Big Mart Data

Analysis using R programming

The Big Mart dataset presents a comprehensive view of retail operations, incorporating detailed information about various items, outlet characteristics, and sales metrics. It serves as a valuable resource for businesses seeking to enhance their understanding of consumer behavior and refine their marketing strategies. By analyzing item specifics, such as weight, visibility, and maximum retail price, companies can tailor product attributes to better meet customer preferences and demands. Additionally, the dataset provides insights into outlet features and categorizations, enabling businesses to optimize resource allocation based on location-specific factors and outlet performance. Furthermore, the inclusion of sales indicators allows for the identification of trends and patterns, facilitating informed decision-making processes regarding inventory management, pricing strategies, and promotional activities. Overall, the Big Mart dataset serves as a strategic tool for retailers, empowering them to make data-driven decisions that drive growth, improve customer satisfaction, and maximize profitability in a competitive market landscape.

Here's a general interpretation of columns in Big Mart data:

- **Item_Identifier:** This column likely contains unique identifiers for each item sold in the store. It could be alphanumeric codes or product IDs.
- **Item_Weight:** This column may represent the weight of each item sold. It helps in understanding the physical characteristics of the products being sold.
- **Item_Fat_Content:** This column might indicate the fat content of the items, typically categorized as 'Low Fat' or 'Regular'.
- **Item_Visibility:** This column could represent the percentage of shelf space allocated to the particular item in the store. Higher visibility might correlate with higher sales.
- **Item_Type:** This column would likely specify the category or type of each item being sold, such as 'Dairy', 'Frozen Foods', 'Fruits and Vegetables', etc.
- **Item_MRP:** This column would contain the maximum retail price of each item. It helps in understanding the pricing strategy of the store.
- **Outlet_Identifier:** This column could contain unique identifiers for each outlet or store where the sales are made.
- **Outlet_Establishment_Year:** This column likely indicates the year when each outlet was established. It helps in understanding the age of each store. The value range from 1985 to 2009.
- **Outlet_Size:** This column may represent the size of each outlet, categorized as 'Small', 'Medium', or 'High'.
- **Outlet_Location_Type:** This column could indicate the type of location where each outlet is situated, such as 'Tier1', 'Tier2', or 'Tier3'.
- **Outlet_Type:** This column would specify the type of outlet, such as 'Supermarket Type1', 'Supermarket Type2', 'Grocery Store', etc.

#import & Read data through read.csv()

mydata<-read.csv("C:/Users/Rama/Downloads/Test.csv")
mydata

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
1	FDW58	20.750	Low Fat	0.007564836	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
2	FDW14	8.300	Regular	0.038427677	Dairy	87.3198	OUT017	2007	NA	Tier 2	Supermarket Type1
3	NCN55	14.600	Low Fat	0.099574908	Others	241.7538	OUT010	1998	NA	Tier 3	Grocery Store
4	FDQ58	7.315	Low Fat	0.015388393	Snack Foods	155.0340	OUT017	2007	NA	Tier 2	Supermarket Type1
5	FDY38	NA	Regular	0.118599314	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3
6	FDH56	9.800	Regular	0.063817206	Fruits and Vegetables	117.1492	OUT046	1997	Small	Tier 1	Supermarket Type1
7	FDL48	19.350	Regular	0.082601537	Baking Goods	50.1034	OUT018	2009	Medium	Tier 3	Supermarket Type2
8	FDC48	NA	Low Fat	0.015782495	Baking Goods	81.0592	OUT027	1985	Medium	Tier 3	Supermarket Type3
9	FDN33	6.305	Regular	0.123365446	Snack Foods	95.7436	OUT045	2002	NA	Tier 2	Supermarket Type1
10	FDA36	5.985	Low Fat	0.005698435	Baking Goods	186.8924	OUT017	2007	NA	Tier 2	Supermarket Type1
11	FDT44	16.600	Low Fat	0.103569075	Fruits and Vegetables	118.3466	OUT017	2007	NA	Tier 2	Supermarket Type1
12	FDQ56	6.590	Low Fat	0.105811470	Fruits and Vegetables	85.3908	OUT045	2002	NA	Tier 2	Supermarket Type1
13	NCC54	NA	Low Fat	0.171079215	Health and Hygiene	240.4196	OUT019	1985	Small	Tier 1	Grocery Store
14	FDU11	4.785	Low Fat	0.092737611	Breads	122.3098	OUT049	1999	Medium	Tier 1	Supermarket Type1
15	DRL59	16.750	Low Fat	0.021206464	Hard Drinks	52.0298	OUT013	1987	High	Tier 3	Supermarket Type1
16	FDM24	6.135	Regular	0.079450700	Baking Goods	151.6366	OUT049	1999	Medium	Tier 1	Supermarket Type1
17	FDI57	19.850	Low Fat	0.054135210	Seafood	198.7768	OUT045	2002	NA	Tier 2	Supermarket Type1
18	DRC12	17.850	Low Fat	0.037980963	Soft Drinks	192.2188	OUT018	2009	Medium	Tier 3	Supermarket Type2
19	NCM42	NA	Low Fat	0.028184344	Household	109.6912	OUT027	1985	Medium	Tier 3	Supermarket Type3
20	FDA46	13.600	Low Fat	0.196897637	Snack Foods	193.7136	OUT010	1998	NA	Tier 3	Grocery Store
21	FDA31	7.100	Low Fat	0.109920138	Fruits and Vegetables	175.0080	OUT013	1987	High	Tier 3	Supermarket Type1

Exploratory Data Analysis

to get the summary of data using summary()

summary(mydata)

```
R 4.3.1 · ~/
> summary(mydata)
 Item_Identifier     Item_Weight     Item_Fat_Content  
 Length:5681        Min.   : 4.555   Length:5681       
 Class :character   1st Qu.: 8.645   Class :character  
 Mode  :character   Median :12.500   Mode  :character  
                    Mean   :12.696                     
                    3rd Qu.:16.700                     
                    Max.   :21.350                     
                    NA's   :976                        
 Item_Visibility    Item_Type             Item_MRP     
 Min.   :0.00000   Length:5681        Min.   : 31.99  
 1st Qu.:0.02705   Class :character   1st Qu.: 94.41  
 Median :0.05415   Mode  :character   Median :141.42  
 Mean   :0.06568                      Mean   :141.02  
 3rd Qu.:0.09346                      3rd Qu.:186.03  
 Max.   :0.32364                      Max.   :266.59  

 Outlet_Identifier  Outlet_Establishment_Year
 Length:5681        Min.   :1985             
 Class :character   1st Qu.:1987             
 Mode  :character   Median :1999             
                    Mean   :1998             
                    3rd Qu.:2004             
                    Max.   :2009             
```

```
 Outlet_Size        Outlet_Location_Type Outlet_Type       
 Length:5681        Length:5681          Length:5681       
 Class :character   Class :character     Class :character  
 Mode  :character   Mode  :character     Mode  :character  
```

the "summary" function provides a concise overview of the data, including measures of central tendency (Mean, Median, 1st Quartile,3rd Quartile, Max) dispersion, and other relevant statistics(Length, Class, Mode).

to display the structure of the data

str(mydata)

```
> # to display the structure of the data
> str(mydata)
'data.frame':	5681 obs. of  11 variables:
 $ Item_Identifier          : chr  "FDW58" "FDW14" "NCN55" "FDQ58"
 ...
 $ Item_Weight              : num  20.75 8.3 14.6 7.32 NA ...
 $ Item_Fat_Content         : chr  "Low Fat" "reg" "Low Fat" "Low
Fat" ...
 $ Item_Visibility          : num  0.00756 0.03843 0.09957 0.01539
0.1186 ...
 $ Item_Type                : chr  "Snack Foods" "Dairy" "Others"
"Snack Foods" ...
 $ Item_MRP                 : num  107.9 87.3 241.8 155 234.2 ...
 $ Outlet_Identifier        : chr  "OUT049" "OUT017" "OUT010" "OUT
017" ...
 $ Outlet_Establishment_Year: int  1999 2007 1998 2007 1985 1997 2
009 1985 2002 2007 ...
 $ Outlet_Size              : chr  "Medium" "" "" "" ...
 $ Outlet_Location_Type     : chr  "Tier 1" "Tier 2" "Tier 3" "Tie
r 2" ...
 $ Outlet_Type              : chr  "Supermarket Type1" "Supermarke
t Type1" "Grocery Store" "Supermarket Type1" ...
```

Str() provides a concise and informative summary of the internal structure of an R object, including its type, length, and contents. The primary purpose of the "str" function is to help users understand the underlying structure of their data.

to get the first few rows

head(mydata)

```
> head(mydata)
  Item_Identifier Item_Weight Item_Fat_Content Item_Visibility
1           FDW58      20.750          Low Fat     0.007564836
2           FDW14       8.300              reg     0.038427677
3           NCN55      14.600          Low Fat     0.099574908
4           FDQ58       7.315          Low Fat     0.015388393
5           FDY38          NA          Regular     0.118599314
6           FDH56       9.800          Regular     0.063817206
              Item_Type Item_MRP Outlet_Identifier
1           Snack Foods 107.8622            OUT049
2                 Dairy  87.3198            OUT017
3                Others 241.7538            OUT010
4           Snack Foods 155.0340            OUT017
5                 Dairy 234.2300            OUT027
6 Fruits and Vegetables 117.1492            OUT046
  Outlet_Establishment_Year Outlet_Size Outlet_Location_Type
1                      1999      Medium               Tier 1
2                      2007                           Tier 2
3                      1998                           Tier 3
4                      2007                           Tier 2
5                      1985      Medium               Tier 3
6                      1997       Small               Tier 1
```

```
       Outlet_Type
1 Supermarket Type1
2 Supermarket Type1
3       Grocery Store
4 Supermarket Type1
5 Supermarket Type3
6 Supermarket Type1
>
```

The "head()" function in R is used to view the first few rows of a data object. It allows users to quickly inspect the structure and content of a dataset without displaying the entire dataset, which can be particularly useful for large datasets

 # to show the last few rows

tail(mydata)

```
> tail(mydata)
     Item_Identifier Item_Weight Item_Fat_Content
5676           FDW46        13.0          Regular
5677           FDB58        10.5          Regular
5678           FDD47         7.6          Regular
5679           NCO17        10.0          Low Fat
5680           FDJ26        15.3          Regular
5681           FDU37         9.5          Regular
     Item_Visibility          Item_Type Item_MRP
5676      0.07041096        Snack Foods  63.4484
5677      0.01349647        Snack Foods 141.3154
5678      0.14299090      Starchy Foods 169.1448
5679      0.07352856 Health and Hygiene 118.7440
5680      0.00000000             Canned 214.6218
5681      0.10472015             Canned  79.7960
     Outlet_Identifier Outlet_Establishment_Year Outlet_Size
5676            OUT049                      1999      Medium
5677            OUT046                      1997       Small
5678            OUT018                      2009      Medium
5679            OUT045                      2002
5680            OUT017                      2007
5681            OUT045                      2002
```

```
     Outlet_Location_Type       Outlet_Type
5676               Tier 1 Supermarket Type1
5677               Tier 1 Supermarket Type1
5678               Tier 3 Supermarket Type2
5679               Tier 2 Supermarket Type1
5680               Tier 2 Supermarket Type1
5681               Tier 2 Supermarket Type1
>
```

"tail" displays the last few rows. It allows users to inspect the end of a dataset without needing to view the entire dataset, which is particularly useful for large datasets.

 # To show correlation between item_mrp and item_visibility

cor(mydata$Item_MRP,mydata$Item_Visibility)

```
> cor(mydata$Item_MRP,mydata$Item_Visibility)
[1] -0.01401297
>
```

correlation is a statistical measure that quantifies the strength and direction of the relationship between two variables. As the correlation coefficient is near 0 indicating no linear relationship between the two variables.

