


**IF YOU TORTURE THE DATA LONG ENOUGH,
IT WILL CONFESS TO ANYTHING.**

- RONALD COASE -

WHY R ?

- The style of coding is quite easy.
- It's open source. No need to pay any subscription charges.
- Availability of instant access to over 10,000 packages customized for various computation tasks.
- The community support is overwhelming. There are numerous forums to help you out.
- Get high performance computing experience (require packages).
- One of highly sought after skill by analytics and data science companies.

Analysis Tool	Similar Superhero	Super Powers in Common
<p>R</p> 	<p>Batman</p> 	<ul style="list-style-type: none">• Detective Work• Intelligence• Cunning• Usage of Tools• More Brain than Muscles
<p>Python</p> 	<p>Superman</p> 	<ul style="list-style-type: none">• Muscle Power• Super Strength• Elegance• Wide Range• More Muscles than Brain

R Analytics

Module I: The Basics

- Data Types, Structures & Variable Types in R
- Control Structures (Functions) in R
- Useful R Packages

Module II: Exploratory Analysis

- Data Visualization
- Data Manipulation
- Predictive Modeling using Machine Learning (Linear Regression)

I. Data types

- **character:** "a", "swc"
- **numeric:** 2, 15.5
- **integer:** 2L (the L tells R to store this as an integer)
- **logical:** TRUE, FALSE
- **complex:** 1+4i (complex numbers with real and imaginary parts)

II. Data structures - Vectors

- Vector is a basic data structure in R that contains element of similar type. A vector is what is called an array in all other programming languages except R.

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
```

```
b <- c("one","two","three") # character vector
```

```
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

- Refer to elements of a vector using subscripts. `a[c(2,4)]` # 2nd and 4th elements.
- In R using the function, `typeof()` one can check the data type of vector.
- One more significant property of R vector is its length. The function `length()` determines the number of elements in the vector.

Matrices

A **matrix** is a two-dimensional vector (fixed size, all cell types the same).

```
# generates 5 x 4 numeric matrix  
y<-matrix(1:20, nrow=5,ncol=4)
```

```
# another example (byrow =TRUE implies storing data row-wise)  
cells <- c(1,26,24,68)  
rnames <- c("R1", "R2")  
cnames <- c("C1", "C2")  
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,  
  dimnames=list(rnames, cnames))
```

Identify rows, columns or elements using subscripts.

```
y[,4] # 4th column of matrix  
y[3,] # 3rd row of matrix  
y[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

Lists

An ordered collection of objects (components), allowing you to gather a variety of (possibly unrelated) objects under one name. A list can hold items of different types and the list size can be increased on the fly.

example of a list with 4 components -

a string, a numeric vector, a matrix, and a scalar

```
w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
```

There are a variety of ways to identify the elements of a list:

```
w[1:2] # columns 1,2 of list
```

```
w[c("name","mynumbers")] # columns name and mynumbers from list
```

```
w$age # variable age in the list
```


Data frames

It is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.). A **data frame** is called a table in most languages. Each column holds the same type.

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") # variable names
print (mydata)
```

There are a variety of ways to identify the elements of a data frame:

```
mydata[1:2] # columns 1,2 of data frame
mydata[c("ID","Color")] # columns ID and Age from data frame
mydata$ID # variable ID in the data frame
```

Arrays

An **array** is a vector with one or more dimensions. So, an array with one dimension is (almost) the same as a vector. An array with two dimensions is (almost) the same as a matrix. An array with three or more dimensions is an n-dimensional array.

- `arr = array(0.0, 3) # [0.0 0.0 0.0] print(arr)`
- `arr = array(0.0, c(2,3)) # 2x3 matrix print(arr)`
- `arr = array(0.0, c(2,5,4)) # 2x5x4 n-array # print(arr) # 40 values displayed`

Useful functions to handle data structures

`length(mydata)` # number of elements or components

`str(mydata)` # structure of an object

`class(mydata)` # class or type of an object

`names(mydata)` # names

`object` # prints the object

`newobject <- edit(mydata)` # edit copy and save as newobject

`fix(mydata)` # edit in place

`rbind()` # combine data structures

`rm(mydata)` # delete an object

III. Categorical & Continuous variables

- In a dataset, we can distinguish two types of variables: **categorical** and **continuous**.
- In a categorical variable, the value is limited and usually based on a particular finite group. For example, a categorical variable can be gender, education level, etc.
- A continuous variable, however, can take any values, from integer to decimal. For example, we can have the revenue, price of a share, etc.
- Continuous class variables are the default value in R. They are stored as numeric or integer. We can see it from the dataset below. mtcars is a built-in dataset. It gathers information on different types of car. We can import it by using mtcars and check the class of the variable mpg (miles per gallon)- `class(mtcars$mpg)`. It returns a numeric value, indicating a continuous variable.

IV. Factors

- R stores categorical variables into a factor. Let's check the code below to convert a character variable into a factor variable.
- `factor(x = character(), levels, labels = levels, ordered = is.ordered(x))`
- **x**: A vector of data. Need to be a string or integer, not decimal.
- **Levels**: A vector of possible values taken by x. This argument is optional. The default value is the unique list of items of the vector x.
- **Labels**: Add a label to the x data. For example, 1 can take the label `male` while 0, the label `female`.
- **ordered**: Determine if the levels should be ordered.

Nominal categorical variable

A categorical variable has several values but the order does not matter. For instance, male or female categorical variable do not have ordering.

```
# Create gender vector
```

```
gender_vector=c("Male", "Female", "Female", "Male", "Male")
```

```
class(gender_vector)
```

```
# Convert gender_vector to a factor
```

```
factor_gender_vector = factor(gender_vector)
```

```
class(factor_gender_vector)
```

```
#factor_gender_vector
```

```
[1] Male  Female Female Male  Male
```

```
Levels: Female Male
```

Ordinal categorical variable

Ordinal categorical variables have a natural ordering. We can specify the order, from the lowest to the highest with `order = TRUE`.

```
# Create Ordinal categorical vector
```

```
day_vector <- c('evening', 'morning', 'afternoon', 'midday', 'midnight', 'evening')
```

```
# Convert `day_vector` to a factor with ordered level
```

```
factor_day <- factor(day_vector, order = TRUE, levels = c('morning', 'midday', 'afternoon', 'evening',  
'midnight'))
```

```
# Print the new variable
```

```
factor_day
```

```
# Count the number of occurrences of each level
```

```
summary(factor_day)
```

V. Control structures in R- if,else

if,else – This structure is used to test a condition.

