfast
##                   648                   251                   110
##                Canned                 Dairy          Frozen Foods
##                   649                   682                   856
## Fruits and Vegetables           Hard Drinks    Health and Hygiene
##                  1232                   214                   520
##             Household                  Meat                Others
##                   910                   425                   169
##               Seafood           Snack Foods           Soft Drinks
##                    64                  1200                   445
##         Starchy Foods
```

```
##                    148
##
## $Outlet_Identifier
##
## OUT010 OUT013 OUT017 OUT018 OUT019 OUT027 OUT035 OUT045 OUT046 OUT049
##    555    932    926    928    528    935    930    929    930    930
##
## $Outlet_Size
##
##   High Medium  Small
##    932   2793   2388
##
## $Outlet_Location_Type
##
## Tier 1 Tier 2 Tier 3
##   2388   2785   3350
##
## $Outlet_Type
##
##     Grocery Store Supermarket Type1 Supermarket Type2 Supermarket Type3
##              1083              5577               928               935
```

We aggregate on outlet level to impute outlet size

```
## character(0)
```

```
## character(0)
```

We see that there are no new stores in the test data that are not already encountered in the training data.

```
## -- Attaching packages ---------------------------------------------------------------------------------

## √ ggplot2 3.0.0     √ purrr   0.2.5
## √ tibble  1.4.2     √ dplyr   0.7.6
## √ tidyr   0.8.1     √ stringr 1.3.1
## √ readr   1.2.1     √ forcats 0.3.0

## -- Conflicts ------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## # A tibble: 10 x 3
##    Item_Identifier l_u_weights u_weights
##    <chr>                 <int> <chr>
##  1 FDE52                     1 NA
##  2 FDK57                     1 NA
##  3 FDN52                     1 NA
##  4 FDQ60                     1 NA
##  5 FDT16                     2 9.895 | NA
##  6 NCJ54                     2 9.895 | NA
##  7 DRD49                     1 9.895
##  8 FDR13                     1 9.895
##  9 FDA23                     2 9.8 | NA
## 10 FDC10                     2 9.8 | NA
```

We see that in some places weights are NA whereas it is not NA in other rows for the same item. We can just use weights from other observations where weight is not NA (For the same item). For this purpose the whole train and test datasets have been used to impute the missing information. Hence, we define the following function for treating these missing values.

Then, we call it and check if the problem is resolved.

```
##          Item_Identifier               Item_Weight
##                        0                         0
##         Item_Fat_Content           Item_Visibility
##                        0                         0
##                Item_Type                  Item_MRP
##                        0                         0
##        Outlet_Identifier Outlet_Establishment_Year
##                        0                         0
##              Outlet_Size      Outlet_Location_Type
##                     2410                         0
##              Outlet_Type         Item_Outlet_Sales
##                        0                         0
```

```
##          Item_Identifier               Item_Weight
##                        0                         0
##         Item_Fat_Content           Item_Visibility
##                        0                         0
##                Item_Type                  Item_MRP
##                        0                         0
##        Outlet_Identifier Outlet_Establishment_Year
##                        0                         0
##              Outlet_Size      Outlet_Location_Type
##                     1606                         0
##              Outlet_Type
##                        0
```

Since, there is no more missing values for the feature `Item_Weight`, we treat in the following section the missing values for the feature `Outlet_Size`.

Given that `Outlet_Size` is an outlet specific attribute, we first begin by printing all the outlets available in our training set (10 outlets in total). Hence, we figure out that the 2410 missing values in training set belong to only 3 outlets and the size of these outlets is also missing in the testing set.

```
##    Outlet_Identifier Outlet_Size
## 1             OUT049      Medium
## 2             OUT018      Medium
## 4             OUT010        <NA>
## 5             OUT013        High
## 8             OUT027      Medium
## 9             OUT045        <NA>
## 10            OUT017        <NA>
## 12            OUT046       Small
## 20            OUT035       Small
## 24            OUT019       Small
```

In order to achieve this, we start by transforming the categorical attributes into dummy variables using `One Hot Encoding` as shown below:

```
## Loading required package: lattice
```

```
## Loading required package: grid
```

```
##  [1] "Outlet_Establishment_Year_1999" "Outlet_Establishment_Year_2009"
##  [3] "Outlet_Establishment_Year_1998" "Outlet_Establishment_Year_1987"
##  [5] "Outlet_Establishment_Year_1985" "Outlet_Establishment_Year_2002"
```

```
##  [7] "Outlet_Establishment_Year_2007" "Outlet_Establishment_Year_1997"
##  [9] "Outlet_Establishment_Year_2004" "Outlet_Location_Type_Tier 1"
## [11] "Outlet_Location_Type_Tier 3"    "Outlet_Location_Type_Tier 2"
## [13] "Outlet_Size_Medium"             "Outlet_Size_High"
## [15] "Outlet_Size_Small"              "Outlet_Type_Supermarket Type1"
## [17] "Outlet_Type_Supermarket Type2"  "Outlet_Type_Grocery Store"
## [19] "Outlet_Type_Supermarket Type3"
```

Then, we predict the missing values for `Outlet_Size` using the K-Nearest Neighbors with K=5. The algorithm reaches out for the 5 closest neighbors (after scaling) for each observation where the attribute is missing and according to a vote assigns a score using a weighted average (`meth='weighAvg'`). Therefore, we compute the maximum among the three possible values (Small, Medium and High) and assign it to the corresponding observation.

We can see here the missing values and their prediction according to 5NN.

```
##    Outlet_Identifier Outlet_Establishment_Year Outlet_Size
## 4             OUT010                      1998      Medium
## 9             OUT045                      2002       Small
## 10            OUT017                      2007       Small
##    Outlet_Location_Type       Outlet_Type
## 4                Tier 3     Grocery Store
## 9                Tier 2 Supermarket Type1
## 10               Tier 2 Supermarket Type1
```

```
##   Outlet_Size_Medium Outlet_Size_High Outlet_Size_Small
## 3          0.4116651        0.2144811         0.3738537
## 6          0.1643578        0.1728448         0.6627974
## 7          0.1643578        0.1728448         0.6627974
```

Finally, we check to see that there is still any missing values:

```
##           Item_Identifier               Item_Weight
##                         0                         0
##          Item_Fat_Content           Item_Visibility
##                         0                         0
##                 Item_Type                  Item_MRP
##                         0                         0
##         Outlet_Identifier Outlet_Establishment_Year
##                         0                         0
##               Outlet_Size      Outlet_Location_Type
##                         0                         0
##               Outlet_Type         Item_Outlet_Sales
##                         0                         0
```

We fill the missing values in the testing set with the above-predicted values for each outlet as shown below:

After this, we check if there is any missing values in the testing set:

```
##           Item_Identifier               Item_Weight
##                         0                         0
##          Item_Fat_Content           Item_Visibility
##                         0                         0
##                 Item_Type                  Item_MRP
##                         0                         0
##         Outlet_Identifier Outlet_Establishment_Year
##                         0                         0
##               Outlet_Size      Outlet_Location_Type
```