Data cleaning

to remove missing value

missing_value<-is.na(mydata)
col_missing_value<-colSums(missing_value)
col_missing_value

```
> col_missing_value<-colSums(missing_value)
> col_missing_value
          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 
                        0                         0 
              Outlet_Type 
                        0 
> 
```

is.na() is a function used to identify missing or NA (Not Available) values in a dataset. It returns a logical vector indicating whether each element in the input object is NA or not. When combined with colSum() function, is.na() can be used to calculate the number of missing values in each column of a data frame. In this dataset, there is no NA value in any column except Item_Weight

Impute missing values with mean

mydata$Item_Weight[is.na(mydata$Item_Weight)] <- mean(mydata$Item_Weight, na.rm = TRUE)
mydata$Item_Weight
missing_value1<-is.na(mydata)
col_missing_value1<-colSums(missing_value1)
col_missing_value1

```
> col_missing_value1
          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 
> 
```

method to replace the missing value is by replacing the missing value by the mean of other available values or just simply remove them. Here we replaced them with mean value. Now if we use colSums function to find is.na value it is 0 in all columns

 # Identify and remove duplicates

deduplicated_data <- unique(mydata)
deduplicated_data

the unique() function is used to extract unique elements from a vector or a data frame column. We can use this function to get only the unique value and eliminate the duplicate values

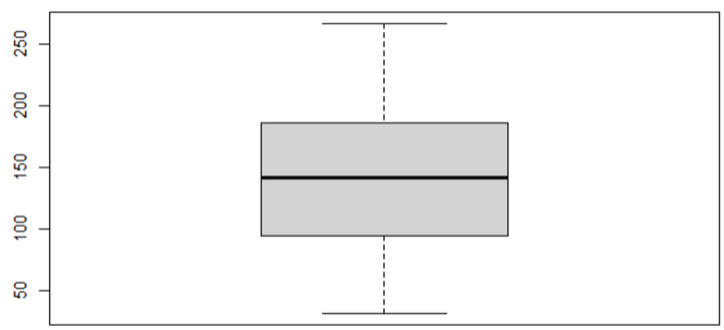
 # to remove blanks from outlet size column

data<-subset(mydata,Outlet_Size!="")
data

subset() function is used to subset a data frame based on specified conditions. It allows you to extract rows or columns from a data frame that meet certain criteria. Here the criteria is to remove blanks from Outlet size Column so we can use subset() function for it

 # Visualize outliers with boxplot

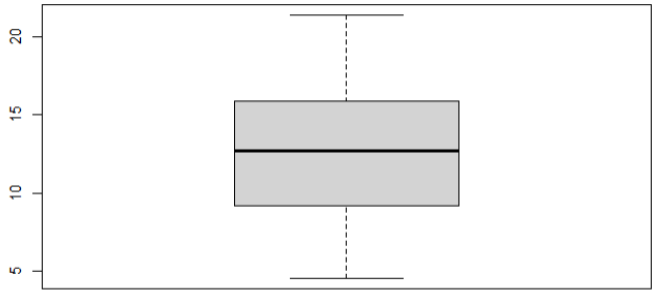
boxplot(data$Item_MRP)



boxplot() function is used to create boxplots, which are graphical representations of the distribution of numerical data. The box in the middle represents the interquartile range (IQR), with the line inside the box indicating the median (50th percentile) of the data. The "whiskers" extend from the box to the minimum and maximum values within 1.5 times the IQR from the first and third quartiles, respectively. Points outside this range are considered outliers and are plotted individually. Here in this graph there is no Outliers present & median of Item_MRP is around 140.

 # boxplot of Item Weight

boxplot(data$Item_Weight)



Similarly Item_Weight also has no Outlier's and has median around 13.

 # Covariance between Item_MRP & Item_Weight

cov(data$Item_MRP,data$Item_Weight)

```
>
[1] 9.374604
```

the cov() function is used to calculate the covariance between two numeric vectors or columns in a data frame. Covariance measures the degree to which the variables change together. A positive covariance indicates that the variables tend to increase or decrease together, while a negative covariance indicates that one variable tends to increase as the other decreases. Covariance is calculated as the average of the product of the deviations of each variable from their respective means. So Item_MRP & Item_Weight has covariance of 9.37.

#Aggregation with dplyr for Summarized Insights

 # load package dplyr

 # call the function

library(dplyr)

 # using dplyr finding avg_MRP per item_type

```
data%>%
group_by(Item_Type)  %>%
  summarise(avg_Item_MRP=mean(Item_MRP))
```

```
> mydata%>%
+ group_by(Item_Type)  %>%
+   summarise(avg_Item_MRP=mean(Item_MRP))
# A tibble: 16 × 2
   Item_Type             avg_Item_MRP
   <chr>                        <dbl>
 1 Baking Goods                  129.
 2 Breads                        142.
 3 Breakfast                     132.
 4 Canned                        137.
 5 Dairy                         145.
 6 Frozen Foods                  133.
 7 Fruits and Vegetables         144.
 8 Hard Drinks                   138.
 9 Health and Hygiene            137.
10 Household                     148.
11 Meat                          141.
12 Others                        135.
13 Seafood                       140.
14 Snack Foods                   147.
15 Soft Drinks                   142.
16 Starchy Foods                 152.
```

- *The `dplyr` package provides a set of functions for data manipulation and transformation. `%>%` is the pipe operator in R, which allows you to chain together multiple operations, passing the result of one function as the input to the next function.*
- *Group_by () groups the data by the "Item_Type" column. It means that subsequent operations will be applied to each group separately.*
- *Summarize() function summarizes the data within each group created by `group_by()`. We have summarize the avg_ MRP by using mean().*
- *The result of this summarization is a new data frame with one row for each unique "Item_Type", and a column "avg_Item_MRP" containing the average MRP (Maximum Retail Price) for each group of items.*
- *Here Starchy food has highest avg_Item_MRP whereas baking goods has the least*

 # grouping data by Outlet type and summarizing by Maximum Item visibility

```
data%>%
 group_by(Outlet_Type)%>%
 reframe(Max_Item_Visibility=max(Item_Visibility))
```

```
# A tibble: 4 × 2
  Outlet_Type       Max_Item_Visibility
  <chr>                           <dbl>
1 Grocery Store                   0.324
2 Supermarket Type1               0.187
3 Supermarket Type2               0.184
4 Supermarket Type3               0.187
>
```

- *group_by(Outlet_Type): This line groups the data by the "Outlet_Type" column. It means that subsequent operations will be applied to each group separately based on the unique outlet types. summarise(Max_Item_Visibility = max(Item_Visibility)): This line summarizes the data within each group created by group_by().*
- *It calculates the maximum visibility of items (max(Item_Visibility)) within each group of outlet types.*

- *The result is a new data frame with one row for each unique outlet type, containing the maximum item visibility value for each group. Grocery Store has the higher shelf space available to various goods, supermarket type 2 has the least*

Filtering the outlet establishment year as 2007 and arranging Item weight in descending order.

top_10_itemweight<-data%>%
filter(Outlet_Establishment_Year==2007)%>%
arrange(desc(Item_Weight))
head(top_10_itemweight,10)

```
> head(top_10_itemweight,10)
   Item_Identifier Item_Weight Item_Fat_Content Item_Visibility
1            FDR07       21.35          Low Fat      0.07818366
2            FDC02       21.35          Low Fat      0.06921176
3            FDA45       21.25          Low Fat      0.00000000
4            FDG35       21.20          Regular      0.00000000
5            FDO01       21.10          Regular      0.02083585
6            NCE42       21.10          Low Fat      0.01066226
7            FDP59       20.85          Regular      0.05678511
8            FDB45       20.85          Low Fat      0.02145044
9            FDS24       20.85          Regular      0.06257645
10           FDS60       20.85          Low Fat      0.03263207
               Item_Type Item_MRP Outlet_Identifier Outlet_Establishment_Year
1  Fruits and Vegetables  96.8094            OUT017                      2007
2                 Canned 258.8278            OUT017                      2007
3            Snack Foods 174.5370            OUT017                      2007
4          Starchy Foods 173.0738            OUT017                      2007
5              Breakfast 129.7994            OUT017                      2007
6              Household 231.9958            OUT017                      2007
7                 Breads 102.2648            OUT017                      2007
8  Fruits and Vegetables 103.6306            OUT017                      2007
9           Baking Goods  89.9514            OUT017                      2007
10          Baking Goods 179.5660            OUT017                      2007
```

```
   Outlet_Size Outlet_Location_Type       Outlet_Type
1                            Tier 2 Supermarket Type1
2                            Tier 2 Supermarket Type1
3                            Tier 2 Supermarket Type1
4                            Tier 2 Supermarket Type1
5                            Tier 2 Supermarket Type1
6                            Tier 2 Supermarket Type1
7                            Tier 2 Supermarket Type1
8                            Tier 2 Supermarket Type1
9                            Tier 2 Supermarket Type1
10                           Tier 2 Supermarket Type1
```

- *filter(Outlet_Establishment_Year == 2007): This line filters the data to include only rows where the "Outlet_Establishment_Year" column equals 2007.*

It means that only data related to outlets established in 2007 will be retained.

- *arrange(desc(Item_Weight)): This line arranges the filtered data in descending order based on the "Item_Weight" column.*

It means that the data will be sorted from highest to lowest based on the weight of the items.

- *top_10_itemweight <-: This assigns the result of the filtering and arranging operations to a new data frame called "top_10_itemweight".*
- *head(top_10_itemweight, 10): This line displays the first 10 rows of the "top_10_itemweight" data frame.*
- *It allows you to inspect the top 10 rows of the filtered and arranged data, showing the items with the highest weights in outlets established in 2007.*
- *Highest Item weight is 21.35 kg of fruit & Vegetable available in supermarket 1*
- *It also shows that all top 10 highest Item weight in year 2007 belongs to Tier 2 location type & outlet type of supermarket type 1*

 # grouping data by Outlet type and summarizing the data by mean MRP

data%>%

group_by(Outlet_Type)%>%
summarise(avg_item_MRP=mean(Item_MRP))

```
Console  Terminal  Background Jobs
R 4.3.1 · ~/
  Outlet_Type         avg_item_MRP
  <chr>                      <dbl>
1 Grocery Store               142.
2 Supermarket Type1           141.
3 Supermarket Type2           140.
4 Supermarket Type3           143.
>
```

- *group_by(Outlet_Type): This line groups the data by the "Outlet_Type" column. It means that subsequent operations will be applied to each group separately based on the unique outlet types.*
- *summarise(avg_item_MRP = mean(Item_MRP)): This line summarizes the data within each group created by group_by().*
- *It calculates the average MRP (Maximum Retail Price) of items (mean(Item_MRP)) within each group of outlet types.*
- *The result is a new data frame with one row for each unique outlet type, containing the average item MRP for each group.*
- *Supermarket type 1 has the highest average MRP while Supermarket type 2 has the lowest*

 # grouping the data by Outlet size and Summarize by Maximum MRP

data%>%
group_by(Outlet_Size)%>%
summarise(Max_Item_MRP<-max(Item_MRP))

```
Console  Terminal  Background Jobs
R 4.3.1 · ~/
  Outlet_Size `Max_Item_MRP ← max(Item_MRP)`
  <chr>                                <dbl>
1 High                                  265.
2 Medium                                266.
3 Small                                 267.
>
```

- *group_by(Outlet_Size): This line groups the data by the "Outlet_Size" column. It means that subsequent operations will be applied to each group separately based on the unique outlet sizes. summarise(Max_Item_MRP = max(Item_MRP)): This line summarizes the data within each group created by group_by().*
- *It calculates the maximum MRP (Maximum Retail Price) of items (max(Item_MRP)) within each group of outlet sizes.*
- *The result is a new data frame with one row for each unique outlet size, containing the maximum item MRP for each group.*
- *Small sized enterprises has higher MRP whereas large smalled enterprises has lower MRP due to economics of small*

Convert your big mart data to a data.table (assuming it's in a data frame called 'big_mart_data')

big_mart_data_dt <- as.data.table(data)

- **as.data.table(data): This line converts the data frame named "data" into a data table using the as.data.table() function from the data.table package.**
- **The resulting data table is assigned to the variable big_mart_data_dt**