#initialize a variable

```
N <- 10
```

#check if this variable * 5 is > 40

```
if (N * 5 > 40){
```

```
    print("This is easy!")
```

```
} else {
```

```
    print ("It's not easy!")
```

```
}
```

```
# ifelse(N*5>40,"this is easy", "this is not easy")
```


for

This structure is used when a loop is to be executed fixed number of times. It is commonly used for iterating over the elements of an object (list, vector).

```
#initialize a vector
```

```
y <- c(99,45,34,65,76,23)
```

```
#print the first 4 numbers of this vector
```

```
for(i in 1:4){  
  print (y[i])}
```

Exercise: Find sum of all elements of y

Find the length of the data structure

len=length(y)

Make a new variable to store the sum of all values

sum=0

Use the for loop to iteratively add add elements to y to sum

for (i in 1:len) {sum= sum+y[i]}

Print the variable sum

sum

while

It begins by testing a condition, and executes only if the condition is found to be true. Once the loop is executed, the condition is tested again.

```
#initialize a condition
```

```
i=1
```

```
sum=0
```

```
#check if age is less than 17
```

```
while (i<=len) {
```

```
    sum=sum+y[i]; i=i+1;} #Once the loop is executed, this code breaks the loop
```

```
#print sum value
```

```
sum
```

VI. Useful R packages

- **Importing Data:** R offers wide range of packages for importing data available in any format such as .txt, .csv, .json, .sql etc. To import large files of data quickly, it is advisable to install and use *data.table*, *readr*, *RMySQL*, *sqldf*, *jsonlite*.
- **Data Visualization:** R has in built plotting commands as well. They are good to create simple graphs. But, becomes complex when it comes to creating advanced graphics. Hence, you should install *ggplot2*.
- **Data Manipulation:** R has a fantastic collection of packages for data manipulation. These packages allows you to do basic & advanced computations quickly. These packages are *dplyr*, *plyr*, *tidyr*, *lubridate*, *stringr*.
- **Modeling / Machine Learning:** For modeling, *caret* package in R is powerful enough to cater to every need for creating machine learning model. However, you can install packages algorithms wise such as *randomForest*, *rpart*, *gbm* etc.

Steps of data exploration and preparation

- Variable Identification
- Data Visualization- Univariate Analysis/ Bi-variate Analysis
- Missing values treatment
- Feature Engineering
- Label Encoding / One Hot Encoding
- Outlier Handling

Exploratory analysis using a sample dataset

- The market research team at SuperMart have collected sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store.
- Using this model, SuperMart will try to understand the properties of products and stores which play a key role in increasing sales.
- Please note that the data may have missing values as some stores might not report all the data due to technical glitches. Hence, it will be required to treat them accordingly.

Reading the data

```
test = read.csv("C:\\Users\\Username\\Desktop\\191207\\Test_Super.txt", header = T, stringsAsFactors = F)
train = read.csv("C:\\users\\Username\\Desktop\\191207\\Train_Super.txt", header = T, stringsAsFactors = F)
```

#check dimesions (number of row & columns) in data set

dim(train) # 8523 rows and 12 columns in train data set

dim(test) # 5681 rows and 11 columns in data set, Test data should always have one column less

#check the variables and their types in train

str(train)

Variable identification

- First, identify **Predictor** (Input) and **Target** (Output) variables. Next, identify the data type and category of the variables.
- **Response variable (or Dependent variable)**: In a data set, the response variable (y) is one on which we make predictions. In this case, we'll predict 'Item_Outlet_Sales'.
- **Predictor variable (or Independent variable)**: In a data set, predictor variables (X_i) are those using which the prediction is made on response variable.
- **Train Data**: The predictive model is always built on train data set. An intuitive way to identify the train data is, that it always has the 'response variable' included.
- **Test Data**: Once the model is built, it's accuracy is 'tested' on test data. This data always contains less number of observations than train data set. Also, it does not include 'response variable'.

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility
Length:8523	Min. : 4.555	Length:8523	Min. :0.00000
Class :character	1st Qu.: 8.774	Class :character	1st Qu.:0.02699
Mode :character	Median :12.600	Mode :character	Median :0.05393
	Mean :12.858		Mean :0.06613
	3rd Qu.:16.850		3rd Qu.:0.09459
	Max. :21.350		Max. :0.32839
	NA's :1463		

Item_Type	Item_MRP	Outlet_Identifier
Length:8523	Min. : 31.29	Length:8523
Class :character	1st Qu.: 93.83	Class :character
Mode :character	Median :143.01	Mode :character
	Mean :140.99	
	3rd Qu.:185.64	
	Max. :266.89	

Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type
Min. :1985	Length:8523	Length:8523
1st Qu.:1987	Class :character	Class :character
Median :1999	Mode :character	Mode :character
Mean :1998		
3rd Qu.:2004		
Max. :2009		

Outlet_Type	Item_Outlet_Sales
Length:8523	Min. : 33.29
Class :character	1st Qu.: 834.25
Mode :character	Median : 1794.33
	Mean : 2181.29
	3rd Qu.: 3101.30
	Max. :13086.97

Variable	Description
Item_Identifier	Unique product ID
Item_Weight	Weight of product
Item_Fat_Content	Whether the product is low fat or not
Item_Visibility	The % of total display area of all products in a store allocated to the particular product
Item_Type	The category to which the product belongs
Item_MRP	Maximum Retail Price (list price) of the product
Outlet_Identifier	Unique store ID
Outlet_Establishment_Year	The year in which store was established
Outlet_Size	The size of the store in terms of ground area covered
Outlet_Location_Type	The type of city in which the store is located
Outlet_Type	Whether the outlet is just a grocery store or some sort of supermarket
Item_Outlet_Sales	Sales of the product in the particular store. This is the outcome variable to be predicted.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

Graphical representation of variables

Univariate Analysis v Bivariate Analysis

```
summary(train)
```

```
install.packages("ggplot2")
```

```
library(ggplot2)
```