```
##                            0                            0
##                  Outlet_Type
##                            0
```

Since there are no more missing values we proceed further with data cleaning. After observing the Item_Fat_Content, we found that different labels represented same information. To fix that, remap the labels to only two logically significant labels, namely, low_fat and regular.

Feature Engineering:

1. Since the Outlet_Establishment_Year is in years, which is logically numeric, we transform it to the Years_Operating and then drop the column.
2. We created a feature named Item_Cat which represents the Category of the Item. It's created from the first two letters of Item_Identifier labels which represents the category of the products.
3. We then observed that some Non-consumables have either low_fat or regular, which doesn't really make sense. So we changed those labels accordingly.
4. We observed that a lot of Items with 0% visibility that have made sales. We fixed that by taking the aggregated visibility of the same item and setting the visibility to the obtained aggregated value

Applying the same steps on the test data set, so that are models stay healthy for test set as well.

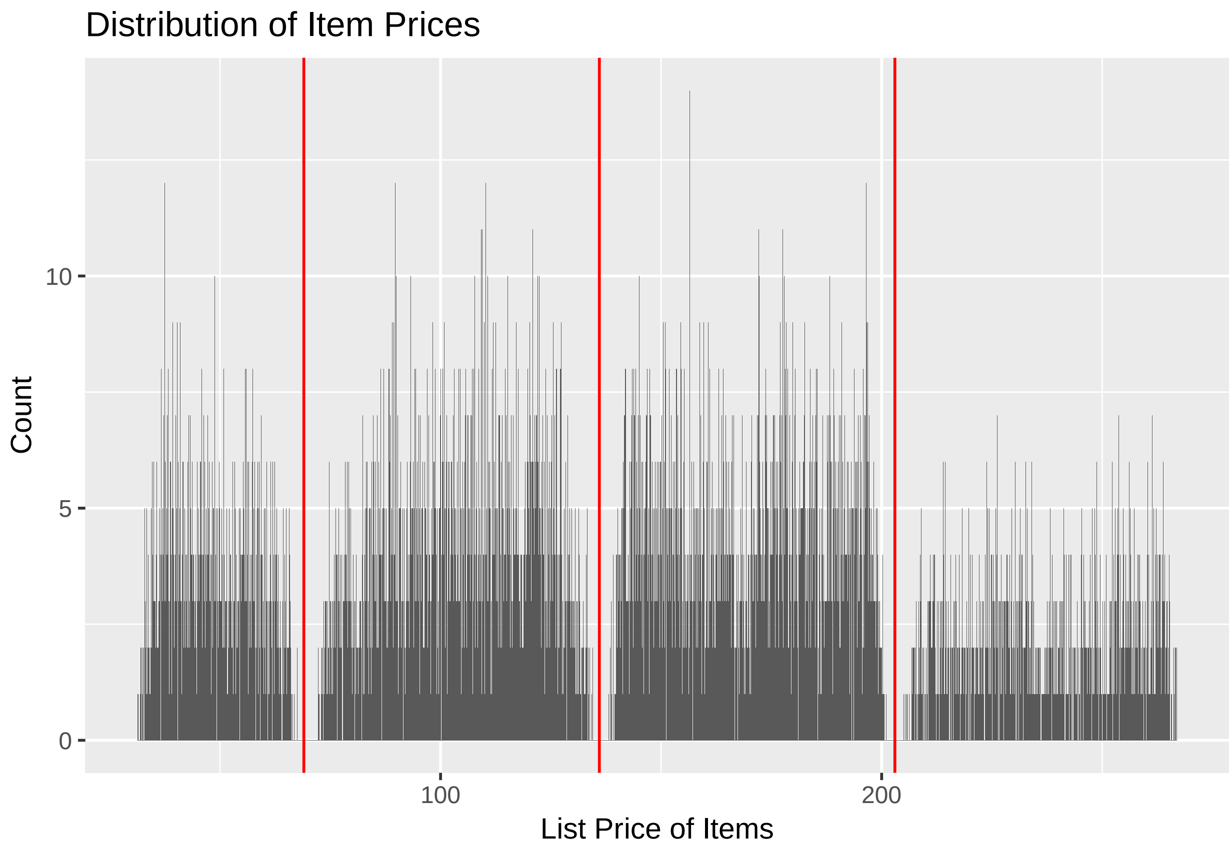
```
## [1] "Item_Outlet_Sales"
```

Finally, we split sample from our training set 1000 observations that we are going to keep aside(in the vault).

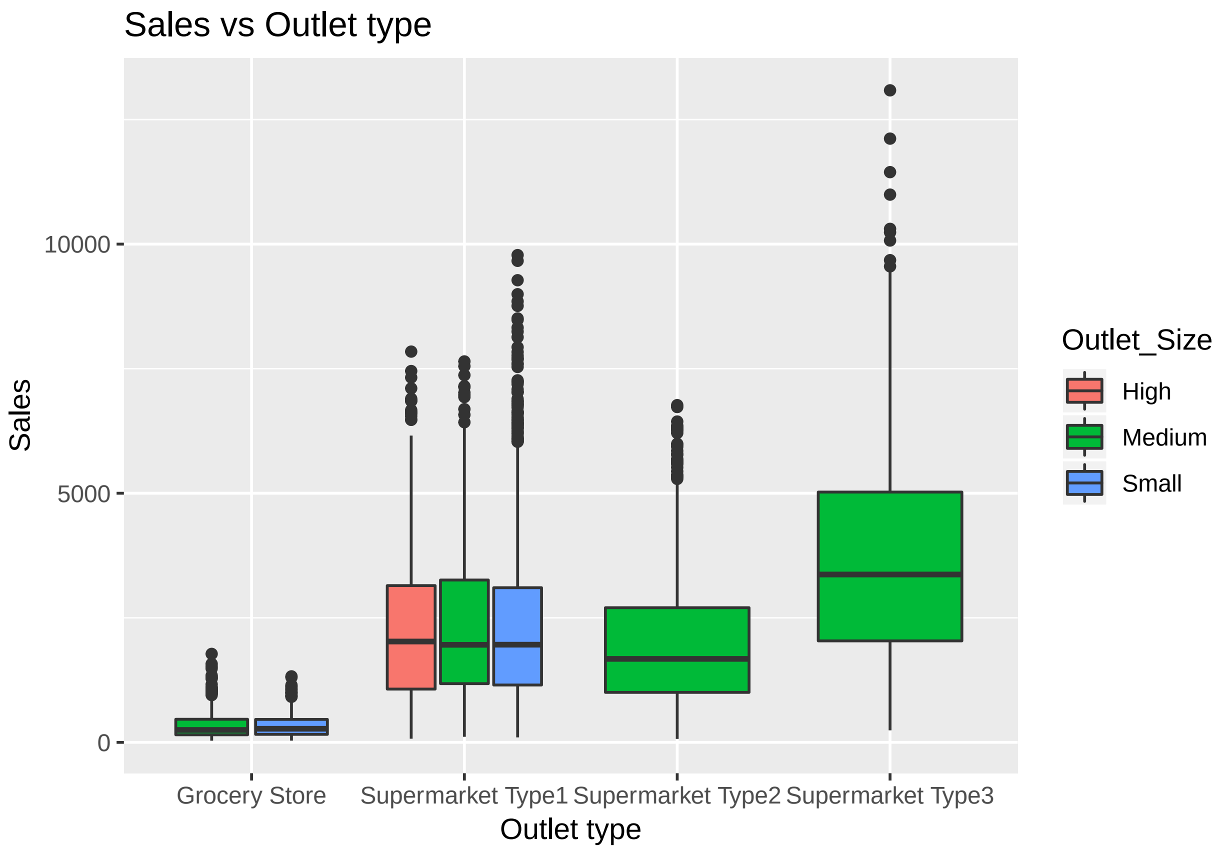
Data Exploration

First, we started by looking at the data to find any interesting relationships between our predictors.

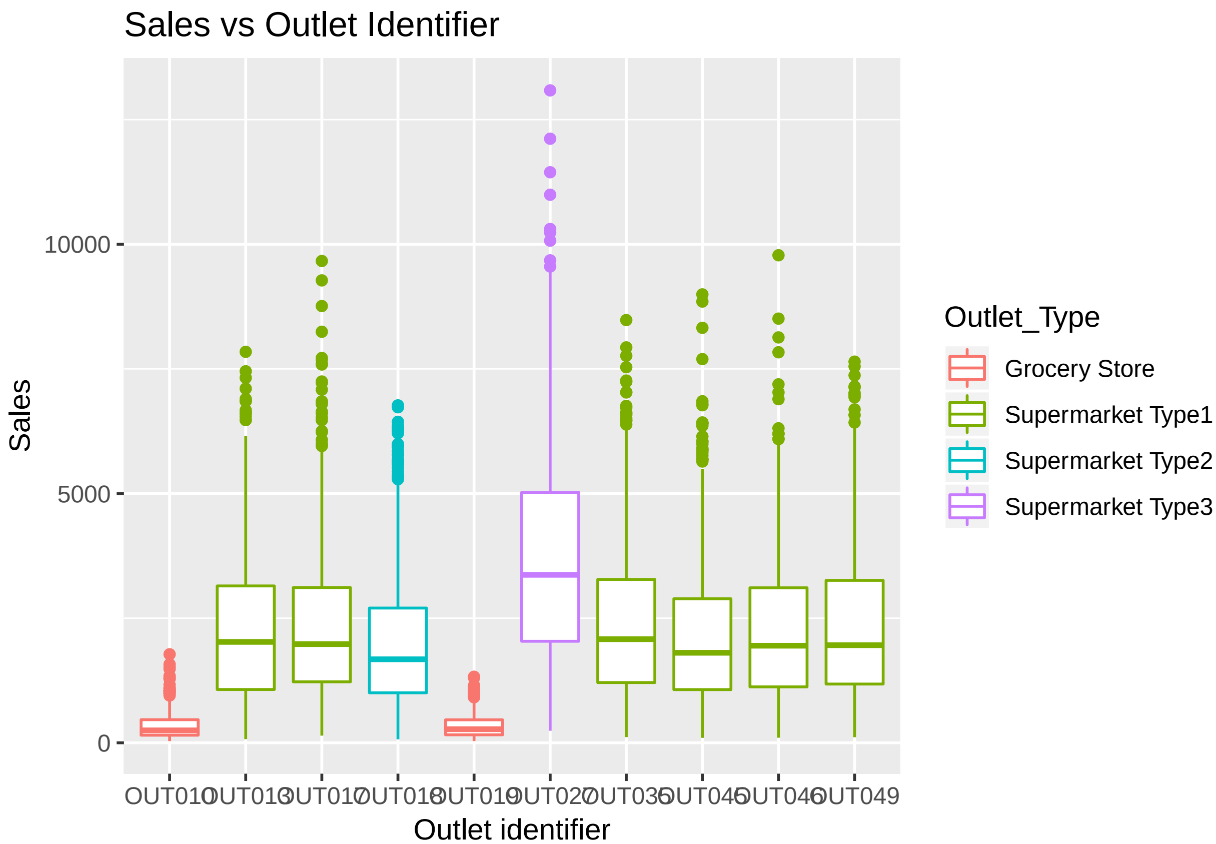
One of the most obvious relashionships is looking at the distribution Retail Price of the Items (Item_MRP) in our training data. We observe that there are 4 major ranges of Item_MRP accross all the items.



We also try to plot the sales against the type of outlet colored based on the outlet size. We get some interesting observations such as sales for a given Outlet Size appear to be similar across Outlet Type. For example, Supermarket Type 1 has all three Outlet Sizes which all have about the same Sales.



Next are the sales of each of the 10 outlets. The intuition behind the plot was to observe which outlets perform well and which do not. Through this plot we see that the two outlets that have extremely low sales are the Grocery Stores.



Simple Linear Models

```
##
## Call:
## lm(formula = Item_Outlet_Sales ~ ., data = subset(train.data,
##     select = -Item_Identifier))
##
## Residuals:
##     Min       1Q   Median       3Q      Max
## -4387.2  -672.6    -83.8    572.0   7914.6
##
## Coefficients: (10 not defined because of singularities)
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -1837.6908   148.9258 -12.340   <2e-16 ***
## Outlet_IdentifierOUT013        1952.7800    81.0305  24.099   <2e-16 ***
## Outlet_IdentifierOUT017        2035.6996    80.0415  25.433   <2e-16 ***
## Outlet_IdentifierOUT018        1680.3542    80.0340  20.995   <2e-16 ***
## Outlet_IdentifierOUT019          32.6504    74.4328   0.439    0.661
## Outlet_IdentifierOUT027        3392.0917    82.0306  41.352   <2e-16 ***
## Outlet_IdentifierOUT035        2069.6792    80.5799  25.685   <2e-16 ***
## Outlet_IdentifierOUT045        1852.9558    80.7307  22.952   <2e-16 ***
## Outlet_IdentifierOUT046        1942.2389    80.4290  24.148   <2e-16 ***
## Outlet_IdentifierOUT049        2018.1183    80.8640  24.957   <2e-16 ***
## Item_TypeBreads                 -40.3447    89.2015  -0.452    0.651
## Item_TypeBreakfast               62.9687   122.7321   0.513    0.608
## Item_TypeCanned                  15.9567    66.6322   0.239    0.811
```