Perform operations

data_table<-big_mart_data_dt[, .(Avg_MRP = mean(Item_MRP)), by = .(Item_Type, Outlet_Location_Type,Outlet_Type)]
head(data_table)

```
>    head(data_table)
                 Item_Type Outlet_Location_Type       Outlet_Type
                    <char>               <char>            <char>
1:             Snack Foods               Tier 1 Supermarket Type1
2:                   Dairy               Tier 3 Supermarket Type3
3: Fruits and Vegetables                 Tier 1 Supermarket Type1
4:            Baking Goods               Tier 3 Supermarket Type2
5:            Baking Goods               Tier 3 Supermarket Type3
6:      Health and Hygiene               Tier 1     Grocery Store
    Avg_MRP
      <num>
1: 145.3351
2: 144.8959
3: 137.2942
4: 128.3826
5: 124.9775
6: 120.3811
>
```

- ***[ ] is used to subset and summarize data in the data table.***
- ***.() creates a list of expressions to be calculated or returned.***
- ***.() with .(Item_Type, Outlet_Location_Type, Outlet_Type) defines the grouping variables for summarization.***
- ***.(Avg_MRP = mean(Item_MRP)) calculates the average MRP (Mean Retail Price) of items for each unique combination of "Item_Type", "Outlet_Location_Type", and "Outlet_Type".***
- ***The result is a new data table named data_table containing one row for each unique combination of "Item_Type", "Outlet_Location_Type", and "Outlet_Type", with the corresponding average MRP.***
- ***head(data_table): This line displays the first few rows of the data_table data table, allowing you to inspect the summarized results.***
- ***Snacks foods of tier1 and supermarket type 1 has the highest average MRP***

linear regression of item weight and item visbility on item MRP

model <- lm(Item_MRP ~ Item_Weight + Item_Visibility , data = data)
summary(model)

```
> model <- lm(Item_MRP ~ Item_Weight + Item_Visibility , data = data)
>    summary(model)

Call:
lm(formula = Item_MRP ~ Item_Weight + Item_Visibility, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-111.283  -46.939    0.621   44.622  129.973

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     134.3613     3.4433  39.022   <2e-16 ***
Item_Weight       0.5759     0.2403   2.397   0.0166 *
Item_Visibility -15.7866    19.6147  -0.805   0.4210
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 61.74 on 4072 degrees of freedom
Multiple R-squared:  0.00158,	Adjusted R-squared:  0.001089
F-statistic: 3.221 on 2 and 4072 DF,  p-value: 0.04
```

This code fits a linear regression model to the data, specifically predicting the "Item_MRP" (Maximum Retail Price) based on two predictor variables: "Item_Weight" and "Item_Visibility".

- `lm()` *function is used to fit a linear regression model.*
- `Item_MRP ~ Item_Weight + Item_Visibility` *specifies the model formula, where "Item_MRP" is the response variable to be predicted, and "Item_Weight" and "Item_Visibility" are the predictor variables.*
- `data = data` *specifies the data frame where the variables are located.*
- `summary(model)` *provides a summary of the fitted linear regression model.*

The summary includes coefficients, standard errors, t-values, and p-values for each predictor variable in the model, as well as other statistics such as R-squared, adjusted R-squared, F-statistic, and p-value for the overall model fit.

- *Coefficients represent the estimated effect of each predictor variable on the response variable.*
- *Standard errors indicate the variability of the estimated coefficients.*
- *T-values measure the significance of each coefficient, with higher absolute t-values indicating greater significance. This regression shows that only intercept is statistically significant and Item weight*
- *P-values represent the probability of observing the estimated coefficient if the null hypothesis (no effect) were true. Lower p-values (< 0.05) indicate statistical significance.*

Coefficients:

- *Intercept: The intercept coefficient represents the estimated mean value of "Item_MRP" when both "Item_Weight" and "Item_Visibility" are zero. In this case, it is estimated to be 134.3613.*
- *Item_Weight: The coefficient for "Item_Weight" (0.5759) represents the estimated change in "Item_MRP" for a one-unit increase in "Item_Weight" while holding "Item_Visibility" constant. It is statistically significant at the 0.05 significance level (p-value = 0.0166).*
- *Item_Visibility: The coefficient for "Item_Visibility" (-15.7866) represents the estimated change in "Item_MRP" for a one-unit increase in "Item_Visibility" while holding "Item_Weight" constant. However, it is not statistically significant at the 0.05 significance level (p-value = 0.4210).*

Significance Codes:

- *The significance codes provide a quick way to assess the statistical significance of each coefficient. In this case, "Item_Weight" is significant at the 0.05 level, indicated by *, while "Item_Visibility" is not.*

Residual Standard Error:

- *The residual standard error (61.74) is an estimate of the standard deviation of the residuals, which are the differences between the observed and predicted values of "Item_MRP".*

Multiple R-squared and Adjusted R-squared:

- *These measures indicate the proportion of variance explained by the model.*
- *In this case, the multiple R-squared is 0.00158, suggesting that only a very small proportion of the variability in "Item_MRP" is explained by "Item_Weight" and "Item_Visibility". The adjusted R-squared, which accounts for the number of predictors in the model, is even smaller at 0.001089.*

F-statistic and p-value:

- *The F-statistic tests the overall significance of the regression model.*
- *The p-value associated with the F-statistic is 0.04, which suggests that the model is statistically significant at the 0.05 significance level. However, given the low R-squared values, the practical significance of the model may be limited.*

Overall Interpretation:

- *The model suggests that "Item_Weight" has a significant positive effect on "Item_MRP", but "Item_Visibility" does not have a significant effect.*
- However, the overall explanatory power of the model is very low, suggesting that other factors not included in the model may be important in explaining variations in "Item_MRP".

```
library(caTools)

# Split the data into training and testing sets

set.seed(123)
train_index <- sample.split(data, SplitRatio = 0.8)
train_data <- subset(data,train_index==TRUE)
test_data <- subset(data,train_index==FALSE)
MRP_500<-train_data$Item_MRP[1:500]
weight_500<-train_data$Item_Weight[1:500]
train_data_500<-train_data[1:500,]
# fitting simple regression to the training set
regessor=lm(formula = MRP_500~Weight_500,data=train_data_500)
summary(regessor)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-109.524  -44.390   -0.214   39.864  124.155

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)
(Intercept) 133.9494     9.1105  14.703 <0.0000000000000002 ***
weight_500    0.6037     0.6913   0.873               0.383
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 60.42 on 498 degrees of freedom
Multiple R-squared:  0.001529,  Adjusted R-squared:  -0.0004756
F-statistic: 0.7628 on 1 and 498 DF,  p-value: 0.3829
```

This R code performs several tasks related to data splitting, subsetting, and fitting a simple linear regression model.

1. *Loading the caTools Package:*

library(caTools): This line loads the caTools package, which provides functions for data splitting and manipulation.

2. *Splitting Data into Training and Testing Sets:*

set.seed(123): Sets the random seed for reproducibility.

train_index <- sample.split(data, SplitRatio = 0.8): Splits the data into training and testing sets. 80% of the data will be used for training (train_index == TRUE), and 20% will be used for testing (train_index == FALSE).

3. ***Creating Training and Testing Data:***

train_data <- subset(data, train_index == TRUE): Subsets the original data to create the training data set using rows where train_index is TRUE.
test_data <- subset(data, train_index == FALSE): Subsets the original data to create the testing data set using rows where train_index is FALSE.

4. ***Creating Variables for Model Fitting:***

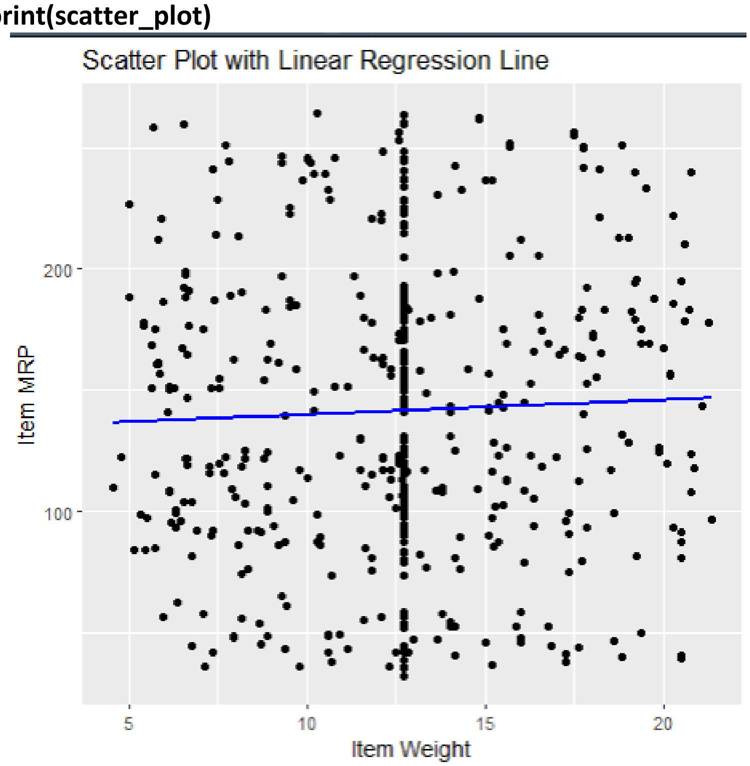
MRP_500 <- train_data$Item_MRP[1:500]: Creates a vector MRP_500 containing the first 500 values of "Item_MRP" from the training data.
weight_500 <- train_data$Item_Weight[1:500]: Creates a vector weight_500 containing the first 500 values of "Item_Weight" from the training data.
train_data_500 <- train_data[1:500,]: Creates a subset train_data_500 containing the first 500 rows of the training data.

5. ***Fitting Simple Linear Regression Model:***

regressor = lm(formula = MRP_500 ~ Weight_500, data = train_data_500): Fits a simple linear regression model to predict MRP_500 based on Weight_500 using the lm() function.
summary(regressor): Prints a summary of the regression model, including coefficients, standard errors, t-values, p-values, and other statistics.

 # Create scatter plot with regression line

```
scatter_plot <- ggplot(train_data_500, aes(x = Weight_500, y = MRP_500)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +  # Add regression line
  labs(x = Item Weight, y = “Item MRP”, title = "Scatter Plot with Linear Regression Line")

print(scatter_plot)
```



This R code generates a scatter plot with a linear regression line overlaid using the `ggplot2` package

- ***scatter_plot <- ggplot(train_data_500, aes(x = Weight_500, y = MRP_500)): This line initializes a scatter plot using the ggplot() function. It specifies the data frame train_data_500 and maps the x-axis to the variable Weight_500 and the y-axis to the variable MRP_500.***

- *geom_point(): This adds points to the plot, representing the observations in the dataset.*
- *geom_smooth(method = "lm", se = FALSE, color = "blue"): This adds a linear regression line to the plot using geom_smooth(). The method = "lm" argument specifies that a linear regression model should be used to fit the line. The se = FALSE argument suppresses the display of the standard error ribbon around the line, and color = "blue" sets the color of the line to blue.*
- *labs(x = "Item Weight", y = "Item MRP", title = "Scatter Plot with Linear Regression Line"): This sets the x-axis label to "Item Weight", the y-axis label to "Item MRP", and the plot title to "Scatter Plot with Linear Regression Line".*
- *print(scatter_plot): This prints the scatter plot with the regression line to the output.*
- *The regression line passes through the* point where the y-intercept is located, which is indicated by the intercept coefficient estimated by the regression model.