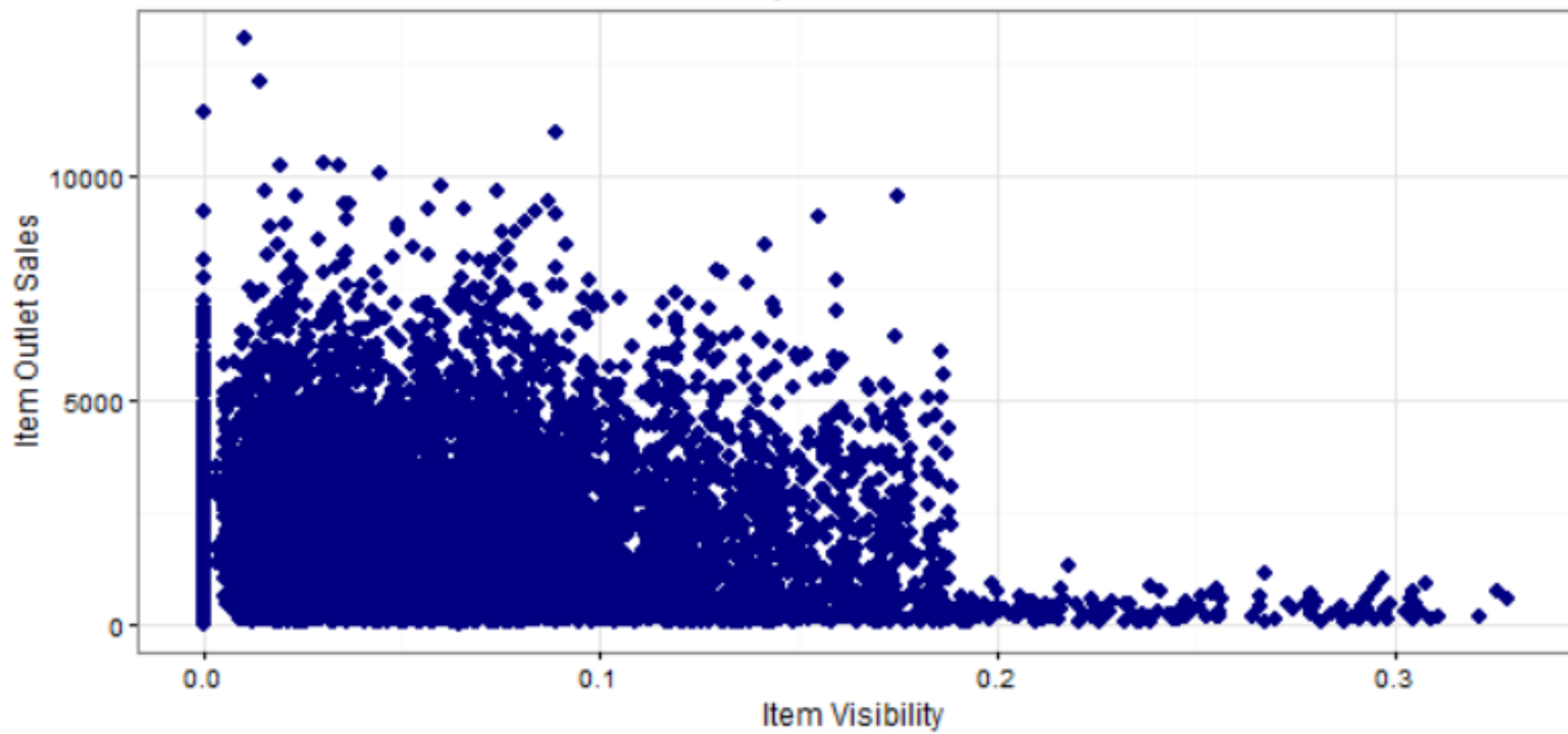
```
ggplot(train, aes(Outlet_Establishment_Year)) +  
  geom_histogram(binwidth = 1.0, color = "black", fill = "grey")
```

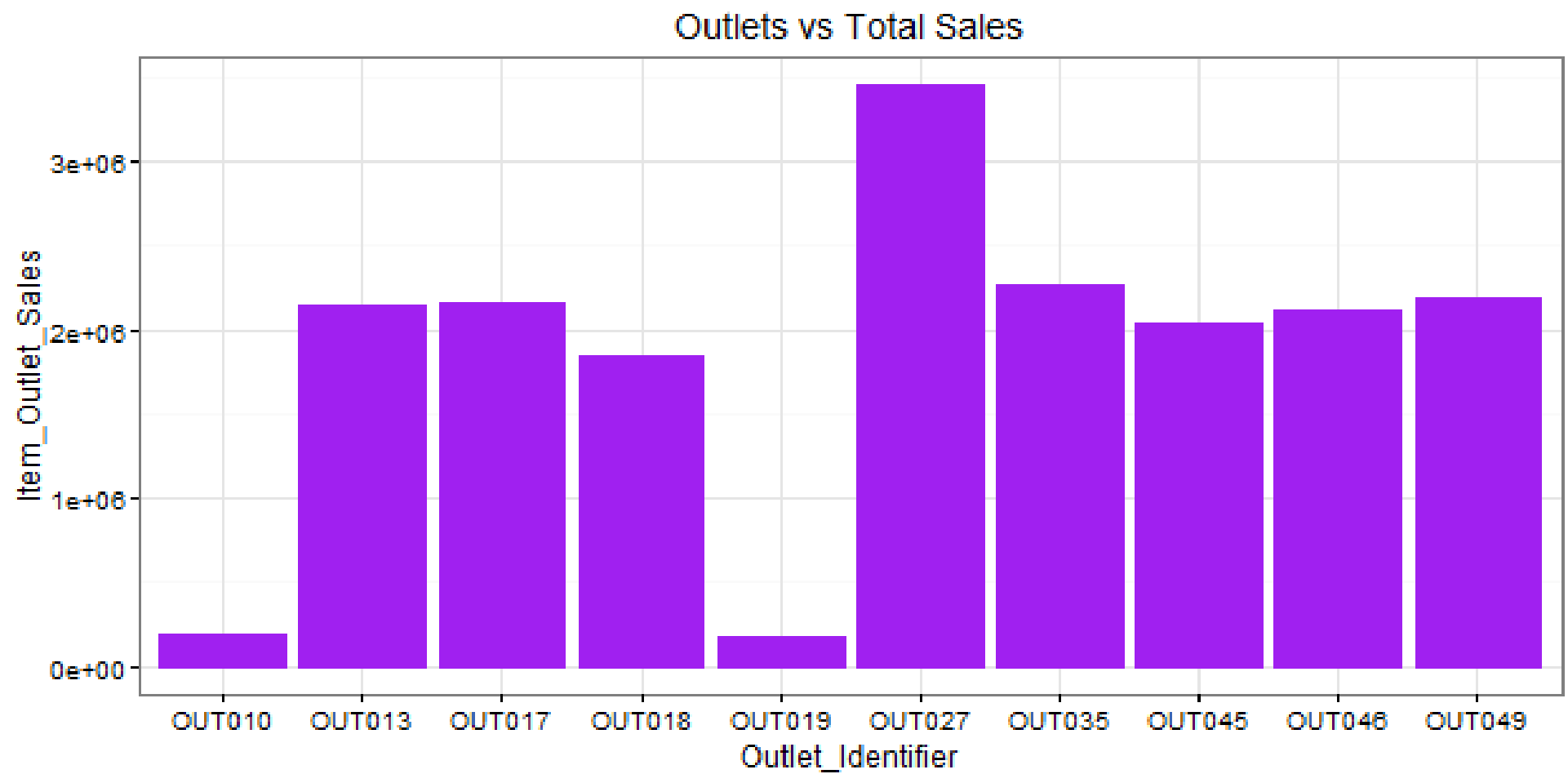


```
ggplot(train, aes(x= Item_Visibility, y = Item_Outlet_Sales)) + geom_point(size = 2.5, color="navy")
```