```
## Item_TypeDairy                   -57.4553    70.2549  -0.818    0.413
## Item_TypeFrozen Foods            -51.1511    62.2969  -0.821    0.412
## Item_TypeFruits and Vegetables    15.4739    58.0536   0.267    0.790
## Item_TypeHard Drinks             -44.6524   149.1452  -0.299    0.765
## Item_TypeHealth and Hygiene      -38.4097   137.0151  -0.280    0.779
## Item_TypeHousehold               -96.9179   132.2051  -0.733    0.464
## Item_TypeMeat                    -17.2454    74.9751  -0.230    0.818
## Item_TypeOthers                  -64.0078   155.2662  -0.412    0.680
## Item_TypeSeafood                  33.9115   158.8165   0.214    0.831
## Item_TypeSnack Foods             -44.5760    58.5189  -0.762    0.446
## Item_TypeSoft Drinks             -82.9700   137.7023  -0.603    0.547
## Item_TypeStarchy Foods            75.2463   108.3818   0.694    0.488
## Item_Weight                        0.7692     2.8213   0.273    0.785
## Item_Fat_Contentnot_edible             NA         NA      NA       NA
## Item_Fat_Contentregular           48.0357    30.2212   1.589    0.112
## Item_Visibility                 -271.1061  1130.4802  -0.240    0.810
## Item_MRP                          15.7435     0.2109  74.649   <2e-16 ***
## Outlet_SizeMedium                      NA         NA      NA       NA
## Outlet_SizeSmall                       NA         NA      NA       NA
## Outlet_Location_TypeTier 2             NA         NA      NA       NA
## Outlet_Location_TypeTier 3             NA         NA      NA       NA
## Outlet_TypeSupermarket Type1           NA         NA      NA       NA
## Outlet_TypeSupermarket Type2           NA         NA      NA       NA
## Outlet_TypeSupermarket Type3           NA         NA      NA       NA
## Years_Operating                        NA         NA      NA       NA
## Item_CatFD                       -55.9546   117.4637  -0.476    0.634
## Item_CatNC                             NA         NA      NA       NA
## Item_Visibility_Ratio              9.5951    80.9093   0.119    0.906
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1124 on 7492 degrees of freedom
## Multiple R-squared:  0.5657, Adjusted R-squared:  0.5639
## F-statistic: 325.2 on 30 and 7492 DF,  p-value: < 2.2e-16

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading
```

```
## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(linear.fit, newdata = subset(train.data, select = -
## Item_Identifier)[folds == : prediction from a rank-deficient fit may be
## misleading
```

```
## [1] 1126.717
```

From the linear model, we can see that the sales of a particular item depends mainly on the store where it is sold and what the MRP is. The rmse is 1124. $R^2 = 0.56$. So only 56% of the variance in the output is explained by this linear model

```
##
## Call:
## lm(formula = Item_Outlet_Sales ~ Item_MRP, data = subset(train.data,
##     select = -Item_Identifier))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3885.0  -755.1   -58.0   688.6  9432.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -22.0421    39.9676  -0.551    0.581
## Item_MRP     15.6440     0.2598  60.208   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1398 on 7521 degrees of freedom
## Multiple R-squared:  0.3252, Adjusted R-squared:  0.3251
## F-statistic:  3625 on 1 and 7521 DF,  p-value: < 2.2e-16
```

From the summary, we see that R^2 has dropped to 0.3, far less than the previous linear model with all the variables.

```
##
## Call:
## lm(formula = Item_Outlet_Sales ~ Outlet_Size * Outlet_Type, data = subset(train.data,
##     select = -Item_Identifier))
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max 
## -3468.6 -1066.0  -193.9   674.6  9376.7 
## 
## Coefficients: (5 not defined because of singularities)
##                                               Estimate Std. Error t value
## (Intercept)                                     312.02      90.54   3.446
## Outlet_SizeMedium                                31.19     113.41   0.275
## Outlet_SizeSmall                                 29.30      57.49   0.510
## Outlet_TypeSupermarket Type1                   1960.85      74.58  26.291
## Outlet_TypeSupermarket Type2                   1669.52      85.86  19.445
## Outlet_TypeSupermarket Type3                   3367.07      86.03  39.138
## Outlet_SizeMedium:Outlet_TypeSupermarket Type1   23.11     113.73   0.203
## Outlet_SizeSmall:Outlet_TypeSupermarket Type1       NA         NA      NA
## Outlet_SizeMedium:Outlet_TypeSupermarket Type2      NA         NA      NA
## Outlet_SizeSmall:Outlet_TypeSupermarket Type2       NA         NA      NA
## Outlet_SizeMedium:Outlet_TypeSupermarket Type3      NA         NA      NA
## Outlet_SizeSmall:Outlet_TypeSupermarket Type3       NA         NA      NA
##                                                Pr(>|t|)    
## (Intercept)                                    0.000572 ***
## Outlet_SizeMedium                              0.783310    
## Outlet_SizeSmall                               0.610338    
## Outlet_TypeSupermarket Type1                    < 2e-16 ***
## Outlet_TypeSupermarket Type2                    < 2e-16 ***
## Outlet_TypeSupermarket Type3                    < 2e-16 ***
## Outlet_SizeMedium:Outlet_TypeSupermarket Type1 0.838965    
## Outlet_SizeSmall:Outlet_TypeSupermarket Type1        NA    
## Outlet_SizeMedium:Outlet_TypeSupermarket Type2       NA    
## Outlet_SizeSmall:Outlet_TypeSupermarket Type2        NA    
## Outlet_SizeMedium:Outlet_TypeSupermarket Type3       NA    
## Outlet_SizeSmall:Outlet_TypeSupermarket Type3        NA    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1489 on 7516 degrees of freedom
## Multiple R-squared:  0.2354, Adjusted R-squared:  0.2348 
## F-statistic: 385.7 on 6 and 7516 DF,  p-value: < 2.2e-16
```

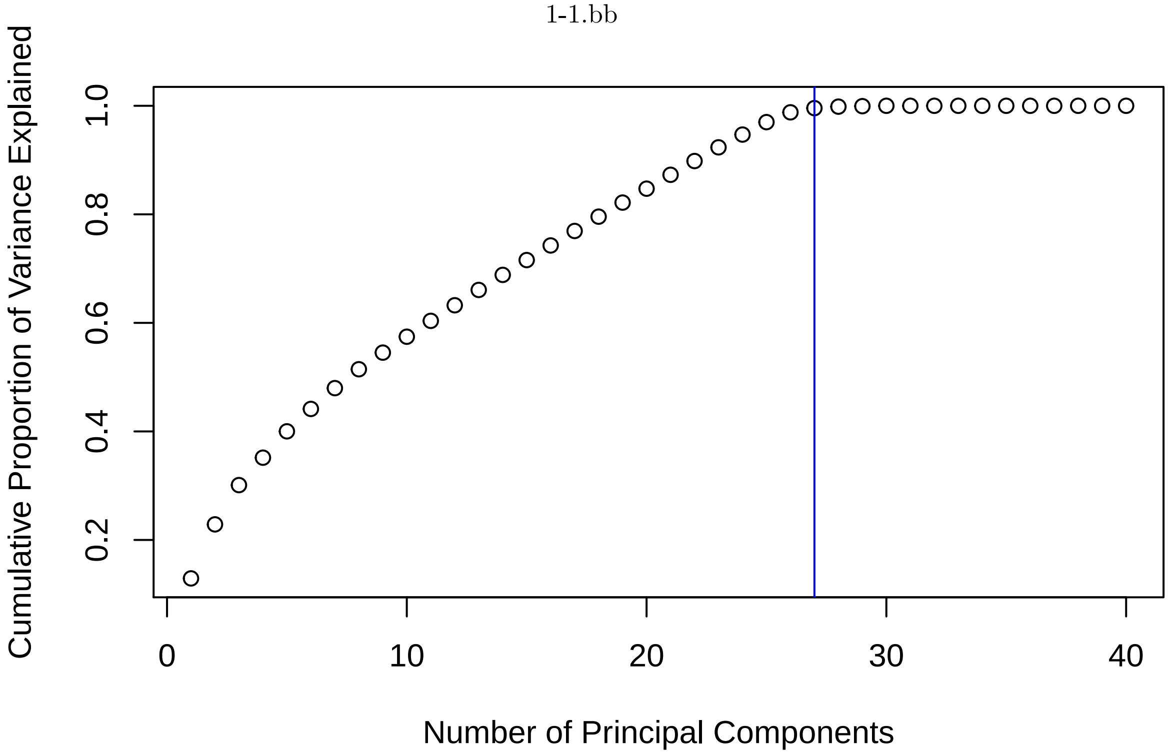
From the summary we see that the interactions are `NA` due to collinearity. That means the interation variables are some linear combination of the other variables and to solve the normal equation. We will park it for now, and then manually one hot encode and add infinitesimal noise to the dummy variables for the linear model to keep them.

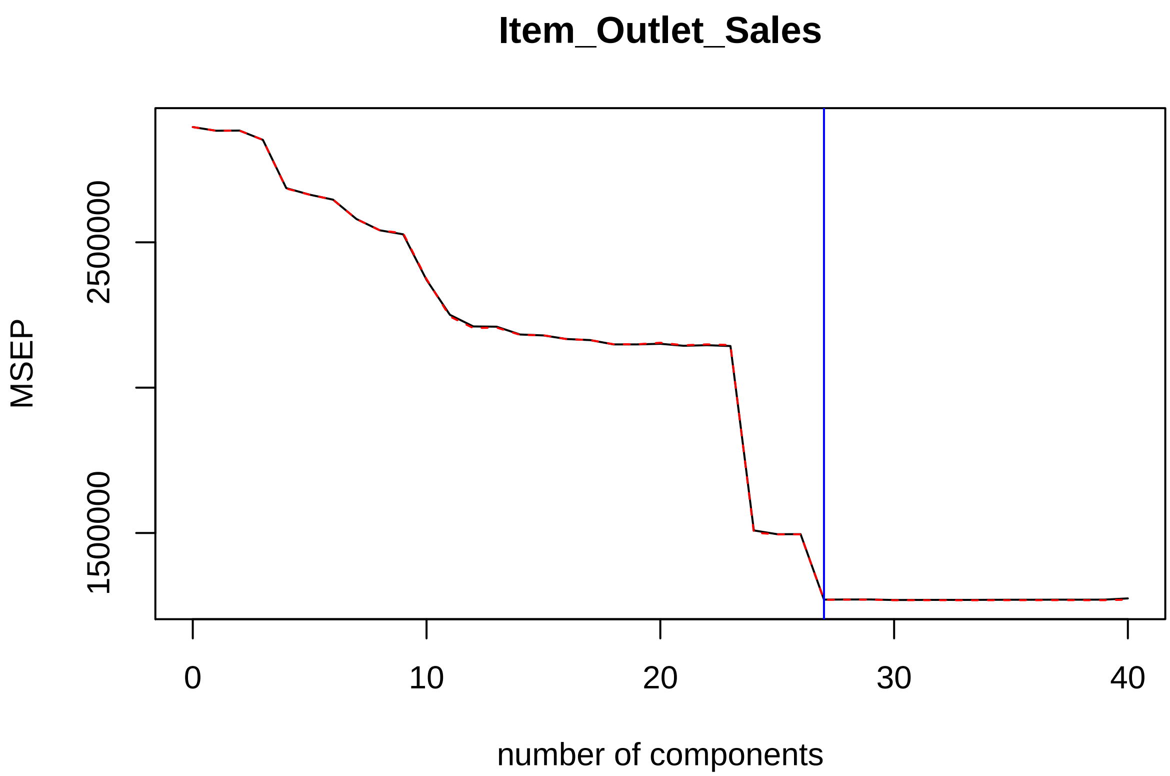
Principal Component Regression

In this section, we fit a Principal Component Regression model. First, we start by find the number of principal components needed to maximize the variance explanation and minimizing the MSE while keeping a reasonable number of components. In our case, the number of Principal Components chosen could be 27 (corresponds to the knee in the curve) as shown on the graphs below:

```
## 
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
## 
##     loadings
```

1-1.bb





1-2.bb

Once the `ncomp` value chosen, we proceed to a 10-fold cross validation process where we estimate the RMSE for a Principal Component Regression model fitted with the first 27 PCs.

```
## [1] 1127.49
```

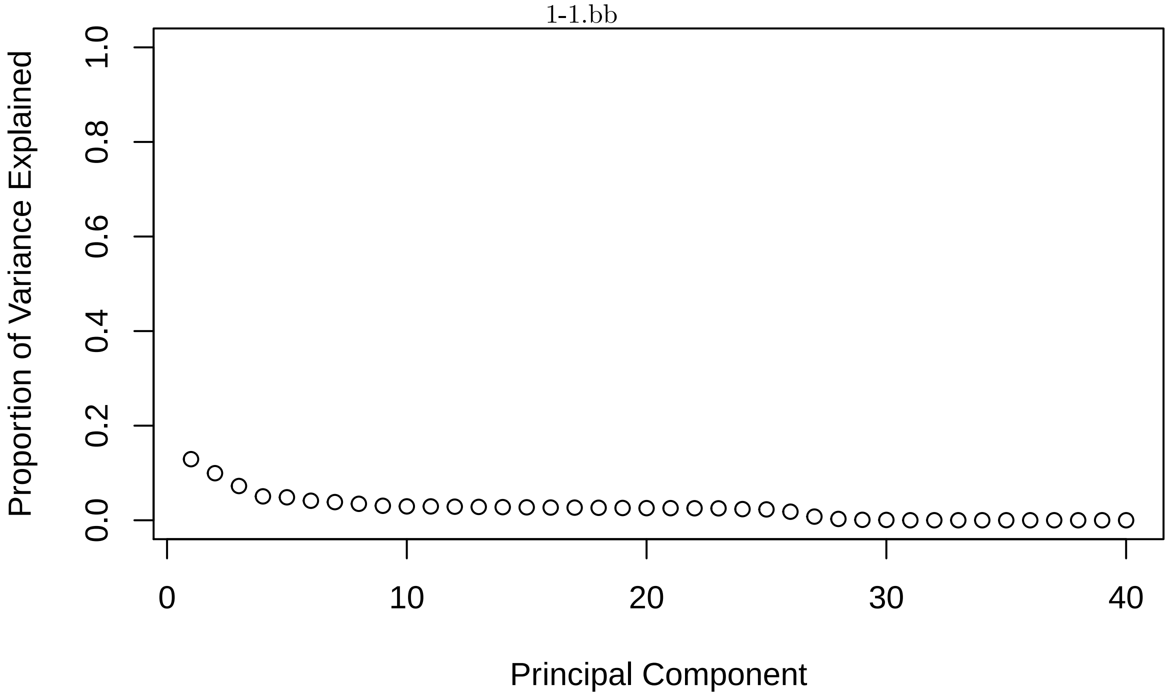
We can observe that the estimated value of the RMSE for this model is around `1126.671`.

PCA

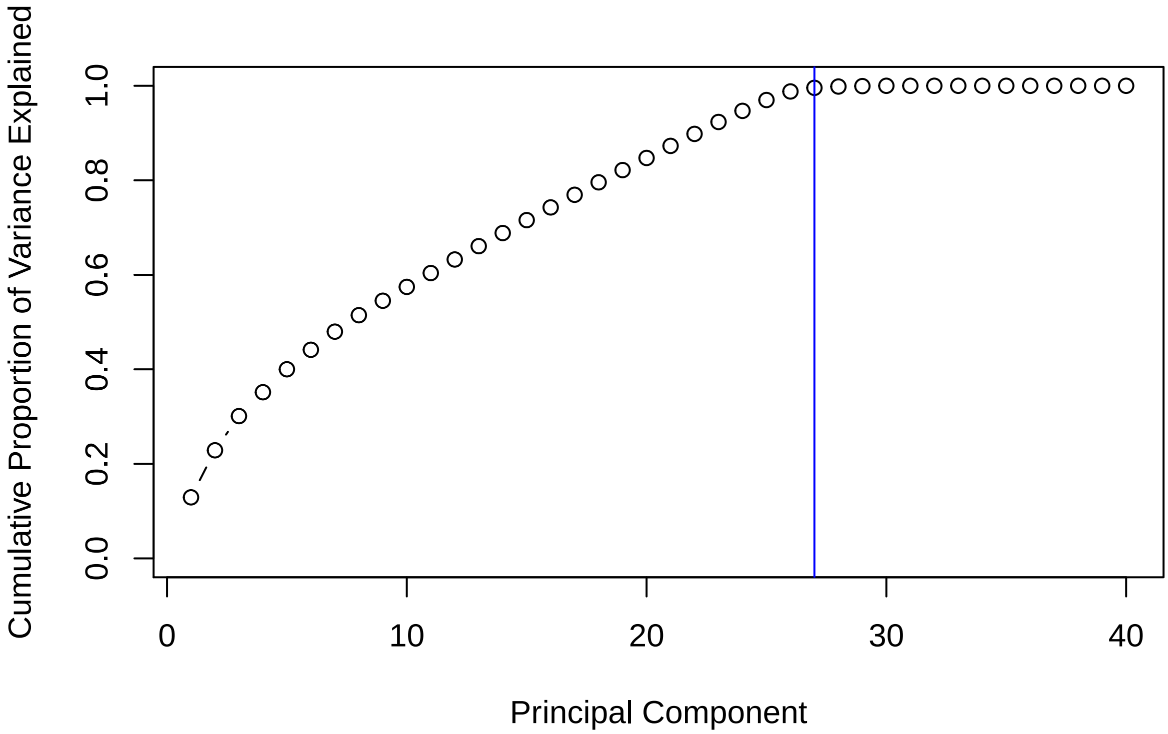
In this section, we are trying to reduce the number of our predictors by finding a normalized linear combination of the original predictors in a data set (41 predictors after hot enconding). In order to do this, we perform a Principal Component Analysis which is a generalization of the above-mentionned PCR where we extract the PCs to give us the possiblity to use them with any other model.

First, we fit our model and plot the proportion of variance explained vs. the number of first principal components chosen.

1-1.bb



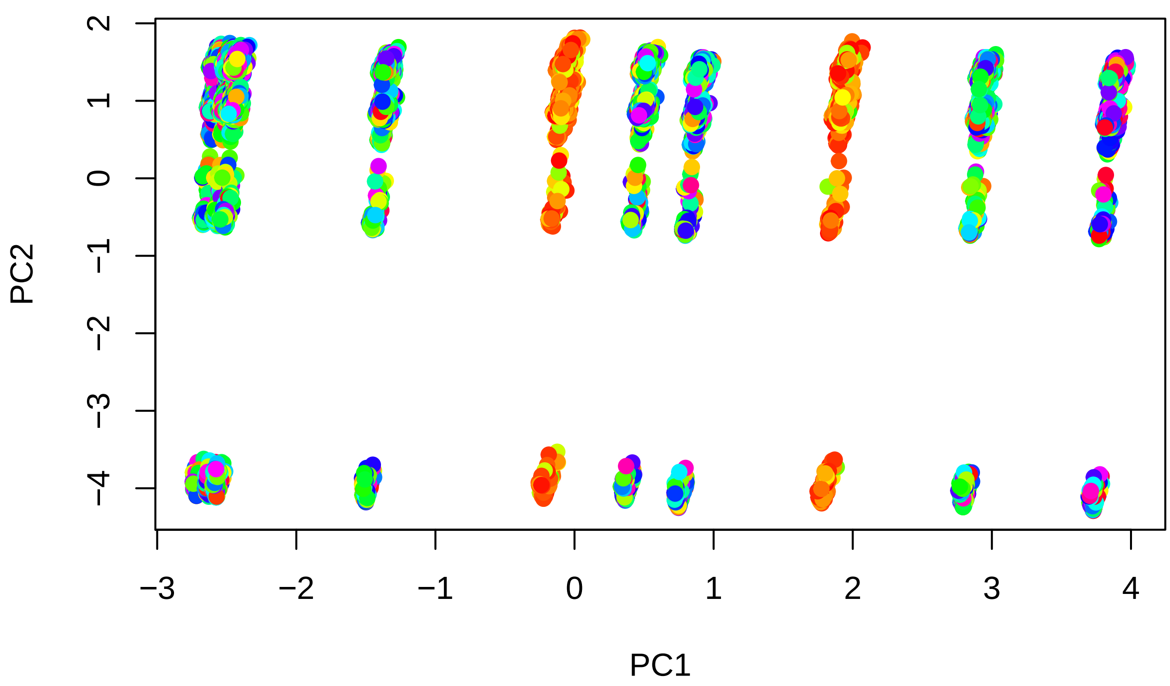
1-2.bb



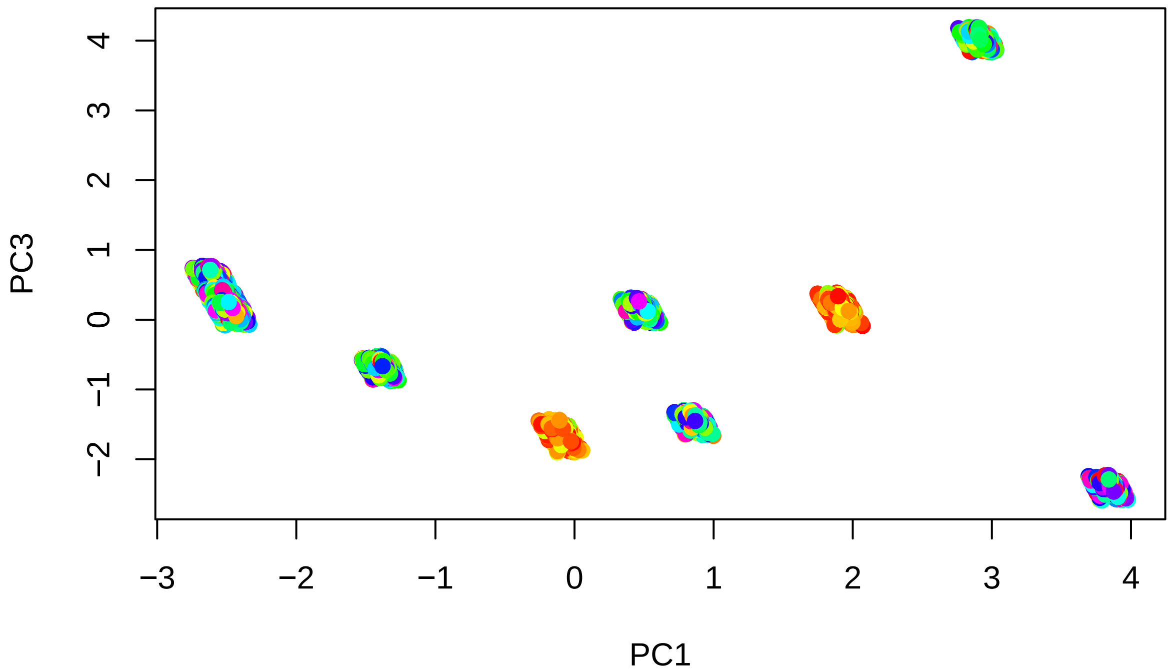