 # conducts a one-sample t-test to assess whether the population mean of "Item_MRP" differs significantly from a specified value (in this case, 130)

```
t_test_1 <- t.test(data$Item_MRP, y = NULL,
                   alternative = c("two.sided", "less", "greater"),
                   mu = 130,
                   conf.level = 0.95)
              t_test_1
```

```
        One Sample t-test

data:  data$Item_MRP
t = 11.006, df = 4074, p-value < 0.00000000000000022
alternative hypothesis: true mean is not equal to 130
95 percent confidence interval:
 138.7532 142.5475
sample estimates:
mean of x
 140.6504
```

This R code conducts a one-sample t-test to assess whether the population mean of "Item_MRP" differs significantly from a specified value (in this case, 130)

- *t_test_1 <- t.test(data$Item_MRP, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 130, conf.level = 0.95): This line performs a one-sample t-test using the t.test() function.*
- *data$Item_MRP: This specifies the variable for which we want to conduct the t-test.*
- *y = NULL: Since we're conducting a one-sample t-test, the comparison value (second sample) is set to NULL.*
- *alternative = c("two.sided", "less", "greater"): This argument specifies the alternative hypothesis for the test. It can be one of "two.sided" (default), "less", or "greater".*
- *"two.sided": Tests if the mean is different from the specified value (130) (two-tailed test).*
- *"less": Tests if the mean is less than the specified value (one-tailed test).*
- *"greater": Tests if the mean is greater than the specified value (one-tailed test).*
- *mu = 130: This specifies the value against which the mean of "Item_MRP" is tested.*
- *conf.level = 0.95: This specifies the confidence level for the test. Here, it's set to 95%*

`t_test_1`**: This line prints the results of the t-test, including the test statistic, degrees of freedom, p-value, and confidence interval, to the output.**

- *Test Statistic (t): The value of the t-statistic is 11.006.*
- *Degrees of Freedom (df): The degrees of freedom for the test are 4074.*
- *p-value: The p-value associated with the test is less than 2.2e-16 (a very small number, effectively zero), indicating strong evidence against the null hypothesis.*
- *Alternative Hypothesis: The alternative hypothesis states that the true mean of "Item_MRP" is not equal to 130.*
- *Confidence Interval: The 95% confidence interval for the population mean of "Item_MRP" is (138.7532, 142.5475).*

Sample Estimate:

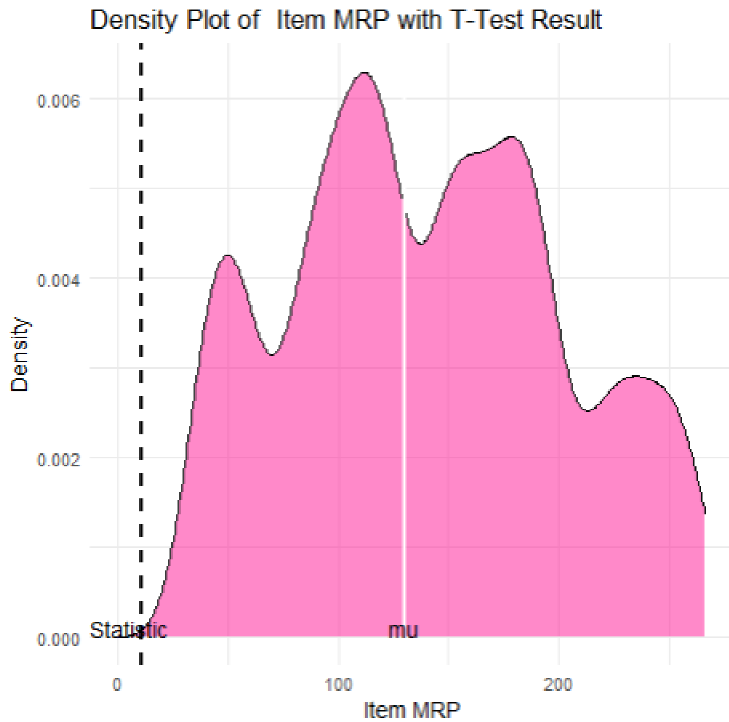
- *Sample Mean: The sample mean of "Item_MRP" is estimated to be 140.6504.*

Overall Interpretation:

- *The one-sample t-test indicates that there is strong evidence to reject the null hypothesis that the population mean of "Item_MRP" is equal to 130.*
- *The confidence interval suggests that we can be 95% confident that the true population mean of "Item_MRP" falls within the range (138.7532, 142.5475), which does not include the hypothesized value of 130.*
- *Therefore, based on the data provided, we can conclude that the population mean of "Item_MRP" is significantly different from 130.*

 # generate a density plot of the "Item_MRP" variable from the dataset, overlaid with a vertical line representing the hypothesized mean and another vertical line representing the calculated t-statistic from the one-sample t-test.

```
density_plot <- ggplot(data, aes(x = Item_MRP)) +
                    geom_density(fill = "deeppink", alpha = 0.5) +
                    geom_vline(xintercept = 130,color="white",lty=1,size=1)+
                    annotate(geom = "text", x = 130, y = 0, label = "mu", vjust = 0) +  # Add label for t-
statistic+
                    geom_vline(xintercept = t_test_1$statistic, color = "black", linetype =
"dashed",size=1) +
                    # Add vertical line for t-statistic
                    annotate(geom = "text", x = t_test_result$statistic, y = 0, label = "T-Statistic",
vjust = 0) +  # Add label for t-statistic
                    labs(x = "Item MRP", y = "Density", title = "Density Plot of  Item MRP with T-Test
Result") +
                    theme_minimal()
                   density_plot
```



- *density_plot <- ggplot(data, aes(x = Item_MRP)) + geom_density(fill = "deeppink", alpha = 0.5): This line initializes a density plot using the ggplot() function. It specifies the data frame data and maps the x-axis to the variable Item_MRP. The geom_density() function adds a density plot of the "Item_MRP" variable with a fill color of "deeppink" and transparency (alpha) set to 0.5.*
- *geom_vline(xintercept = 130, color = "white", lty = 1, size = 1) + annotate(geom = "text", x = 130, y = 0, label = "mu", vjust = 0): This line adds a vertical line at x = 130 (the hypothesized mean) using geom_vline(). The line is white in color, solid (lty = 1), and has a size of 1. The annotate() function adds a text label "mu" at x = 130 with y-coordinate at 0 and vertical justification (vjust) set to 0.*
- *geom_vline(xintercept = t_test_1$statistic, color = "black", linetype = "dashed", size = 1) + annotate(geom = "text", x = t_test_result$statistic, y = 0, label = "T-Statistic", vjust = 0): This line adds a vertical line at the calculated t-statistic from the one-sample t-test using geom_vline(). The line is black in color, dashed (linetype = "dashed"), and has a size of 1. The annotate() function adds a text label "T-Statistic" at the x-coordinate corresponding to the t-statistic value, with y-coordinate at 0 and vertical justification (vjust) set to 0.*
- *labs(x = "Item MRP", y = "Density", title = "Density Plot of Item MRP with T-Test Result"): This line sets the x-axis label to "Item MRP", the y-axis label to "Density",*
- *theme_minimal(): This applies a minimal theme to the plot.*
- *density_plot: This line displays or prints the density plot with the added vertical lines and labels.*

 # perform a Wilcoxon rank sum test, also known as the Mann-Whitney U test, to assess whether there is a statistically significant difference between the distributions of "Item_Weight" and "Item_Visibility".

wilcox.test(data$Item_Weight,data$Item_Visibility)

```
	Wilcoxon rank sum test with continuity correction

data:  data$Item_Weight and data$Item_Visibility
W = 16605625, p-value < 0.00000000000000022
alternative hypothesis: true location shift is not equal to 0
```

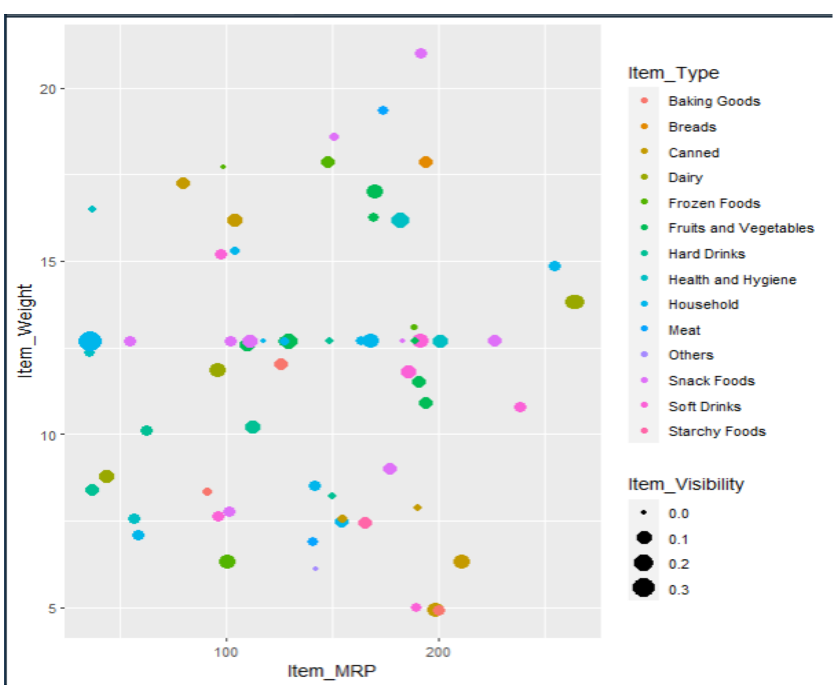
- *wilcox.test(data$Item_Weight, data$Item_Visibility): This line conducts the Wilcoxon rank sum test using the wilcox.test() function.*
- *The Wilcoxon rank sum test is a non-parametric test used to determine if there is a significant difference between the distributions of two independent groups.*
- *The test assesses whether one group tends to have higher or lower values than the other group.*
- *The null hypothesis of the test is that there is no difference between the distributions of the two groups.*

Test Results:

- *Test Statistic (W): The test statistic (W) is 16605625.*
- *p-value: The p-value associated with the test is less than 2.2e-16 (a very small number, effectively zero), indicating strong evidence against the null hypothesis.*
- *Alternative Hypothesis: The alternative hypothesis states that the true location shift (difference in medians) between the two groups is not equal to 0.*
- *Since the p-value is extremely small, we reject the null hypothesis. This suggests that there is a statistically significant difference between the distributions of Item_Weight and Item_Visibility.*
- *Additionally, the alternative hypothesis being "true location shift is not equal to 0" implies that there is a shift in the location (median) of one group compared to the other.*

 # create a scatter plot using the ggplot2 package to visualize the relationship between "Item_MRP" and "Item_Weight" for items labeled as "low fat" in the dataset.

```
dataplot<-data%>%
  filter(Item_Fat_Content=="low fat)
  ggplot(dataplot,aes(x=Item_MRP,y=Item_Weight, color=Item_Type, size=Item_Visibility))+
  geom_point()
```



- *dataplot <- data %>% filter(Item_Fat_Content == "low fat"): This line filters the dataset (data) to include only rows where the "Item_Fat_Content" is "low fat". The filtered dataset is stored in a new dataframe named dataplot.*
- *ggplot(dataplot, aes(x = Item_MRP, y = Item_Weight, color = Item_Type, size = Item_Visibility)): This initializes a scatter plot using the ggplot() function. It specifies the data frame (dataplot) and maps the x-axis to "Item_MRP", the y-axis to "Item_Weight", the color aesthetic to "Item_Type", and the size aesthetic to "Item_Visibility".*
- *geom_point(): This adds points to the scatter plot, with each point representing an observation in the filtered dataset.*

#calculate the range of "Item_MRP" for each combination of "Item_Type" and "Item_Fat_Content" using the dplyr package.

```
mydata1<-mydata%>%
   group_by(Item_Type,Item_Fat_Content)%>%
   summarize(MRP=range(Item_MRP))
  mydata1
```