We can see that majority of sales has been obtained from products having visibility less than 0.2. This suggests that `item_visibility < 0.2` must be an important factor in determining sales. Let's plot few more interesting graphs and explore such hidden stories.

Item Visibility vs Item Outlet Sales

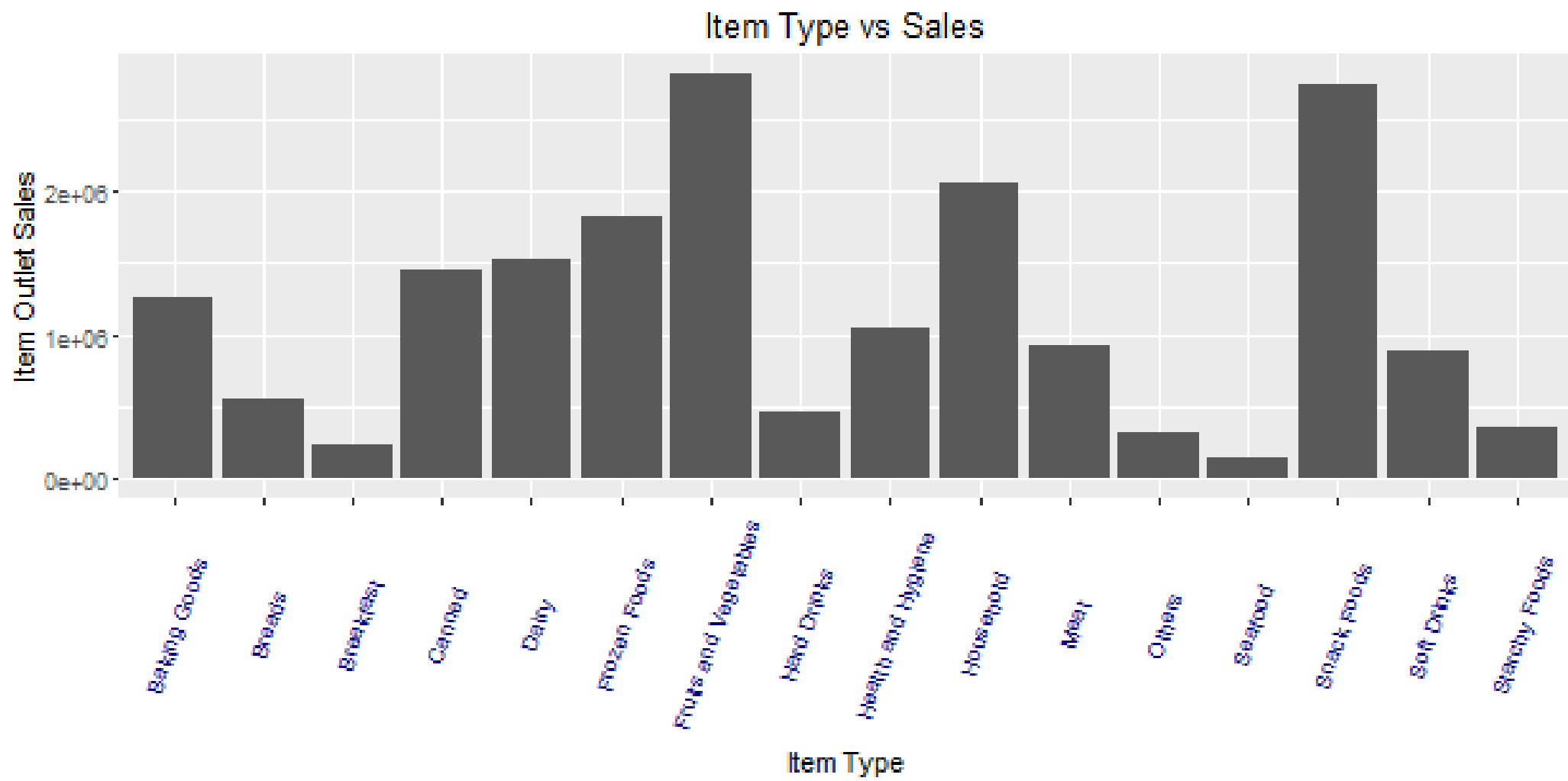




```
ggplot(train, aes(Outlet_Identifier, Item_Outlet_Sales)) + geom_bar(stat = "identity",  
color = "purple") + ggtitle("Outlets vs Total Sales")
```

#By default, **geom_bar** uses **stat="bin"** . This makes the height of each bar equal to the number of cases in each group, and it is incompatible with mapping values to the y aesthetic. If you want the heights of the bars to represent values in the data, use **stat="identity"** and map a value to the y aesthetic.

Here, we infer that OUT027 has contributed to majority of sales followed by OUT35. OUT10 and OUT19 have probably the least footfall, thereby contributing to the least outlet sales.