```
## [1] "PC27"

##     Standard deviation Proportion of Variance  Cumulative Proportion
##              0.5556148              0.0077200              0.9957000
```

As expected, the number of principal components that corresponds to the knee in the curve is around 27, which confirms the value previously chosen in the PCR section.

Now, we plot our data according to the 1st and 2nd Principal Components and then according to the 1st and 3rd Principal Components.



2-1.bb



2-2.bb

We can observe that, unexpectedly, the representation of the data according to the first and second PCs presents well-spreaded, however, less homogenious repartition of the data accroding to the target class `Item_Outlet_Sales`. On the other side, both of these plots represent a poor visualization of our data since they explain barely 22.8% of the variance for the first two components and even less for the 1st and 3rd combined.

Finally, we assign transform the (aside) testing set according to the same Principal Component Analysis tranformation that resulted above.

Feature selection & Dimensionality reduction

After looking at the data, we noticed that, since our dataset is a mixture of categorical (7) and numerical (7) features, we are going to need to hot encode our data in order to be able to apply a big number of resgression algorithms.

That said, the fact that some of the categorical features have up to 10 different values (10 level factors) would introduce a lot of new features after dummy encoding and we finished up with 41 features after one hot encoding (the first generated feature is always dropped). Hence, a reduction of the number of features should processed.

In the next sections, we are going to try different feature selectio and dimentionality reduction techniques and discuss them.

Forward Feature Selection

First, we start with subset selection and given the fact that we cannot perform a best subset feature selection on our data due to the large computation complexity, we try to approximate it with forward feature selection as shown below:

```
## Reordering variables and trying again:
```

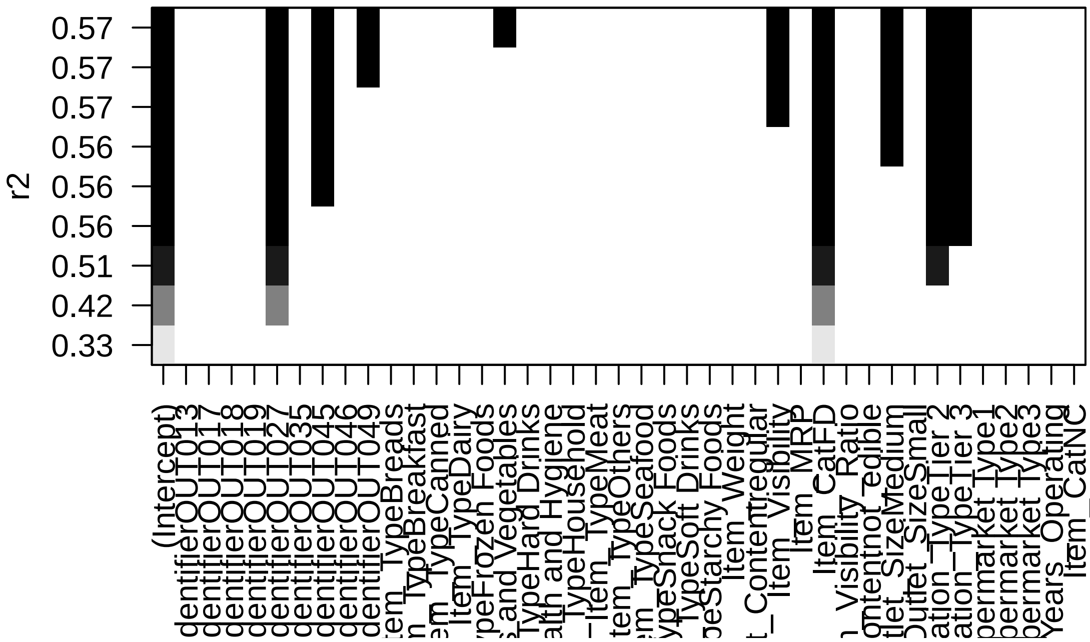


Table 1: Forward Selection

selected_features
Outlet_IdentifierOUT027
Outlet_IdentifierOUT045
Outlet_IdentifierOUT049
Item_TypeFruits and Vegetables
Item_Visibility
Item_CatFD
Outlet_SizeMedium
Outlet_Location_TypeTier 2
Outlet_Location_TypeTier 3

We can see that the forward selection outputs a 9 feature subset (8 + intercept) as an estimation of the best subset features selection with `Item_Visibility`, 3 of the hot encoded `Outlet_Identifier` columns (corresponding to 3 outlets), 2 of the `Outlet_Location` and one `Outlet_Size`. We can interpret that according to FFS, these features are more important than the other, i.e. for `Outlet_Size`, knowing if the outlet is of medium size or not matters more than knowing what exactly is the size of the outlet (small or big).

Backward Feature Selection

In this section, we try again to estimate the best subset of features following another method which is backward subset selection as shown below:

```
## Reordering variables and trying again:
```

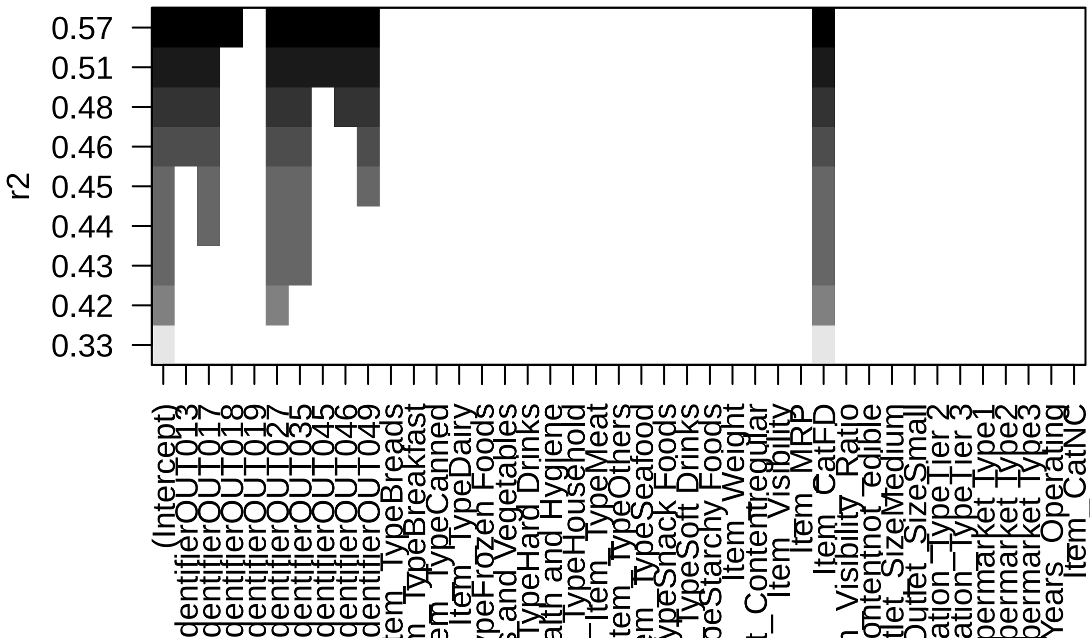


Table 2: Bakcward Selection

selected_features
Outlet_IdentifierOUT013
Outlet_IdentifierOUT017
Outlet_IdentifierOUT018
Outlet_IdentifierOUT027
Outlet_IdentifierOUT035
Outlet_IdentifierOUT045
Outlet_IdentifierOUT046
Outlet_IdentifierOUT049
Item_CatFD

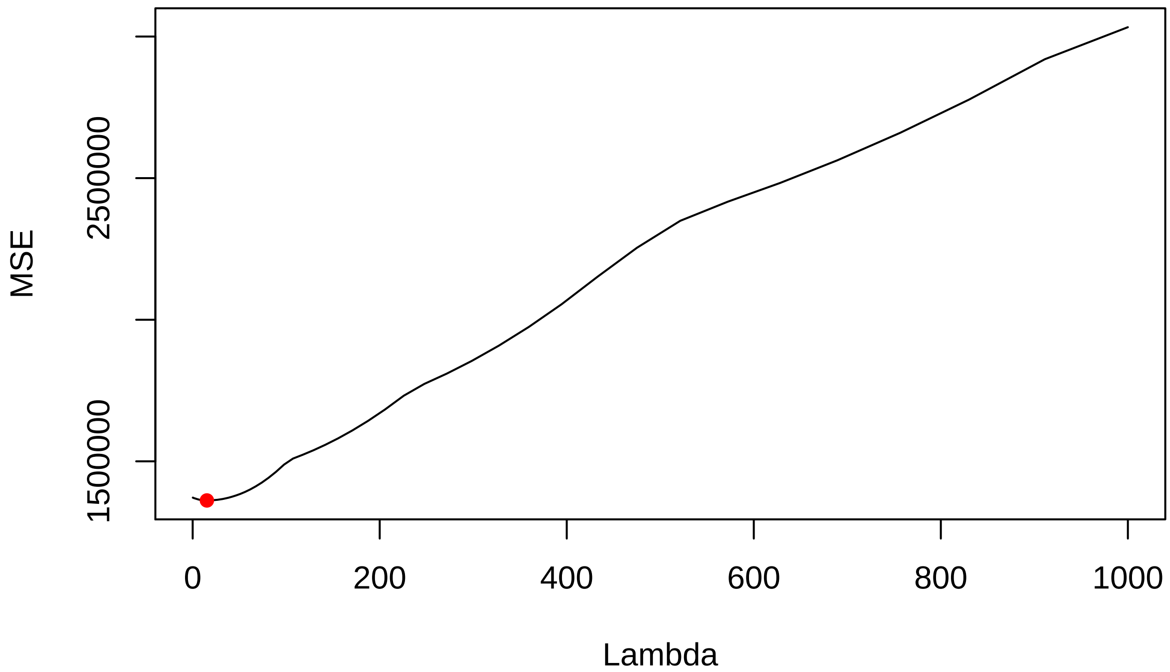
We can observe clearly through the output of BFS that this estimation gives a high importance to the `Outlet_Identifier` feature and we can see again the `Item_Cat` corresponding to the value food which means that knowing if our item is a food product or not would help the prediction of the `Item_Outlet_Sales`.

Lasso Regression

In this section, we are going to try to reduce the number of feature using Lasso. However, the tricky parts resides in choosing the λ value corresponding to the best penalty for our case, namely, a λ value that reduces the variance to prevent overfitting without increasing the bias too much. (yet another variance-bias tradeoff situation)

Hence, we execute a Lasso regression with different values of Lambda, we predict the sales for our validation (aside.test.data) set and then we calculate the error for each λ.

The plot below shows the evolution of the error with respect to different values of λ. The red point show the value corresponding to the lowest MSE.



1-1.bb

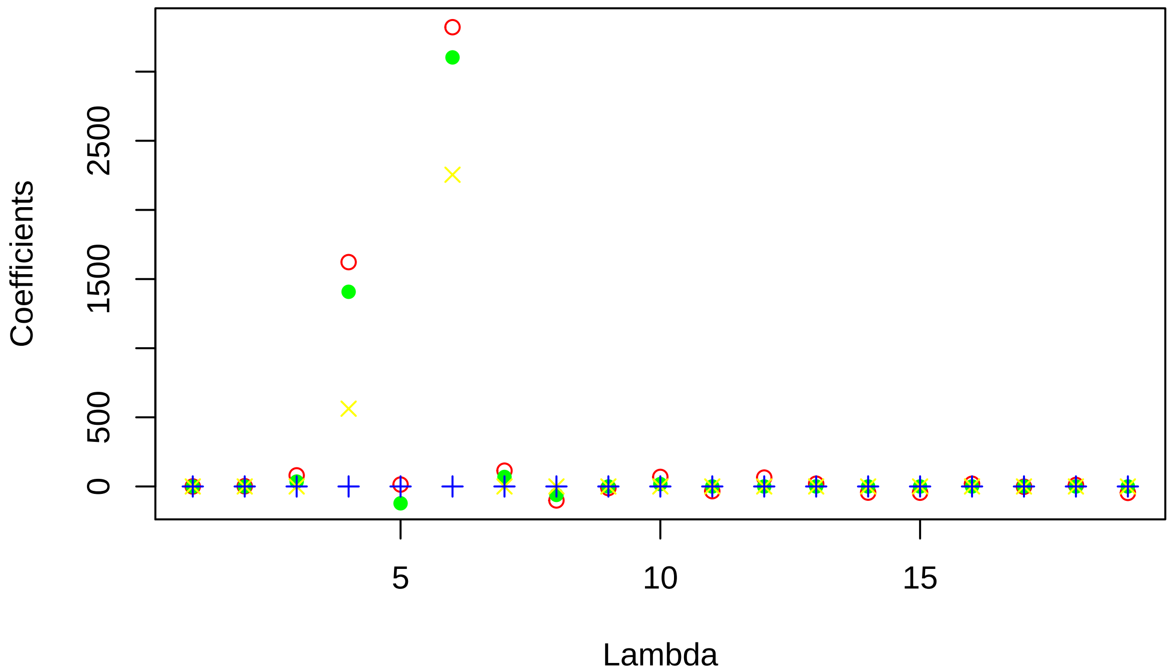
```
## integer(0)
```

	coefficients
Outlet_IdentifierOUT017	35.33963
Outlet_IdentifierOUT018	1408.00854
Outlet_IdentifierOUT019	-121.93867
Outlet_IdentifierOUT027	3102.94673
Outlet_IdentifierOUT035	68.75383
Outlet_IdentifierOUT045	-60.90939
Outlet_IdentifierOUT049	17.26060
Item_Fat_Contentregular	18.54781
Item_Visibility	-113.82002
Item_MRP	15.47142
Outlet_TypeSupermarket Type1	1754.14581
Outlet_TypeSupermarket Type2	26.36374
Outlet_TypeSupermarket Type3	41.99220

```
## Lasso Regression RMSE:  1152.322
```

As we see in the table above, all the `Outlet_Type` and `Outlet_Identifier` (almost) are kept for creating the linear regression model with the lowest RMSE. On the other side, the numerical variables `Item_MRP`and `Item_Visibility` are also kept for this regression.

Below, we can see the plot of the coefficients corresponding to the features for a given λ. The green dots correspond to the λ value (15.1991108) with the lowest error (1152.3220914).



2-1.bb

We can clearly see that for a high value of λ all the coefficients are set to zero which means that the penalty is too large for finding any feature important enough to be kept.

PC-KNN

In the plot of the principal components above we saw some gradient clustering of the `Item_Outlet_Sales`, we want to check if knn on top of PCA performs any better. We determine the best number of neighbours and the best number of principal components by cross-validation.

```
## Loading required package: FNN

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

## XGBoost Train RMSE:  1556.834
```

Tree-Based Models

Simple Trees

We implement regression trees on our training data and run a 10-fold cross validation. All predictors except for Item_Identifier for building each tree. Through this approach we obtain a training RMSE of 1276. We will keep this in mind when we run a random Forrest later on in the report.