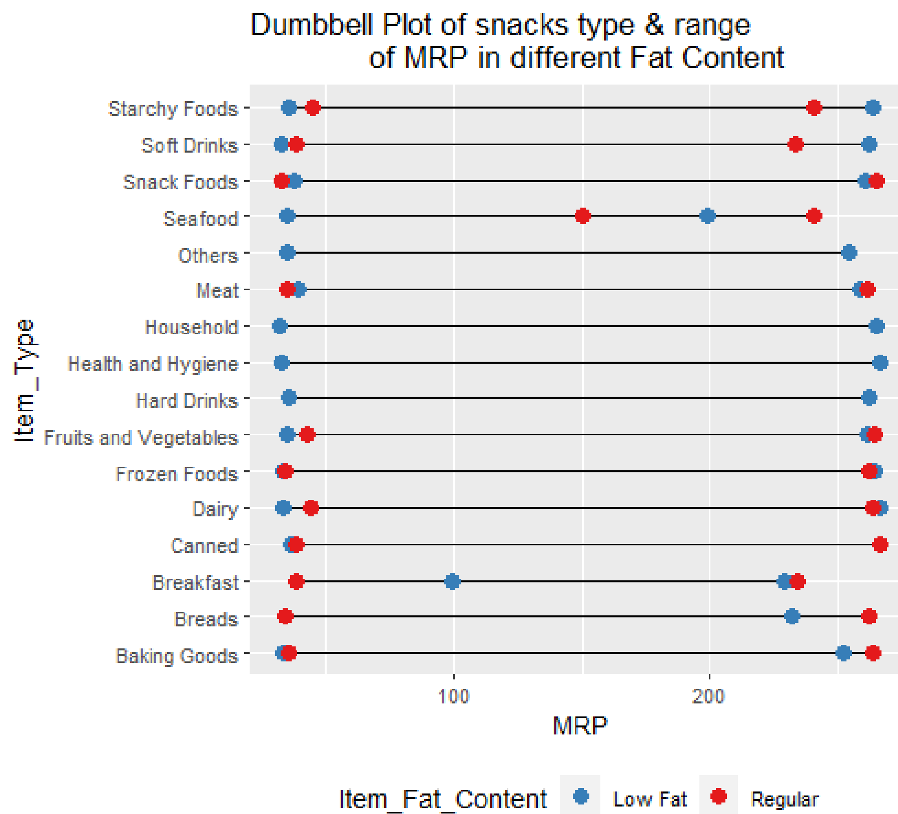
```
   <chr>         <chr>            <dbl>
 1 Baking Goods Low Fat           33.5
 2 Baking Goods Low Fat          252.
 3 Baking Goods Regular           35.3
 4 Baking Goods Regular          263.
 5 Breads       Low Fat           33.7
 6 Breads       Low Fat          232.
 7 Breads       Regular           33.9
 8 Breads       Regular          262.
 9 Breakfast    Low Fat           99.3
10 Breakfast    Low Fat          229.
# i 46 more rows
# i Use `print(n = ...)` to see more rows
```

- *group_by(Item_Type, Item_Fat_Content): This line groups the data by two variables: "Item_Type" and "Item_Fat_Content". This means that subsequent operations will be applied separately for each unique combination of these two variables. summarize(MRP = range(Item_MRP)): This line calculates the range of "Item_MRP" within each group defined by the combination of "Item_Type" and "Item_Fat_Content". The range() function returns the minimum and maximum values of a vector. Here, it calculates the minimum and maximum values of "Item_MRP" for each group.*
- *The result of this operation is a new data frame with three columns: "Item_Type", "Item_Fat_Content", and "MRP", where "MRP" contains the range of "Item_MRP" for each group.*
- *mydata1: This line prints or displays the resulting data frame mydata1, showing the ranges of "Item_MRP" for each combination of "Item_Type" and "Item_Fat_Content".*
- *The data shows health & hygience product has the highest range difference in low fat category*

 # create a dumbbell plot using the ggplot2 package to visualize the range of "Item_MRP" across different types of items (Item_Type), with separate lines for each type of fat content (Item_Fat_Content).

```
ggplot(mydata1, aes(x = MRP, y = Item_Type)) +
   geom_line() +
```

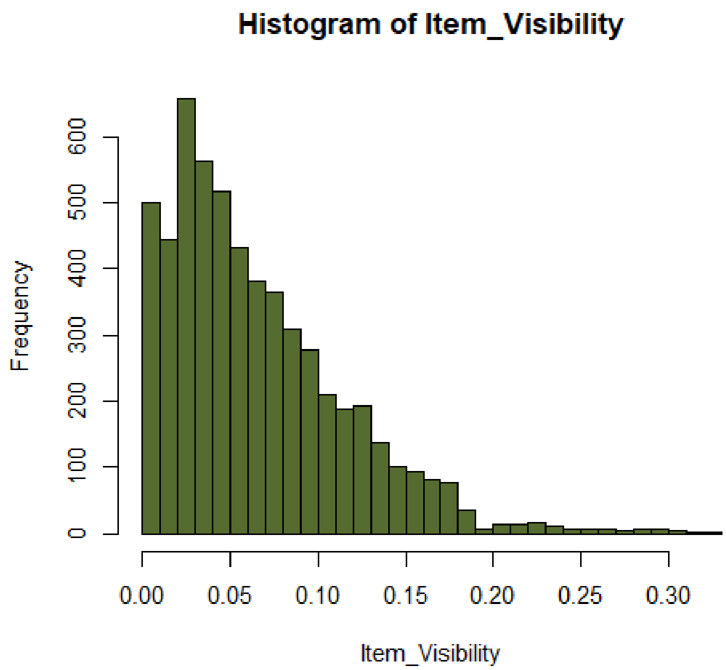
```
geom_point(aes(color = Item_Fat_Content), size = 3) +
ggtitle("Dumbbell Plot of snacks type & range
    of MRP in different Fat Content")+
scale_color_brewer(palette = "Set1", direction = -1) +
theme(legend.position = "bottom")
```



- *ggplot(mydata1, aes(x = MRP, y = Item_Type)): This line initializes a plot using the ggplot() function. It specifies the data frame (mydata1) and maps the x-axis to the range of "Item_MRP" (MRP) and the y-axis to "Item_Type".*
- *geom_line(): This adds lines to the plot, connecting points along the x-axis (range of "Item_MRP") for each "Item_Type". This effectively creates a dumbbell plot where each line represents the range of "Item_MRP" for a specific "Item_Type".*
- *geom_point(aes(color = Item_Fat_Content), size = 3): This adds points to the plot representing each "Item_Type" at the respective minimum and maximum values of "Item_MRP" (dumbbell ends). The color of the points is mapped to "Item_Fat_Content", and the size of the points is set to 3.*
- *ggtitle("Dumbbell Plot of snacks type & range of MRP in different Fat Content"): This line sets the title of the plot to "Dumbbell Plot of snacks type & range of MRP in different Fat Content".*
- *scale_color_brewer(palette = "Set1", direction = -1): This line sets the color palette for the points. Here, the "Set1" palette is used with a direction of -1 to reverse the color order.*
- *theme(legend.position = "bottom"): This line sets the position of the legend to the bottom of the plot.*

 # create a histogram of the "Item_Visibility" variable from your dataset using the "Scott" method to determine the number of bins,

```
hist(Item_Visibility, breaks = "Scott",col="darkolivegreen")
```

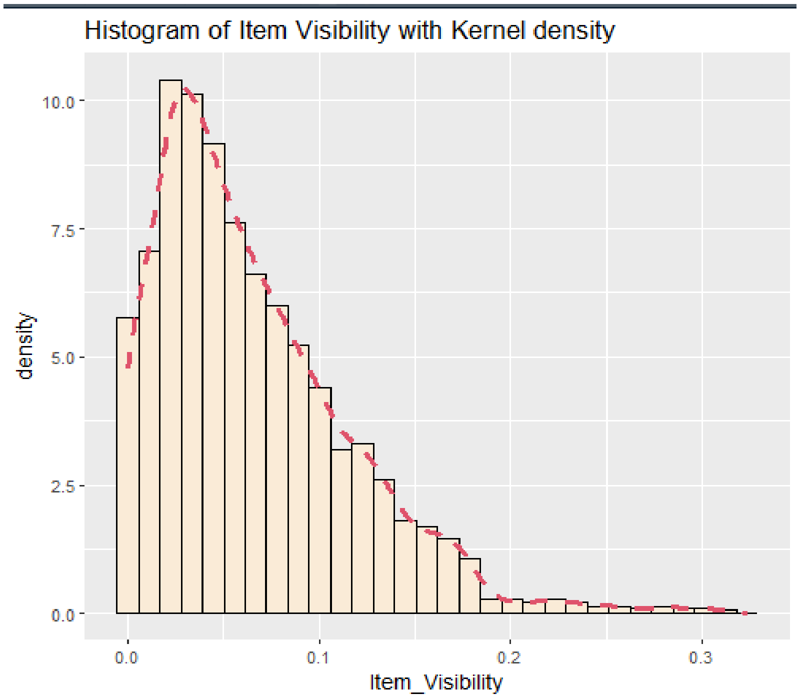


- *hist(Item_Visibility, breaks = "Scott", col = "darkolivegreen"): This line creates a histogram of the "Item_Visibility" variable using the hist() function.*
- *Item_Visibility: This specifies the variable for which the histogram will be created.*
- *breaks = "Scott": This specifies the method used to determine the number of bins in the histogram. The "Scott" method is a data-driven approach that calculates the number of bins based on the sample size and variance of the data.*
- *col = "darkolivegreen": This sets the color of the bars in the histogram to "darkolivegreen".*

- *The histogram shows that 0.03 has the highest frequency of 640. Whereas the data is positively skewed, indicating that the majority of the data points are concentrated on the left side of the graph. As the distribution moves towards the right, the frequency of occurrence decreases, resulting in a lower density of data points in the higher value range. In a positively skewed distribution, the mean is typically greater than the median, which is greater than the mode. It often occurs in datasets where there is a natural lower boundary but no upper boundary like in Item Visibility case*

 # Create a histogram with a kernel density plot overlaid for the "Item_Visibility" variable in your dataset using the `ggplot2` package.

```
ggplot(mydata, aes(x = Item_Visibility)) +
  geom_histogram(aes(y = ..density..),
          colour = 1, fill = "antiquewhite") +
  geom_density(lwd = 1.2,
         linetype = 2,
         colour = 2)+
  ggtitle("Histogram of Item Visibility with Kernel density")
```



- *ggplot(mydata, aes(x = Item_Visibility)): This initializes a plot using the ggplot() function. It specifies the data frame (mydata) and maps the x-axis to the "Item_Visibility" variable.*
- *geom_histogram(aes(y = ..density..), colour = 1, fill = "antiquewhite"): This adds a histogram to the plot using the geom_histogram() function. The aes(y = ..density..) argument specifies that the histogram should be plotted using density values rather than counts.*
- *colour = 1: This sets the color of the outline of the histogram bars to black (color code 1).*
- *fill = "antiquewhite": This sets the fill color of the histogram bars to "antiquewhite", giving them an antique white color.*
- *geom_density(lwd = 1.2, linetype = 2, colour = 2): This adds a kernel density plot to the plot using the geom_density() function.*
- *lwd = 1.2: This sets the line width of the density plot to 1.2.*
- *linetype = 2: This sets the line type of the density plot to dashed.*
- *colour = 2: This sets the color of the density plot to red (color code 2)*
- *ggtitle("Histogram of Item Visibility with Kernel density"): This line sets the title of the plot to "Histogram of Item Visibility with Kernel density".*

The kernel density plot overlaid on the histogram provides a smoothed estimate of the probability density function of the "Item_Visibility" variable.
The combination of histogram and kernel density plot allows for a comprehensive understanding of the distribution of "Item_Visibility" in the dataset, highlighting any peaks, troughs, or patterns present in the data.