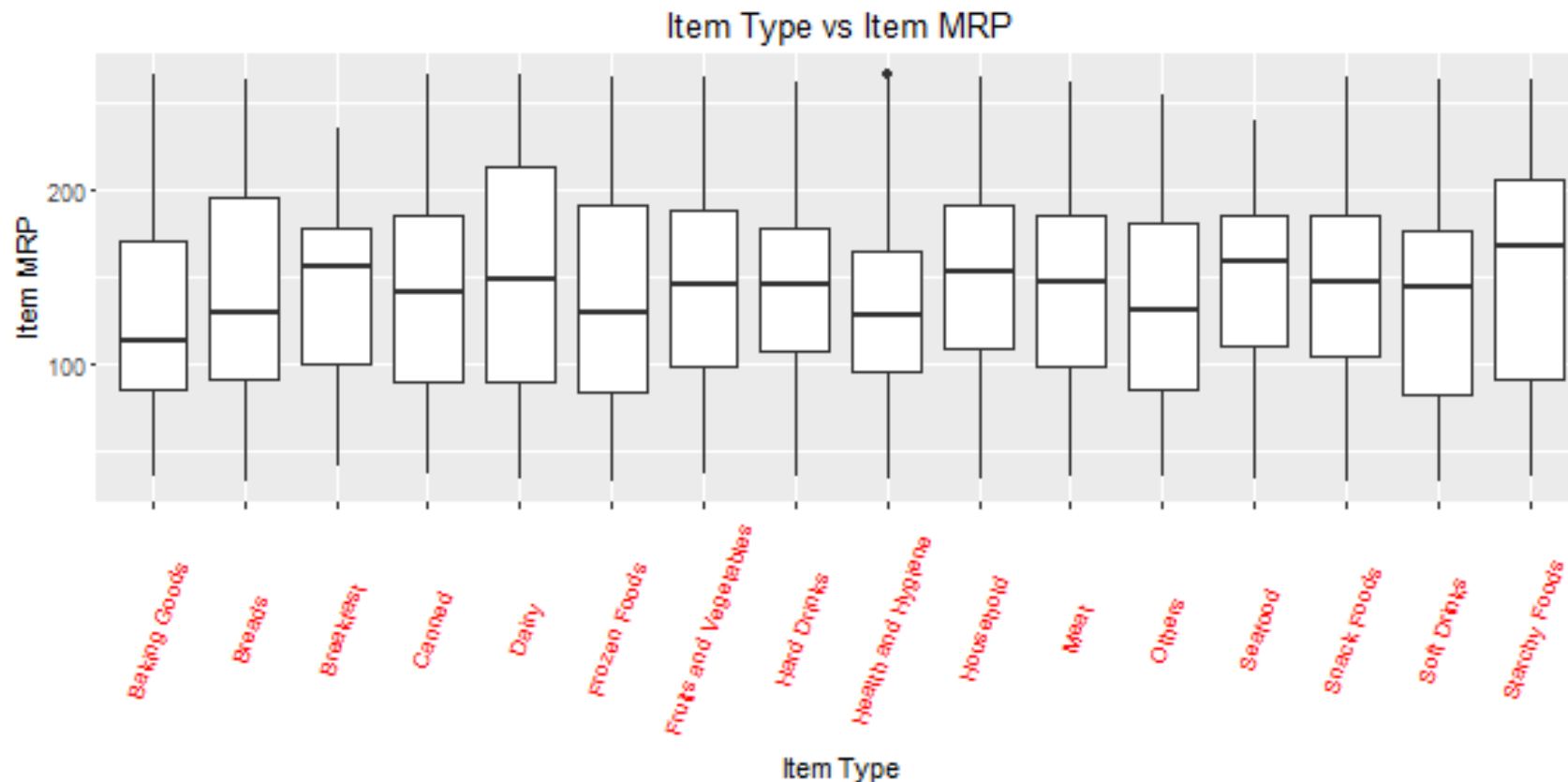
```
ggplot(train, aes(Item_Type, Item_Outlet_Sales)) + geom_bar( stat =  
"identity") +ggtitle("Item Type vs Sales")
```

```
ggplot(train, aes(Item_Type, Item_Outlet_Sales)) + geom_bar( stat =  
"identity") +theme(axis.text.x = element_text(angle = 70, vjust = 0.5, color =  
"navy")) + xlab("Item Type") + ylab("Item Outlet Sales")+ggtitle("Item Type vs  
Sales")
```

From this graph, we can infer that Fruits and Vegetables contribute to the highest amount of outlet sales followed by snack foods and household products. This information can also be represented using a box plot chart. The benefit of using a box plot is, you get to see the outlier and mean deviation of corresponding levels of a variable (shown below).

Box plot

Box Plot shows 5 statistically significant numbers- the minimum, the 25th percentile, the median, the 75th percentile and the maximum. It is thus useful for visualizing the spread of the data is and deriving inferences accordingly.



```
ggplot(train, aes(Item_Type, Item_MRP)) +geom_boxplot()  
+ggtitle("Box Plot") + theme(axis.text.x = element_text(angle = 70, vjust  
= 0.5, color = "red")) + xlab("Item Type") + ylab("Item MRP") +  
ggtitle("Item Type vs Item MRP")
```

The black point you see, is an outlier. The mid line you see in the box, is the mean value of each item type.

Now, we have an idea of the variables and their importance on response variable.

Missing Values

First check if this data has missing values. This can be done by using:

```
table(is.na(train))
```

```
FALSE TRUE
```

```
100813 1463
```

In train data set, we have 1463 missing values. Let's check the variables in which these values are missing. It's important to find and locate these missing values. Missing data in the training data set can reduce the power/fit of a model or can lead to a biased model because we have not analyzed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

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

Item_Identifier	Item_Weight	Item_Fat_Content
0	1463	0
Item_visibility	Item_Type	Item_MRP
0	0	0
outlet_Identifier	outlet_Establishment_Year	outlet_Size
0	0	0
outlet_Location_Type	outlet_Type	Item_Outlet_Sales
0	0	0

Treating missing values

- Reasons for occurrence of missing values. They may occur at two stages:
 - ✓ **Data Extraction:** It is possible that there are problems with extraction process. In such cases, we should double-check for correct data with data guardians.
 - ✓ **Data collection:** These errors occur at time of data collection and are harder to correct.
- We saw variable Item_Weight has missing values. Item_Weight is a continuous variable. Hence, in this case we can impute missing values with mean / median of item_weight.
- Various Methods- Deletion or **Mean/ Mode/ Median Imputation** or Prediction Model.