```
## Simple tree residual RMSE:  952.7941

## Pruned tree residual RMSE:  1115.177

## Simple Tree RMSE:  1276.544

## Pruned Tree RMSE:  1232.276
```

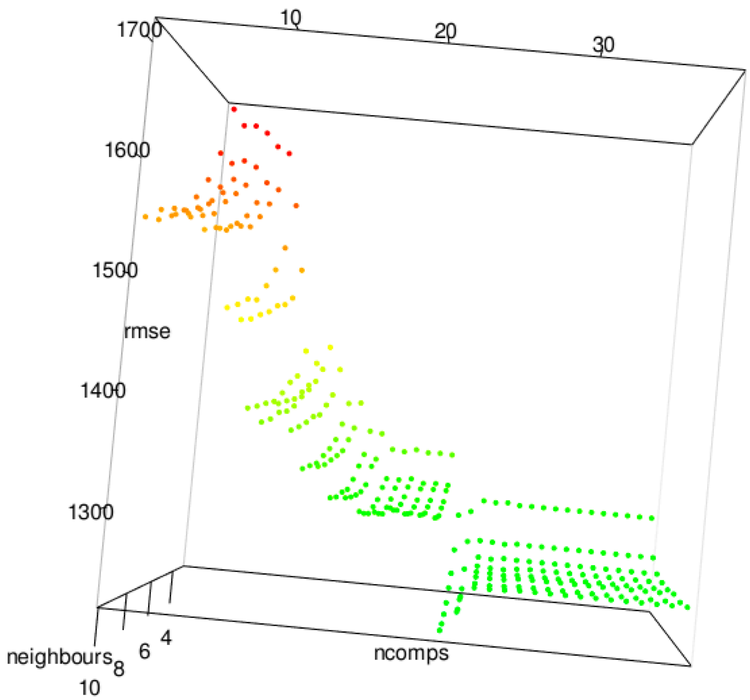
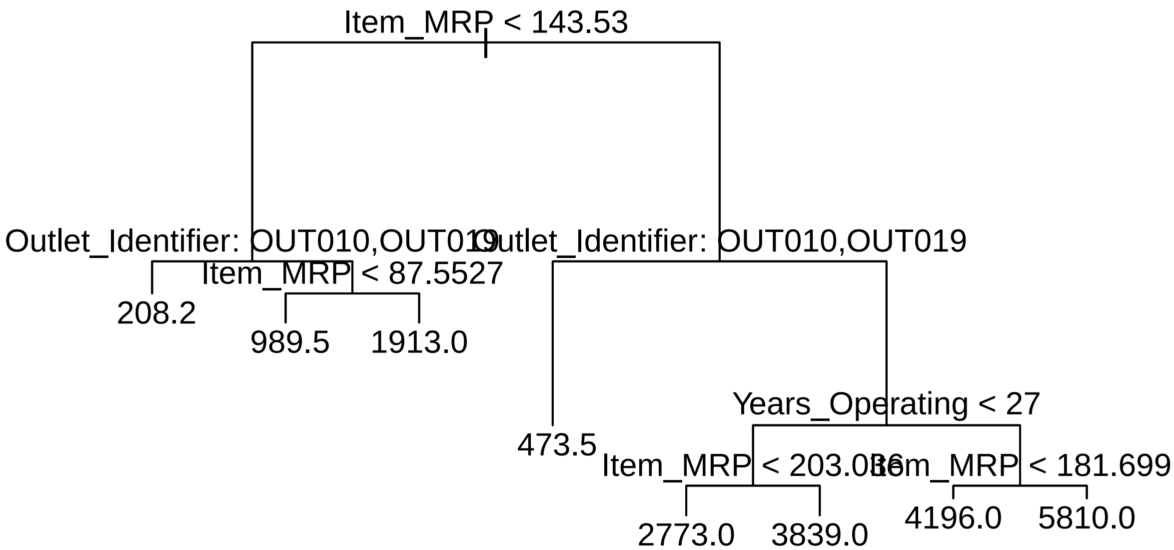


Figure 1: KNN rmse plot

We can see above that pruning the tree actually reduces the RMSE on the validation set (which is an estimation of the real RMSE), however, the complete tree outperforms the pruned tree on the training set. This could simply be explained by the fact that the big tree overfits the data.

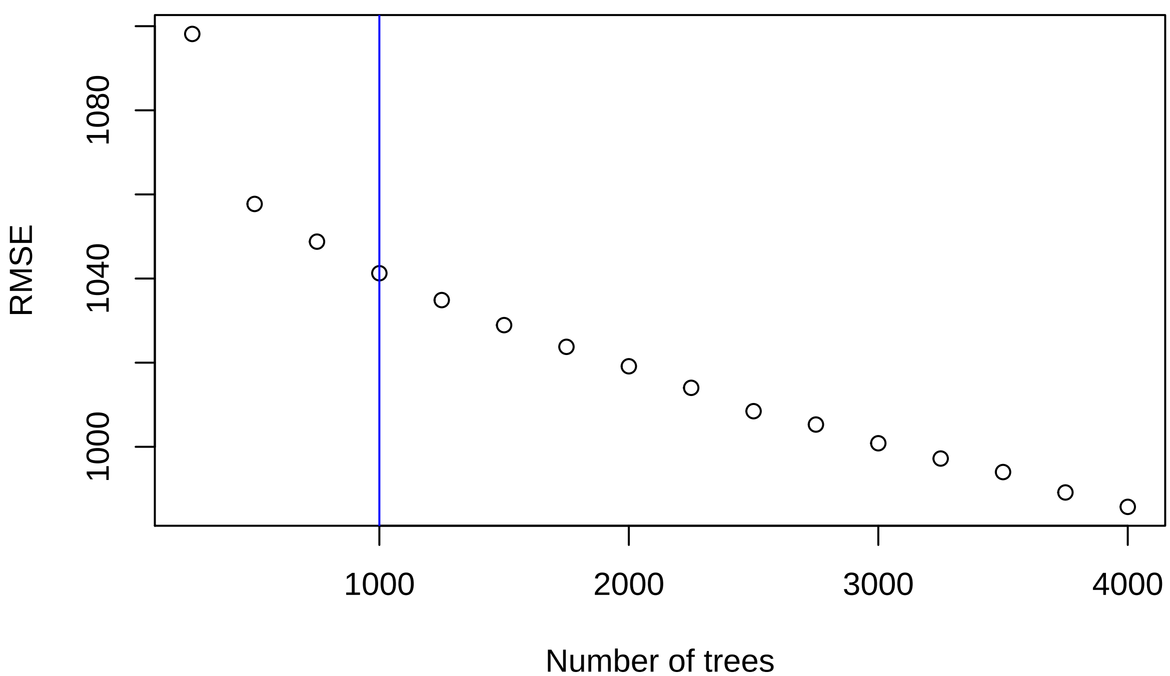
On the other side, pruning a tree doesn't only prevent overfitting but helps also having clear visualizations of the tree as shown below:



Ensembles

Boosting

```
## Loaded gbm 2.1.4
```




```
## integer(0)
```

```
## Boosting RMSE:  1073.609
```

XGBoost

eXtreme Gradient Boosting

Xgboost is a parallelized boosting algorithm that implements dropout regularization (dropping trees that tend to overfit data) to prevent overfitting. Boosting techniques suffer from over fitting since at each step they try to minimize the error from the previous step thus tend to overly adapt to the data.

We do cross validation for searching the best parameters over 100 iterations. Evaluation criterion is `rmse`.

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

Now we plot the `rmse` against the number of trees to see the knee in the curve. We don't want to overfit. From the plot below that 15 is the number of trees after which the error does not drop significantly.

```
## best_param:
```

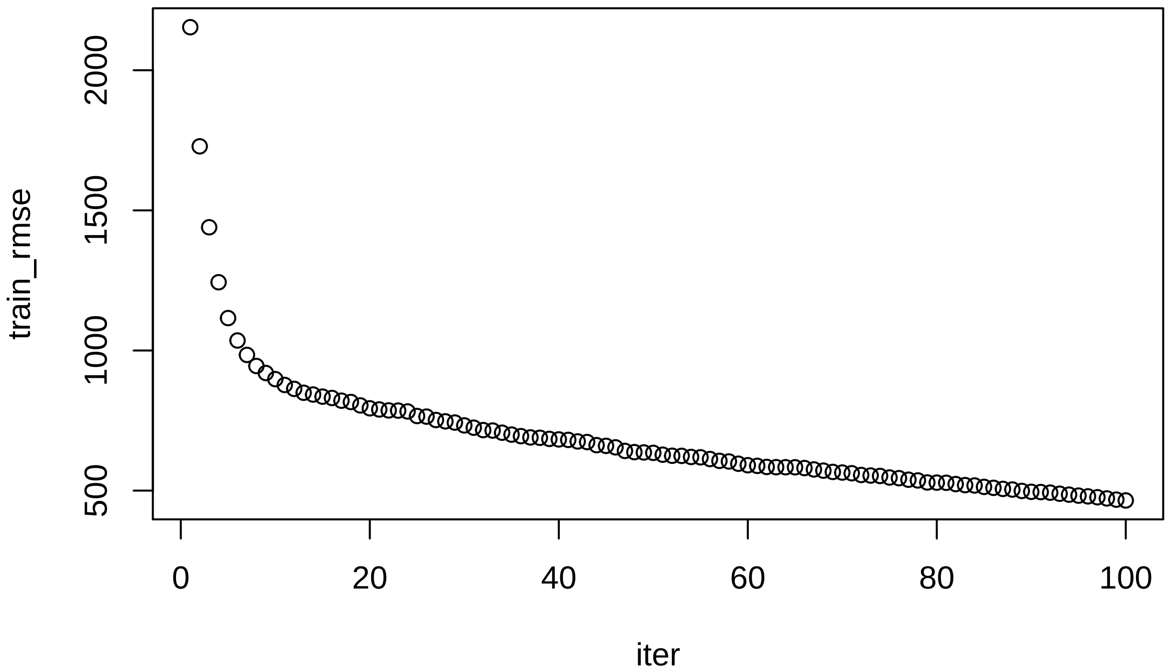
```
## $objective
## [1] "reg:linear"
##
## $eval_metric
## [1] "rmse"
##
## $max_depth
## [1] 8
##
## $eta
## [1] 0.2657478
```