 # Calculate the average item weight (Item_Weight) for each unique value of Outlet_Establishment_Year in the dataset

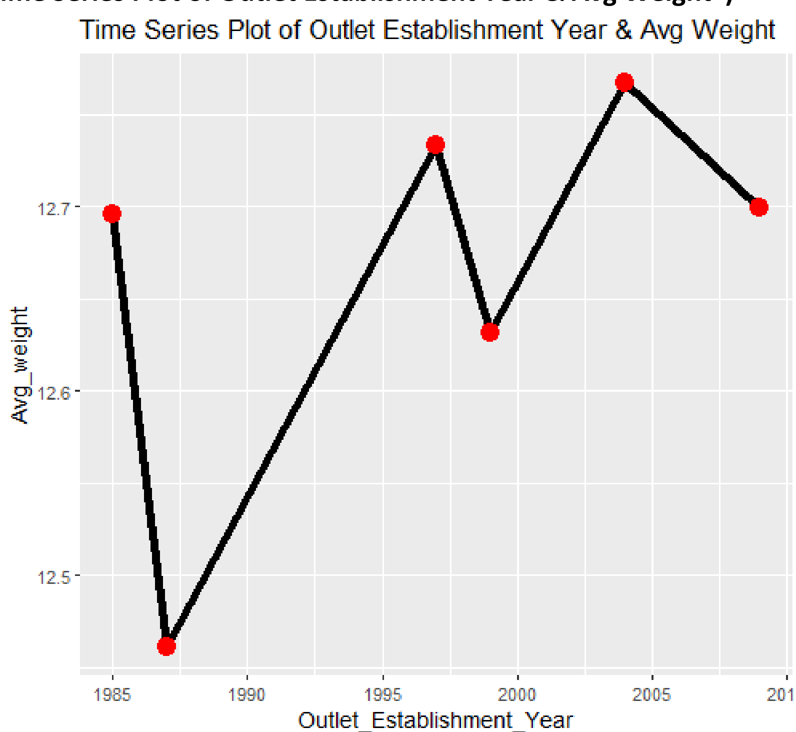
```
data_ffff<-data%>%
        group_by(Outlet_Establishment_Year)%>%
        summarise(Avg_weight=mean(Item_Weight))
       data_ffff
```

```
# A tibble: 6 × 2
  Outlet_Establishment_Year Avg_weight
                      <int>      <dbl>
1                      1985       12.7
2                      1987       12.5
3                      1997       12.7
4                      1999       12.6
5                      2004       12.8
6                      2009       12.7
```

- *group_by(Outlet_Establishment_Year): This line groups the data by the Outlet_Establishment_Year variable. This means that subsequent operations will be applied separately for each unique year.*
- *summarise(Avg_weight = mean(Item_Weight)): This line calculates the mean (average) item weight for each group defined by the Outlet_Establishment_Year. The mean() function calculates the average value of the Item_Weight variable within each group.*
- *The result of this operation is a new data frame with two columns: Outlet_Establishment_Year and Avg_weight, where Avg_weight contains the average item weight for each year.*

 # Create a time series plot using the `ggplot2` package to visualize the relationship between the average item weight (`Avg_weight`) and the outlet establishment year (`Outlet_Establishment_Year`)

```
ggplot(data_ffff, aes(x = Outlet_Establishment_Year, y = Avg_weight)) +
        geom_line(cex=2)+
        geom_point(col="red",size=4)+
        ggtitle("Time Series Plot of Outlet Establishment Year & Avg Weight")
```

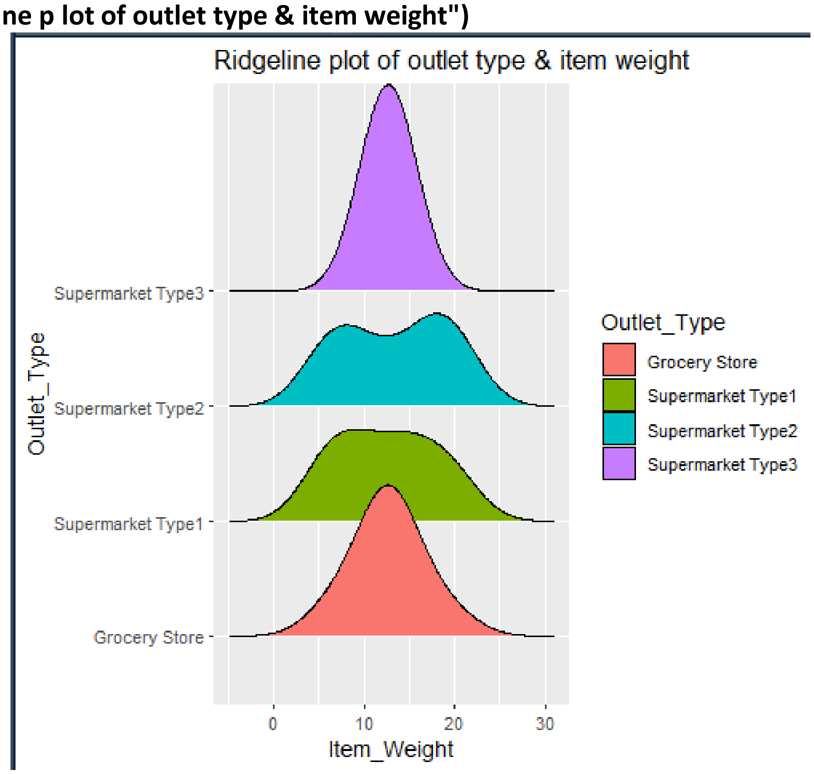


- *ggplot(data_ffff, aes(x = Outlet_Establishment_Year, y = Avg_weight)): This initializes a plot using the ggplot() function. It specifies the data frame (data_ffff) and maps the x-axis to the Outlet_Establishment_Year variable and the y-axis to the Avg_weight variable.*

- *geom_line(cex = 2): This adds a line to the plot using the geom_line() function. It connects the points representing the average item weight across different establishment years.*
- *cex = 2: This parameter sets the size of the line to be larger (size 2).*
- *geom_point(col = "red", size = 4): This adds points to the plot using the geom_point() function. Each point represents the average item weight for a specific establishment year.*
- *col = "red": This sets the color of the points to red.*
- *size = 4: This sets the size of the points to 4.*
- *ggtitle("Time Series Plot of Outlet Establishment Year & Avg Weight"): This line sets the title of the plot to "Time Series Plot of Outlet Establishment Year & Avg Weight".*
- *The time series plot visualizes how the average item weight (Avg_weight) changes over different outlet establishment years (Outlet_Establishment_Year).*
- *Each point on the plot represents the average item weight for a specific establishment year, and the line connecting these points shows the trend or pattern of change over time.*
- *This Time series graph visualizes that was around 12.7 in 1985 but plummeted sharply over the years reaching all time low of 12.4 but that surged to 12. 78, highest was in the year 2004 around 12.85*

 # Create a ridgeline plot to visualize the distribution of top 100 "Item_Weight" across different types of outlets ("Outlet_Type") in a dataset using the `ggridges` package.

```
install.packages("ggridges")
  library(ggridges)

dataf <- mydata[1:100, c("Outlet_Type", "Item_Weight")]
  ggplot(dataf, aes(x = Item_Weight, y =Outlet_Type, fill = Outlet_Type)) +
    geom_density_ridges()+
    ggtitle("Ridgeline p lot of outlet type & item weight")
```



- *install.packages("ggridges"): This line installs the ggridges package if it is not already installed.*
- *library(ggridges): This line loads the ggridges package, enabling the use of its functions for creating ridgeline plots.*
- *dataf <- mydata[1:100, c("Outlet_Type", "Item_Weight")]: This line subsets the dataset mydata to select the first 100 rows and only the columns "Outlet_Type" and "Item_Weight". This subset of data is stored in a new dataframe called dataf.*

- *ggplot(dataf, aes(x = Item_Weight, y = Outlet_Type, fill = Outlet_Type)): This line initializes a plot using the ggplot() function. It specifies the data frame (dataf) and maps the x-axis to "Item_Weight", the y-axis to "Outlet_Type", and the fill aesthetic to "Outlet_Type".*
- *geom_density_ridges(): This adds ridgelines to the plot using the geom_density_ridges() function. Ridgeline plots are essentially density plots stacked on top of each other for different groups (in this case, different outlet types).*
- *The ridgeline plot visualizes the distribution of "Item_Weight" for each type of outlet ("Outlet_Type") in the dataset.*
- *Each ridgeline represents the density of "Item_Weight" values for a specific outlet type, with higher peaks indicating regions of higher density.*
- *The plot allows for easy comparison of the distributions of "Item_Weight" across different outlet types, revealing any differences or similarities in the distribution patterns.*
- *Supermarket type 3 has the highest peak centered at 10kg followed by grocery store summarizing that they have weight concentrating around 10 kg mostly while Supermarket type1 & 2 have relatively flat shape indicating the greater variability in the variable.*

 # Create a new data frame called `df3` containing two variables, "item_visibility" and "outlet_size", extracted from the original dataset.

```
df3<-data.frame(item_visibility=data$Item_Visibility,
            outlet_size=data$Outlet_Size)
df3<-df3[1:200,]
```

```
   item_visibility outlet_size
1      0.007564836      Medium
2      0.118599314      Medium
3      0.063817206       Small
4      0.082601537      Medium
5      0.015782495      Medium
6      0.171079215       Small
7      0.092737611      Medium
8      0.021206464        High
9      0.079450700      Medium
10     0.037980963      Medium
11     0.028184344      Medium
12     0.109920138        High
13     0.182619235       Small
14     0.065630844       Small
15     0.027447057       Small
16     0.035178935       Small
```

- *f3 <- data.frame(item_visibility = data$Item_Visibility, outlet_size = data$Outlet_Size): This line creates a new data frame called df3.*
- *item_visibility = data$Item_Visibility: This assigns the "Item_Visibility" variable from the original dataset data to a column named "item_visibility" in the new data frame df3.*
- *outlet_size = data$Outlet_Size: This assigns the "Outlet_Size" variable from the original dataset data to a column named "outlet_size" in the new data frame df3.*
- *df3 <- df3[1:200,]: This line subsets the df3 data frame to include only the first 200 rows. The comma after the row index indicates that all columns are retained.*

 # make a table of location type and outlet size

```
table_data <- data %>%
              group_by(Outlet_Location_Type, Outlet_Size) %>%
              summarise(mean_MRP = mean(Item_MRP))
          # Pivot the table
              table_data_pivot <- table_data %>%
```

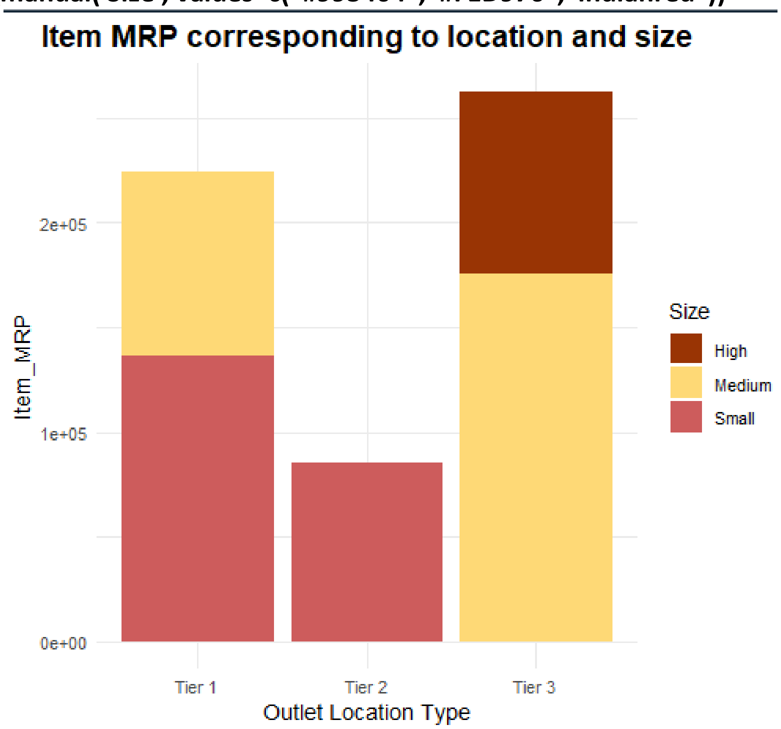
```
pivot_wider(names_from = Outlet_Size, values_from = mean_MRP)
# Print the table
print(table_data_pivot)
```

```
  Outlet_Location_Type Medium Small  High
  <chr>                 <dbl> <dbl> <dbl>
1 Tier 1                 142.  140.    NA
2 Tier 2                   NA  138.    NA
3 Tier 3                 141.    NA  141.
```

- ***table_data <- data %>% group_by(Outlet_Location_Type, Outlet_Size) %>% summarise(mean_MRP = mean(Item_MRP)): This line groups the data by two variables, "Outlet_Location_Type" and "Outlet_Size". Then, it calculates the mean of "Item_MRP" within each group. The result is stored in a data frame called table_data.***
- ***table_data_pivot <- table_data %>% pivot_wider(names_from = Outlet_Size, values_from = mean_MRP): This line reshapes the summarized data frame (table_data). It spreads the data from the "Outlet_Size" column into separate columns, with each unique value of "Outlet_Size" becoming a column header. The mean MRP values are filled into these columns accordingly.***
- ***print(table_data_pivot): This line prints the pivoted table, showing the mean MRP values for each combination of "Outlet_Location_Type" and "Outlet_Size". The table is displayed in a wide format, with separate columns for different outlet sizes.***
- ***This pivot table shows that in Tier 1 there are only Medium and small outlet size whereas tier 2 only comprises of small enterprises. Tier 3 has medium & high enterprises only***

 # Create a stacked bar plot using `ggplot2` to visualize the distribution of "Item_MRP" across different combinations of "Outlet_Location_Type" and "Outlet_Size" in the dataset.