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

```
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
```

```
0          0
```

```
Outlet_Type      Item_Outlet_Sales
```

```
0          0
```

Hence, we see that column Item_Weight has 1463 missing values. Let's get more inferences from this data.

```
summary(train)
```

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility
Length:8523	Min. : 4.555	Length:8523	Min. :0.00000
Class :character	1st Qu.: 8.774	Class :character	1st Qu.:0.02699
Mode :character	Median :12.600	Mode :character	Median :0.05393
	Mean :12.858		Mean :0.06613
	3rd Qu.:16.850		3rd Qu.:0.09459
	Max. :21.350		Max. :0.32839
	NA's :1463		

Item_Type	Item_MRP	Outlet_Identifier
Length:8523	Min. : 31.29	Length:8523
Class :character	1st Qu.: 93.83	Class :character
Mode :character	Median :143.01	Mode :character
	Mean :140.99	
	3rd Qu.:185.64	
	Max. :266.89	

Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type
Min. :1985	Length:8523	Length:8523
1st Qu.:1987	Class :character	Class :character
Median :1999	Mode :character	Mode :character
Mean :1998		
3rd Qu.:2004		
Max. :2009		

Outlet_Type	Item_Outlet_Sales
Length:8523	Min. : 33.29
Class :character	1st Qu.: 834.25
Mode :character	Median : 1794.33
	Mean : 2181.29
	3rd Qu.: 3101.30
	Max. :13086.97

Trouble with Continuous variables & Categorical variables

It's important to learn to deal with continuous and categorical variables separately in a data set. In other words, they need special attention.

If you are still confused, I'll suggest you to once again look at the data set using *str()* and proceed. Weight, Visibility, MRP and the Year of Establishment are numeric variables, others are categorical. Weight, Visibility, MRP and the Year of Establishment are numeric variables, others are categorical.

Categorical variables in data set

```
table (train$Item_Fat_Content)
```

```
table(train$Item_Type)
```

```
table(train$Outlet_Location_Type)
```

```
table(train$Outlet_Type)
```

```
table(train$Outlet_Size)
```


Here are some quick inferences drawn from variables in train data set:

- Minimum value of item_visibility is 0. Practically, this is not possible. If an item occupies shelf space in a grocery store, it ought to have some visibility. We'll treat all 0's as missing values.
- Item_Weight has 1463 missing values (already explained above).
- Item_Fat_Content has mis-matched factor levels.
- Outlet_Size has unmatched factor levels.

These inference will help us in treating these variables more accurately.

Let's first combine the data sets. This will save our time as we don't need to write separate codes for train and test data sets. To combine the two data frames, we must make sure that they have equal columns, which is not the case.

```
dim(train)
```

```
dim(test)
```

```
> [1] 8523 12
```

```
> [1] 5681 11
```

Test data set has one less column (response variable). Let's first add the column. We can give this column any value. An intuitive approach would be to extract the mean value of sales from train data set and use it as placeholder for test variable Item_Outlet_Sales. Anyways, let's make it simple for now. I've taken a value 1. Now, we'll combine the data sets.

```
test$Item_Outlet_Sales=1 #mean(train$Item_Outlet_Sales)
combi <- rbind(train, test)
```

Impute missing value by median. I'm using median because it is known to be highly robust to outliers.

```
combi$Item_Weight[is.na(combi$Item_Weight)] <-  
median(combi$Item_Weight, na.rm = TRUE)
```

```
# na.rm = TRUE excluding missing values from analyses
```

```
table(is.na(combi$Item_Weight))
```

```
>FALSE  
14204
```

Let's take up *Item_Visibility*. In the graph above, we saw item visibility has zero value also, which is practically not feasible. Hence, we'll consider it as a missing value and once again make the imputation using median.

```
combi$Item_Visibility [combi$Item_Visibility == 0] <-  
median(combi$Item_Visibility)
```

```
summary(combi$Item_Visibility)
```

```
      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.   
0.003575 0.033143 0.054023 0.069296 0.094037 0.328391
```

Let's proceed to categorical variables now. During exploration, we saw there are mis-matched levels in variables which needs to be corrected. We can rename various levels of Item_Fat_Content.

```
table(combi$Item_Fat_Content)
```

LF	Low Fat	Regular	Low fat	reg
522	8485	4824	178	195

```
library(plyr)
```

If x is a factor, the named levels of the factor will be replaced with the new values.

```
combi$Item_Fat_Content = revalue(combi$Item_Fat_Content, c("LF" = "Low Fat"))
```

```
combi$Item_Fat_Content = revalue(combi$Item_Fat_Content, c("reg" = "Regular"))
```

```
combi$Item_Fat_Content = revalue(combi$Item_Fat_Content, c("low fat" = "Low Fat"))
```

```
table(combi$Item_Fat_Content)
```

Low Fat	Regular
9185	5019

```
table(combi$Outlet_Size)
```

	High	Medium	Small
4016	1553	4655	3980

```
combi$Outlet_Size [combi$Outlet_Size == ""] <- "Other"
```

	High	Medium	Other	Small
1553	4655	4016	3980	

Using the command above, I've assigned the name 'Other' to unnamed level in *Outlet_Size* variable.