```
## best_rmse:
```

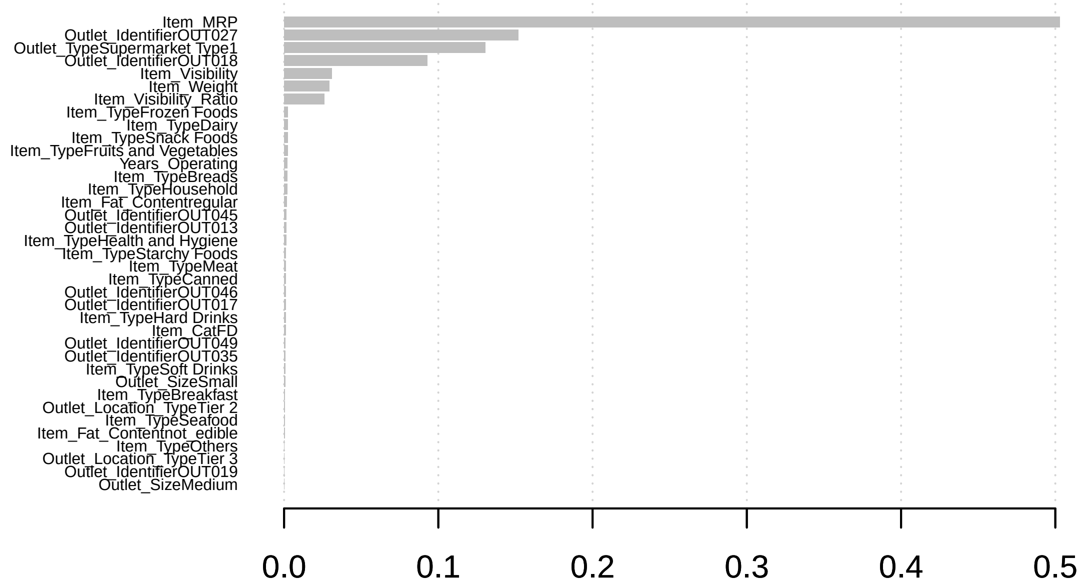
```
## [1] 1124.704
```

```
## best_rounds:
```

```
## 100
```




```
##    iter train_rmse
## 1:   15   835.3906
```



The plot above explains the variable importance where we can clearly see that `MRP` is the most significant feature.

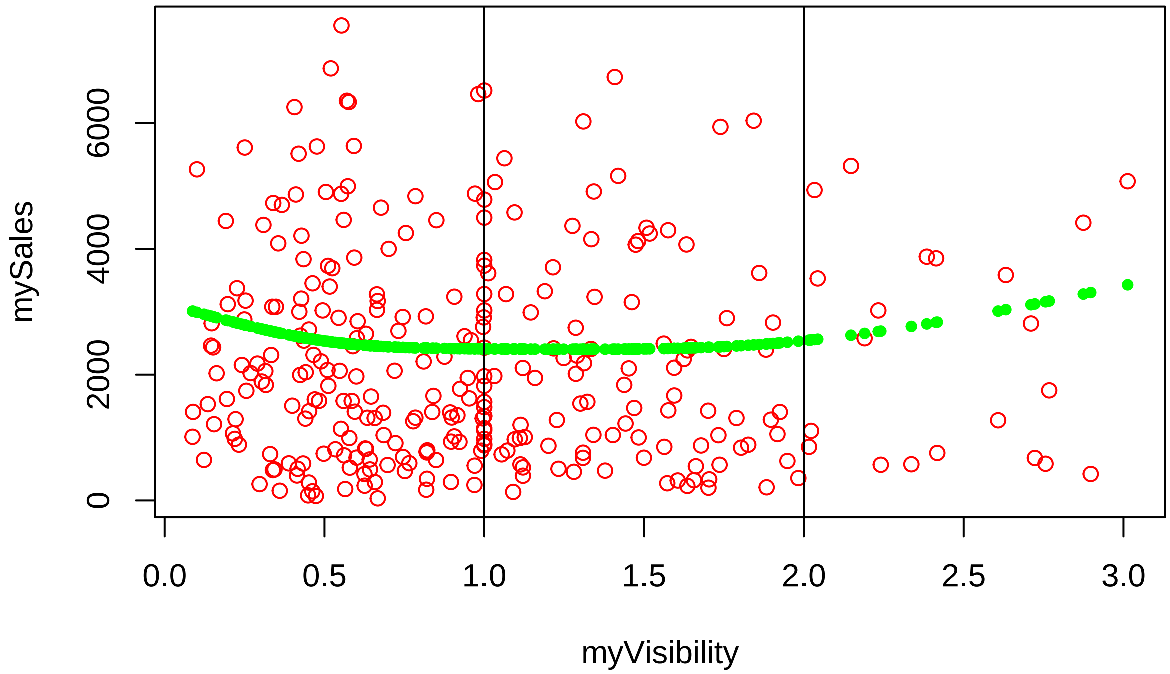
Bagging (Random forest)

In this section we now implement a random Forrest on the Training Data and run a 10-fold cross validation. Looking at the error plots vs. number of trees we decided each Forrest having 50 trees is sufficient in order to get a approximation of our Training RMSE. Additionally, by default within a given Forrest each tree is built on a random sample of 2/3 with 1/3 of the predictors. Through this approach we obtain a training RMSE of about 1185 which is a little better than the Tree RMSE of 1276.

Natural Splines

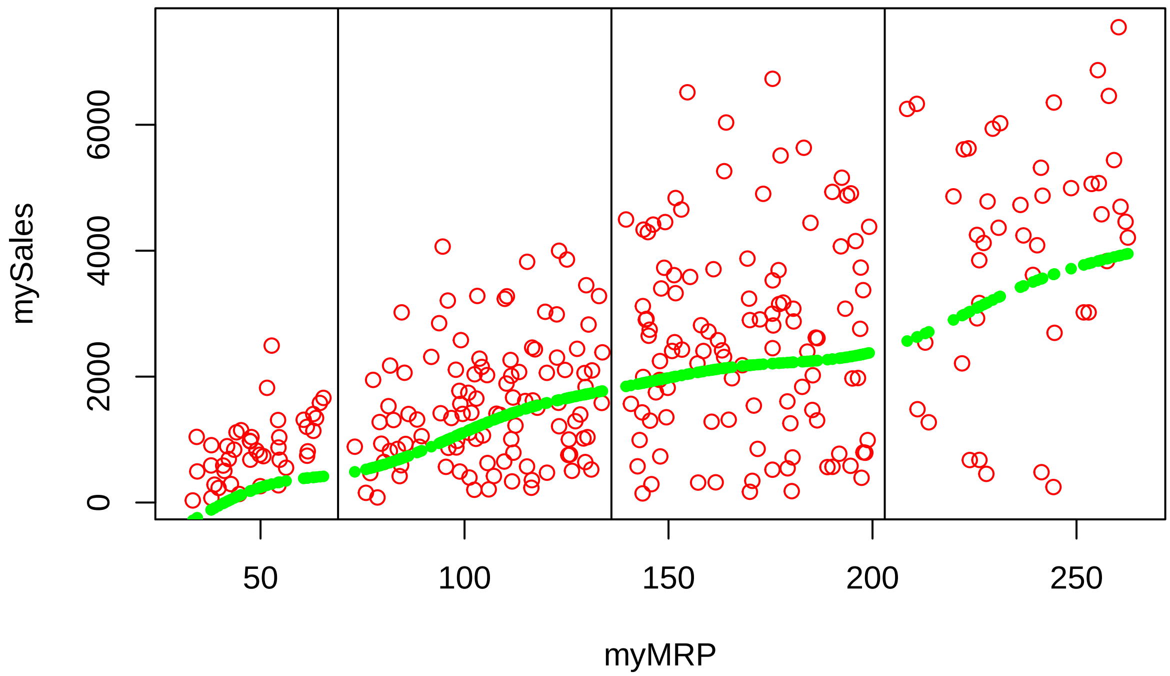
As we observed above in the `Item_Outlet_Sales` vs. `Item_MRP` plot that the `Item_MRP` presents three different seperation between the data. These separations could be interpreted as knots where the underlying function could have changed. On the other side, we know that `Item_MRP` is a predictor of high importance according to the previous experiments. In order to investigate this further, we tried to fit our model with a natural spline with three knots corresponding to the above-mentionned values.

We noticed also during the experiments that a natural spline could be fitted (better than other predictors but still poor) to `Item_Visibility_Ratio`.



1-1.bb

```
## integer(0)
```



1-2.bb

```
## integer(0)
```

We can observe that the model is poorly fitted for `Item_Visibility_Ratio`, however, it presents a slightly better fit for `Item_MRP`.

After multiple experiments, the best fitting spline to the sample of data that we picked corresponds to the predicted value `-3` times the standard error as shown above. Now we fit our model on the whole training data and calculate the RMSE.

```
## Natural Spline RMSE(Item_MRP:  1397.221
```

We can see that the model above is not performing well since its RMSE is around 1397.588 which is higher than the previous models.

We can conclude that this is due to a poor fit of the underlying function since it's based on only one predictor.

Evaluation

From the table, we observe boosting trees perform better in our case.

Conclusion

Overall, we cleaned the data and explored it the using ggplot visualizations. Once we cleaned the data we used Forward,Backward,Lasso, and PCA for feature selection. After some basic feature engineering, we implemented the models above. After accessing models the we decided that the Boosting Trees model was the superior model. We went on to submit this model and got a RMSE of 1152.63.

Performance of Models

Models	(Cross-validation) RMSE
Simple Trees	1277
XGBoost	1128
Boosting Trees	1074
Random Forest	1188
Lasso Regression	1152
PCR	1127
Splines	1397
PC-KNN	1523

Figure 2: Model Performances