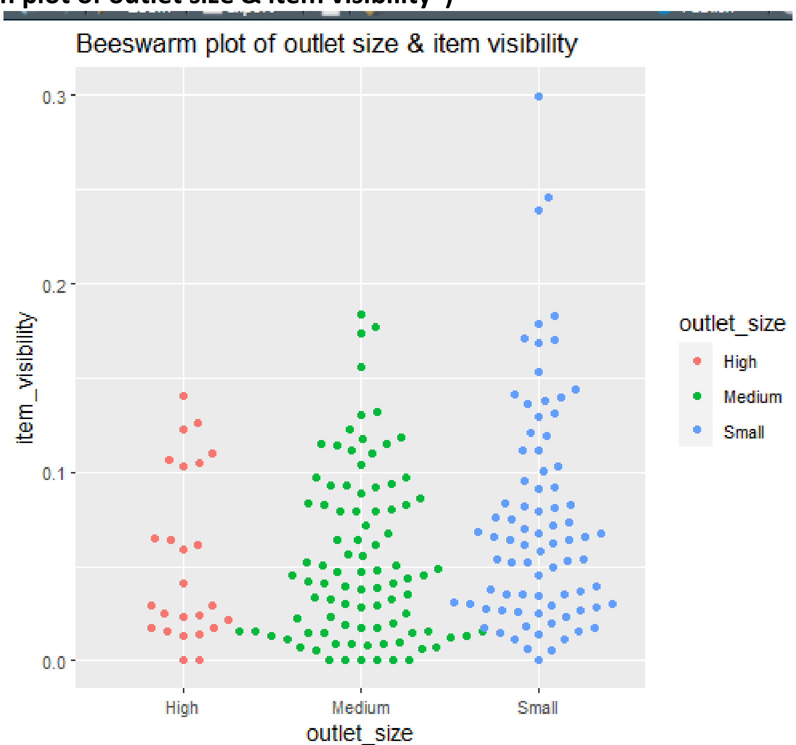
```
ggplot(data, aes(fill=Outlet_Size, y=Item_MRP, x=Outlet_Location_Type)) +
          geom_bar(position='stack', stat='identity') +
          theme_minimal() +
          labs(x='Outlet Location Type', y='Item_MRP', title='Item MRP corresponding to location and
size') +
          theme(plot.title = element_text(hjust=0.5, size=15, face='bold')) +
          scale_fill_manual('Size', values=c("#993404","#FED976","indianred"))
```



- *ggplot(data, aes(fill = Outlet_Size, y = Item_MRP, x = Outlet_Location_Type)): This initializes a plot using the ggplot() function. It specifies the data frame (data) and maps the fill aesthetic to "Outlet_Size", the y-axis to "Item_MRP", and the x-axis to "Outlet_Location_Type".*
- *geom_bar(position = 'stack', stat = 'identity'): This adds a stacked bar plot to the plot using the geom_bar() function. It stacks the bars for different "Outlet_Size" categories on top of each other, with the height of each stack corresponding to the total "Item_MRP" for that combination of "Outlet_Location_Type" and "Outlet_Size".*
- *heme_minimal(): This sets the theme of the plot to a minimalistic style.*
- *labs(x = 'Outlet Location Type', y = 'Item_MRP', title = 'Item MRP corresponding to location and size'): This sets the labels for the x-axis, y-axis, and plot title.*
- *theme(plot.title = element_text(hjust = 0.5, size = 15, face = 'bold')): This adjusts the title alignment, size, and font weight.*
- *Scale_fill_manual('Size', values = c("#993404", "#FED976", "indianred")): This sets the fill colors manually for the "Outlet_Size" categories. Each color corresponds to a specific "Outlet_Size" category, enhancing visual differentiation in the plot.*
- *This stacked bar plot shows none of the tiers has all the 3 size available, tier 2 only has small enterprises in them. The maximum proportion is of medium enterprises*

 # Create a beeswarm plot using the `ggbeeswarm` package to visualize the distribution of "item_visibility" across different categories of "outlet_size".

```
install.packages("ggbeeswarm")
library(ggbeeswarm)

# Basic beeswarm plot in ggplot2
ggplot(df3, aes(x = outlet_size, y = item_visibility,color=outlet_size)) +
  geom_beeswarm(cex=3)+
  ggtitle("Beeswarm plot of outlet size & item visibility")
```



- *install.packages("ggbeeswarm"): This line installs the ggbeeswarm package if it is not already installed.*
- *library(ggbeeswarm): This line loads the ggbeeswarm package, enabling the use of its functions for creating beeswarm plots.*
- *ggplot(df3, aes(x = outlet_size, y = item_visibility, color = outlet_size)): This line initializes a plot using the ggplot() function. It specifies the data frame (df3) and maps the x-axis to "outlet_size", the y-axis to "item_visibility", and the color aesthetic to "outlet_size".*

- *geom_beeswarm(cex = 3): This adds a beeswarm plot to the plot using the geom_beeswarm() function. Beeswarm plots display individual data points along the x-axis without overlapping, providing a clear visualization of the distribution*
- *ggtitle("Beeswarm plot of outlet size & item visibility"): This line sets the title of the plot to "Beeswarm plot of outlet size & item visibility".*
- *This graph shows that the highest concentration of item visibility is around 0.0-0.1 and medium enterprise has the maximum amount of values in general for visibility*

 # calculate the average MRP (Maximum Retail Price) of items for each unique value of "Outlet_Establishment_Year" in the dataset

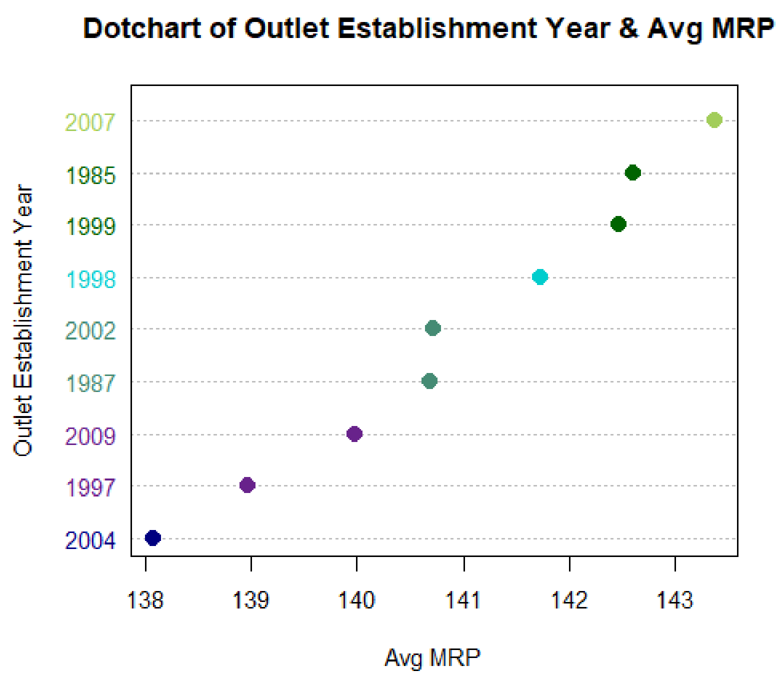
```
mydata<-mydata%>%
  group_by(Outlet_Establishment_Year)%>%
  summarize(Avg_MRP=mean(Item_MRP))
mydata
```

```
# A tibble: 9 × 2
  Outlet_Establishment_Year Avg_MRP
                      <int>   <dbl>
1                      1985    143.
2                      1987    141.
3                      1997    139.
4                      1998    142.
5                      1999    142.
6                      2002    141.
7                      2004    138.
8                      2007    143.
9                      2009    140.
>
```

- *mydata <- mydata %>% group_by(Outlet_Establishment_Year): This line groups the data by the variable "Outlet_Establishment_Year". This means that subsequent operations will be applied separately for each unique year.*
- *summarize(Avg_MRP = mean(Item_MRP)): This line calculates the mean (average) MRP of items within each group defined by "Outlet_Establishment_Year". The mean() function calculates the average value of the "Item_MRP" variable within each group.*
- *The result of this operation is a new data frame called mydata with two columns: "Outlet_Establishment_Year" and "Avg_MRP", where "Avg_MRP" contains the average*

 # Create a dot chart to visualize the relationship between the average MRP (Maximum Retail Price) and the outlet establishment years.

```
cols c(rep("navy",1),rep("darkorchid4",2), rep("aquamarine4", 2),
     rep("cyan3", 1),rep("darkgreen",2),rep("darkolivegreen3", 1),
     rep("gold3",1))
     dotchart(sort(mydata$Avg_MRP),
          labels = mydata$Outlet_Establishment_Year[order(mydata$Avg_MRP)],
          pch = 19, pt.cex = 1.5,
          xlab = "Avg MRP",ylab="Outlet Establishment Year",
          main = "Dotchart of Outlet Establishment Year & Avg MRP ",
          groups = rev(mydata$Outlet_Establishment_Year),
          color = cols)
```