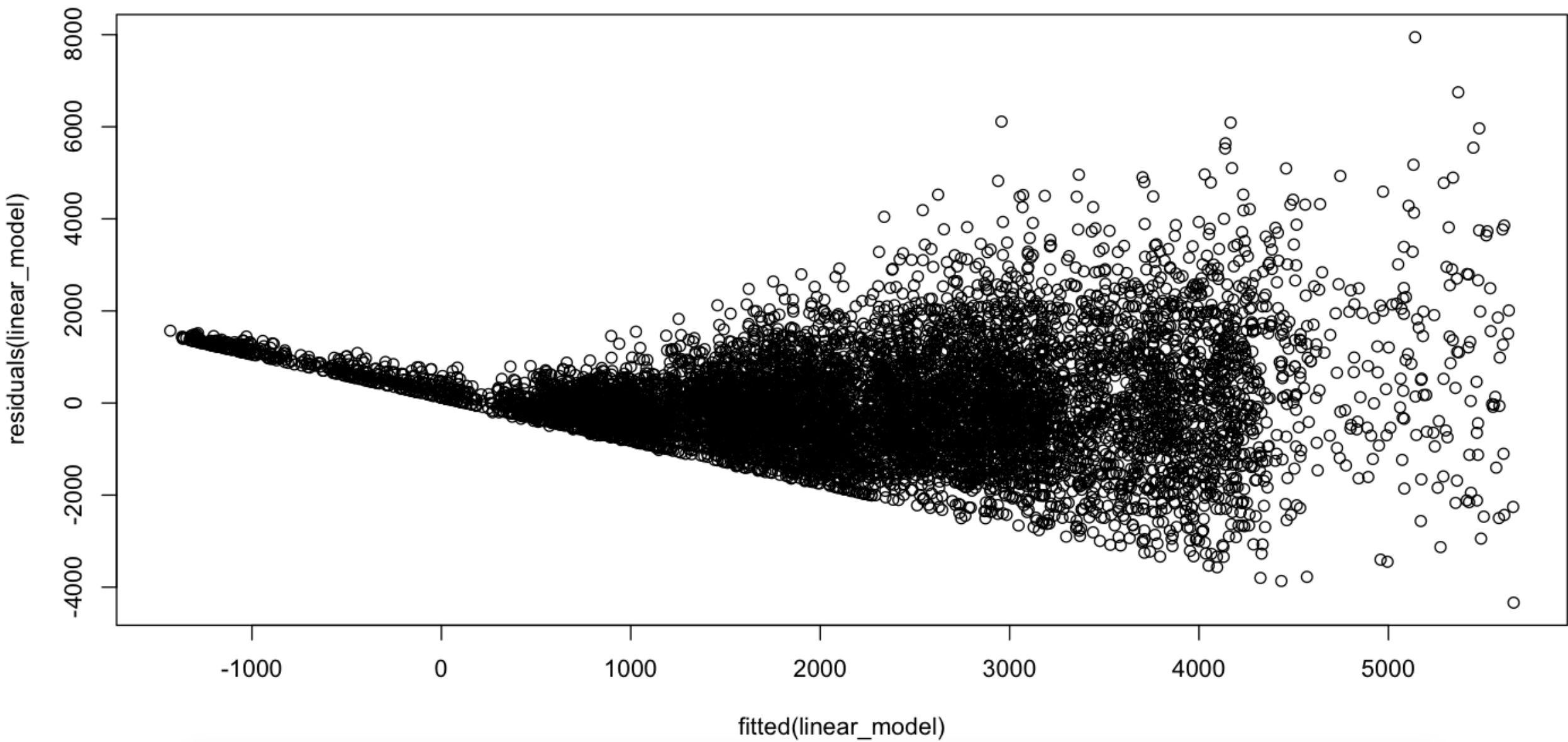
Predictive Modeling/Machine Learning in R

- Let's now build out first regression model on this data set. R uses *lm()* function for regression.

```
linear_model <- lm(Item_Outlet_Sales ~ ., data = new_train)
summary(linear_model)
```

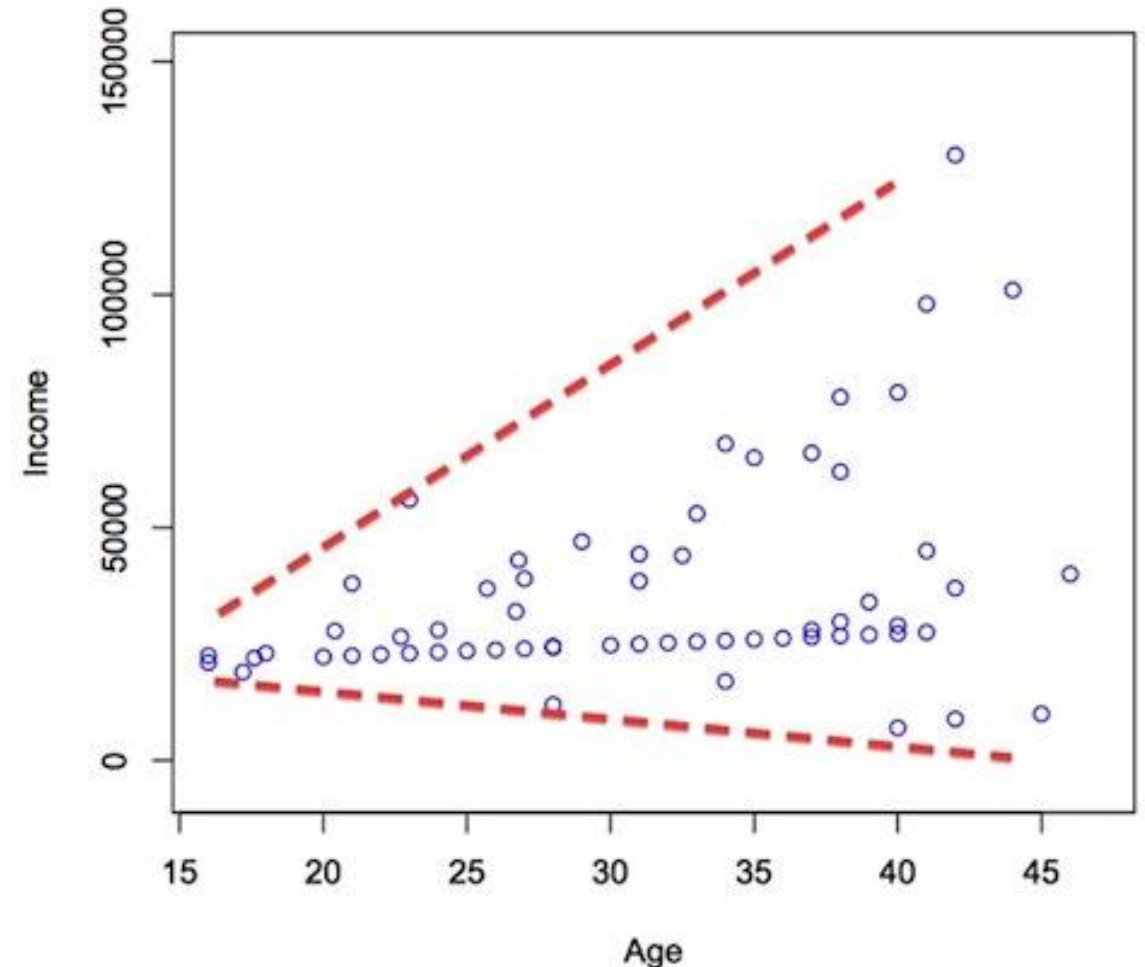
- Adjusted R^2 measures the goodness of fit of a regression model. Higher the R^2 , better is the model. Our **$R^2 = 0.5623$** .
- Let's try to create a more robust regression model. Currently, we have used a simple model.

- We have got **$R^2 = 0.5623$** .
- Let's check out regression plot to find out more ways to improve this model.
`plot(fitted(linear_model), residuals(linear_model));`
- Residual values are the difference between actual and predicted outcome values. Fitted values are the predicted values. If you see carefully, you'll discover it as a funnel shape graph (from right to left). The shape of this graph suggests that our model is suffering from heteroskedasticity (unequal variance in error terms). Had there been constant variance, there would be no pattern visible in this graph.



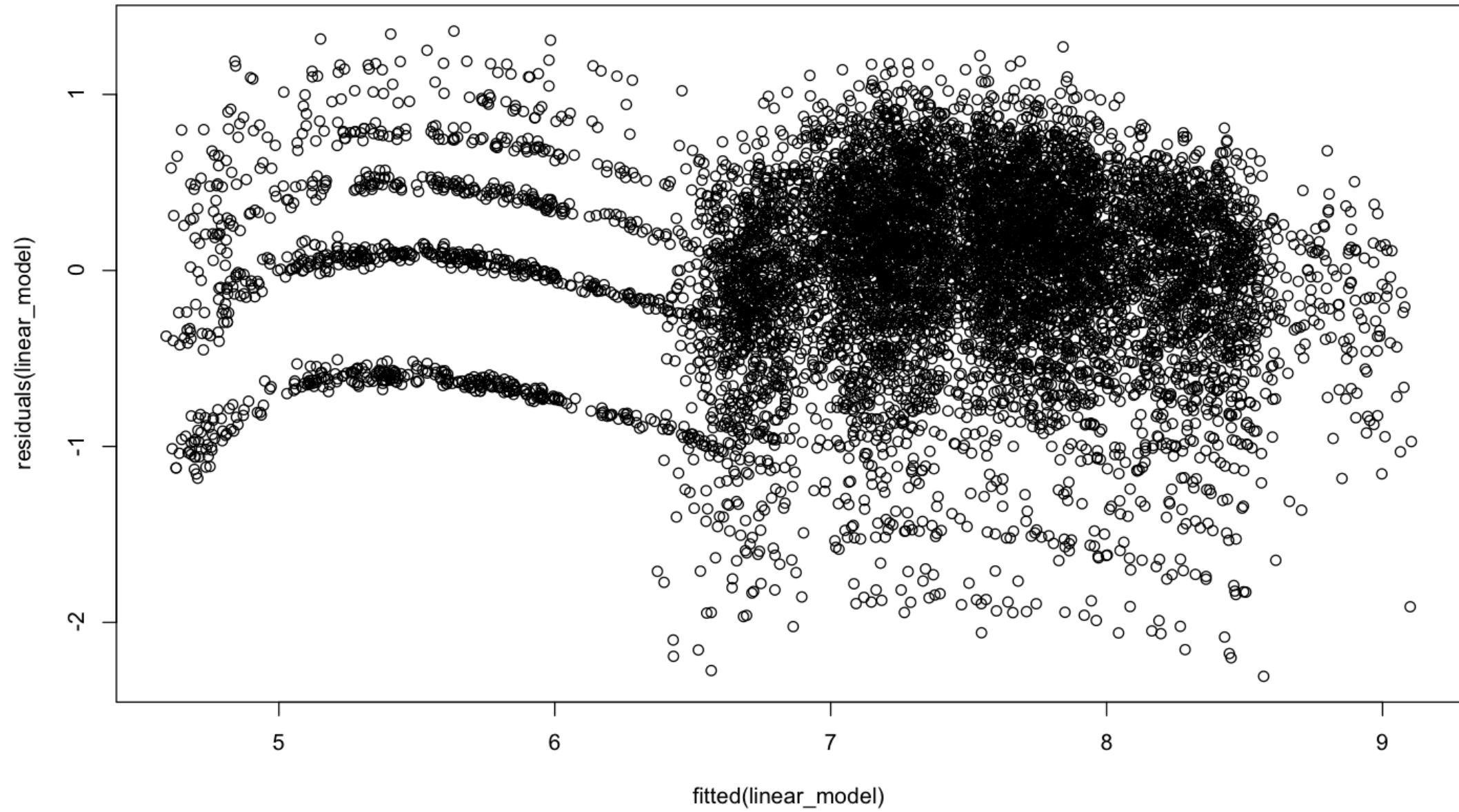
Heteroskedasticity

- A scatterplot of heteroskedastic variables will often create a cone-like shape, as the scatter (or variability) of the dependent variable (DV) widens or narrows as the value of the independent variable (IV) increases.
- For example: annual income might be a heteroscedastic variable when predicted by age. More commonly, teen workers earn close to the minimum wage, so there isn't a lot of variability during the teen years. However, as teens turn into 20-somethings, and 20-somethings into 30-somethings, some will tend to shoot-up the tax brackets, while others will increase more gradually (or perhaps not at all, unfortunately). Put simply, the gap between the "haves" and the "have-nots" is likely to widen with age.
- A plot of the association between age and income would demonstrate heteroscedasticity, as shown on the right side.



- A common practice to tackle heteroskedasticity is by taking the log of response variable. Let's do it and check if we can get further improvement.
- `linear_model <- lm(log(Item_Outlet_Sales) ~ ., data = new_train)`
`summary(linear_model)`
- Congrats! We have got an improved model with **$R^2 = 0.72$** . Now, we are on the right path.

```
plot(fitted(linear_model), residuals(linear_model));
```



- This model can be further improved by detecting outliers and high leverage points. For now, I leave that part to you!
- Let's check our RMSE so that we can compare it with other algorithms (which you must give a go!)
- To calculate RMSE, we can load a package named *Metrics*.
- ```
install.packages("Metrics")
library(Metrics)
rmse(new_train$Item_Outlet_Sales, exp(linear_model$fitted.values))
```
- ```
[1] 1140.004
```

Thank you

You can reach out to me at anuj112358@gmail.com