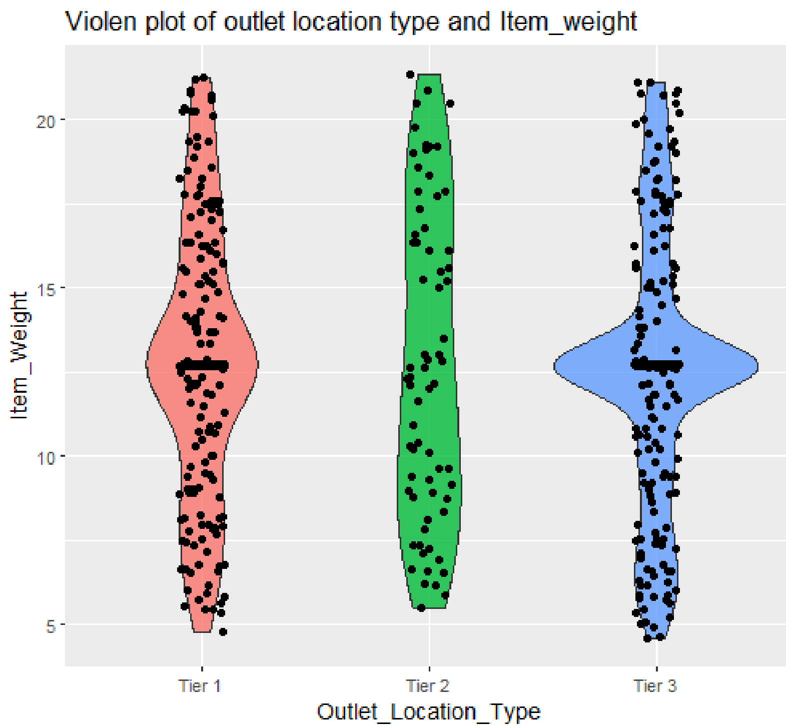
- *cols: This variable defines a vector containing colors corresponding to each unique value of "Outlet_Establishment_Year". Each color is repeated a specific number of times corresponding to the frequency of the corresponding establishment year in the dataset.007*
- *dotchart(): This function creates the dot chart.*
- *sort(mydata$Avg_MRP): This sorts the average MRP values in ascending order.*
- *labels = mydata$Outlet_Establishment_Year[order(mydata$Avg_MRP)]: This specifies the labels for the dots, arranged in the order of sorted average MRP values.*
- *pch = 19, pt.cex = 1.5: These parameters control the appearance of the dots. pch = 19 sets the dot shape to solid circles, and pt.cex = 1.5 sets the size of the dots.*
- *xlab = "Avg MRP", ylab = "Outlet Establishment Year": These set the labels for the x-axis and y-axis, respectively.*
- *main = "Dotchart of Outlet Establishment Year & Avg MRP ": This sets the main title of the plot.*
- *groups = rev(mydata$Outlet_Establishment_Year): This parameter specifies the groups (outlet establishment years) to which the dots belong. It's reversed to ensure that the legend matches the order of the dots.*
- *color = cols: This assigns the colors defined in the cols vector to the dots based on their respective outlet establishment years.*
- *The dot chart visually represents the average MRP for each outlet establishment year.*
- *Each dot represents an establishment year, positioned along the y-axis, with its position determined by the corresponding average MRP value on the x-axis.*
- *The colors of the dots are determined by the establishment years, facilitating easy identification of each year in the chart.*

- *2007 has the highest Avg MRP followed by 1985 and lowest in 2004*

 # generate a violin plot with jittered points to visualize the distribution of top 500 "Item_Weight" across different categories of "Outlet_Location_Type".

```
ggplot(data_500, aes(x = Outlet_Location_Type, y = Item_Weight, fill =Outlet_Location_Type  ))+
   geom_violin(alpha = 0.8) +
  geom_point(position = position_jitter(seed = 1, width = 0.1))+
   ggtitle("Violen plot of outlet location type and Item_weight")+
```

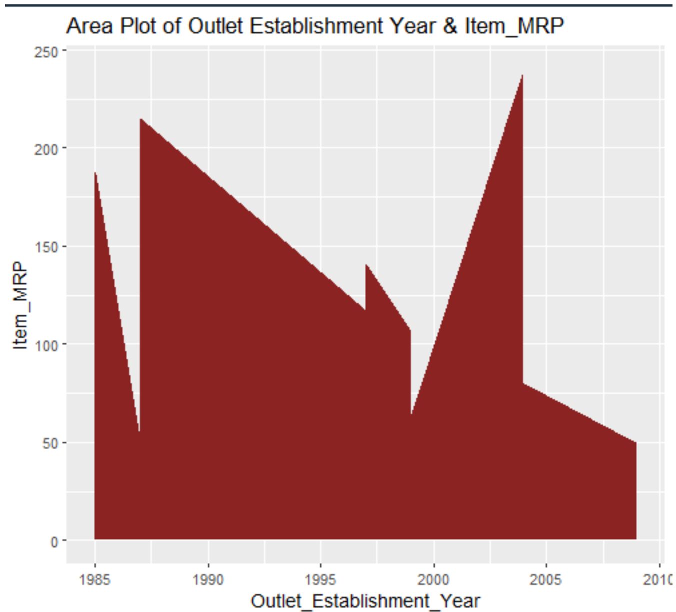
```
theme(legend.position = "none")
```



- *ggplot(data_500, aes(x = Outlet_Location_Type, y = Item_Weight, fill = Outlet_Location_Type)): This initializes a plot using the ggplot() function. It specifies the data frame (data_500) and maps the x-axis to "Outlet_Location_Type", the y-axis to "Item_Weight", and the fill aesthetic to "Outlet_Location_Type".*
- *geom_violin(alpha = 0.8): This adds a violin plot to the plot using the geom_violin() function. Violin plots display the distribution of the data by showing its density along the y-axis, with wider sections indicating higher density.*
- *alpha = 0.8: This sets the transparency of the violin plots to 80%, making them partially transparent.*
- *geom_point(position = position_jitter(seed = 1, width = 0.1)): This adds jittered points to the plot using the geom_point() function. Jittering is used to prevent overplotting by randomly adjusting the position of points along the x-axis.*
- *position = position_jitter(seed = 1, width = 0.1): This specifies the jittering parameters, including the random seed (seed = 1) for reproducibility and the width of the jittering (width = 0.1).*
- *ggtitle("Violin plot of outlet location type and Item_weight"): This sets the title of the plot.*
- *theme(legend.position = "none"): This removes the legend from the plot, as the fill aesthetic is mapped to the same variable used for the x-axis, which is redundant in this case.*
- *Tier 2 has relatively lower variation in item weight as corresponding to other 2*

 # create an area plot to visualize the relationship between "Outlet_Establishment_Year" and "Item_MRP".

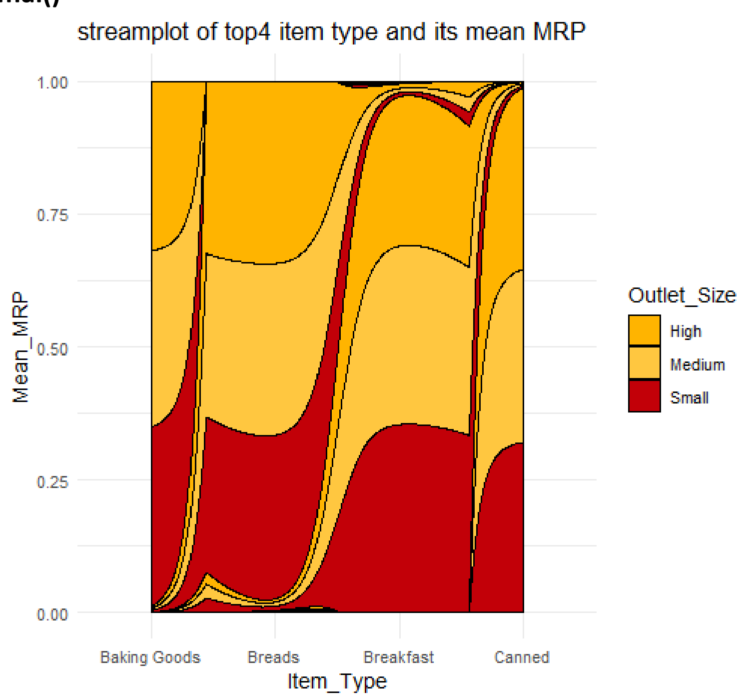
```
ggplot(data, aes(x =Outlet_Establishment_Year, y = Item_MRP)) +
        geom_area(fill = "brown4", alpha = 1)+
        ggtitle("Area Plot of Outlet Establishment Year & Item_MRP")
```



- *ggplot(data, aes(x = Outlet_Establishment_Year, y = Item_MRP)): This initializes a plot using the ggplot() function. It specifies the data frame (data) and maps the x-axis to "Outlet_Establishment_Year" and the y-axis to "Item_MRP". geom_area(fill = "brown4", alpha = 1): This adds an area plot to the plot using the geom_area() function. Area plots fill the area under the curve between the x-axis and the line representing the data. The fill parameter sets the fill color of the area, and alpha controls the transparency of the fill color (1 means fully opaque).*
- *ggtitle("Area Plot of Outlet Establishment Year & Item_MRP"): This sets the title of the plot.*
- *The MRP peaked during to maximum around 2003-2004 then slipped down till 50 once during 2008*

 # create a streamplot to visualize the relationship between "Item_Type", "Mean_MRP", and "Outlet_Size" using the `geom_stream()`

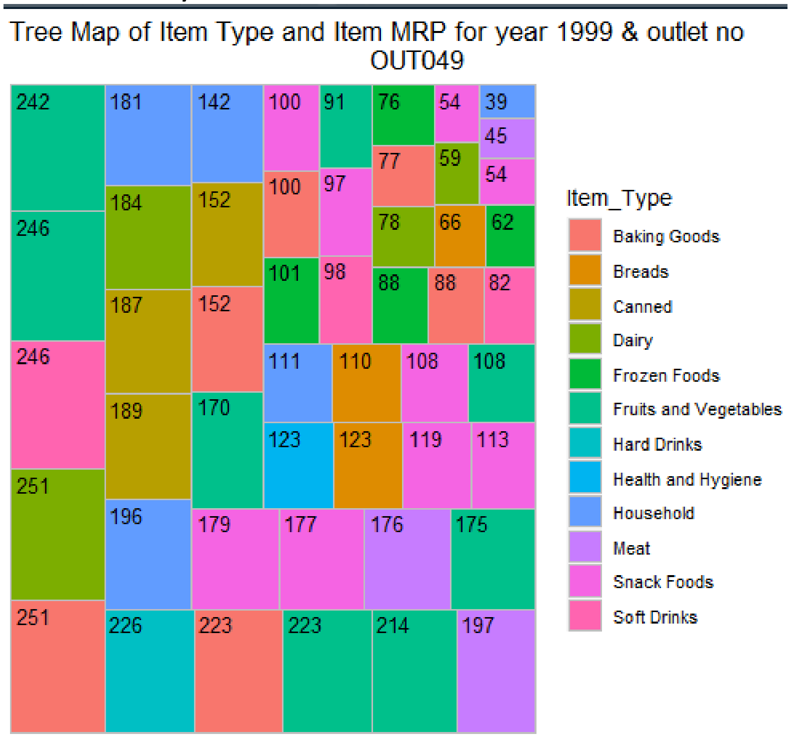
```
ggplot(data_frame_11, aes(x =Item_Type , y = Mean_MRP,fill=Outlet_Size)) +
        geom_stream(type="proportional",color = 1, lwd = 0.25,
                bw = 1) +
        scale_fill_manual(values = cols)+
        ggtitle("streamplot of top4 item type and its mean MRP")+
        theme_minimal()
```



- *ggplot(data_frame_11, aes(x = Item_Type, y = Mean_MRP, fill = Outlet_Size)): This initializes a plot using the ggplot() function. It specifies the data frame (data_frame_11) and maps the x-axis to "Item_Type", the y-axis to "Mean_MRP", and the fill aesthetic to "Outlet_Size".*
- *geom_stream(type = "proportional", color = 1, lwd = 0.25, bw = 1): This adds a streamplot to the plot using the geom_stream() function from the ggstream package. Streamplots represent the flow of data over categories. The type = "proportional" parameter indicates that the width of the streamlines will be proportional to the number of observations in each category. Parameters like color, lwd, and bw control the appearance of the streamlines.*
- *scale_fill_manual(values = cols): This sets the fill colors manually for the "Outlet_Size" categories using the colors defined in the cols vector.*
- *ggtitle("Streamplot of top 4 item types and their mean MRP"): This sets the title of the plot.*
- *theme_minimal(): This sets the plot theme to minimalistic style.*
- *This graphs shows that there is no set pattern in avg MRP some values high size outlet has mean MRP low for some year meanwhile some small size outlet has mean MRP very high for some year*

 # perform data manipulation and visualization tasks using the `dplyr`, `treemapify`, and `ggplot2` packages.

```
ru <- data %>%
                    filter( Outlet_Identifier== "OUT049" & Outlet_Establishment_Year == 1999)
                  ru<- ru[1:50,]
                  Item_MRP1<-ceiling(ru$Item_MRP)

                  install.packages("treemapify")
                  library(treemapify)
                  ggplot(ru,aes(area = Item_MRP,fill = Item_Type,label =Item_MRP1))+
                    geom_treemap()+
                    geom_treemap_text(size=10)+
                    ggtitle("Tree Map of Item Type and Item MRP for year 1999 & outlet no
                        OUT049")
```



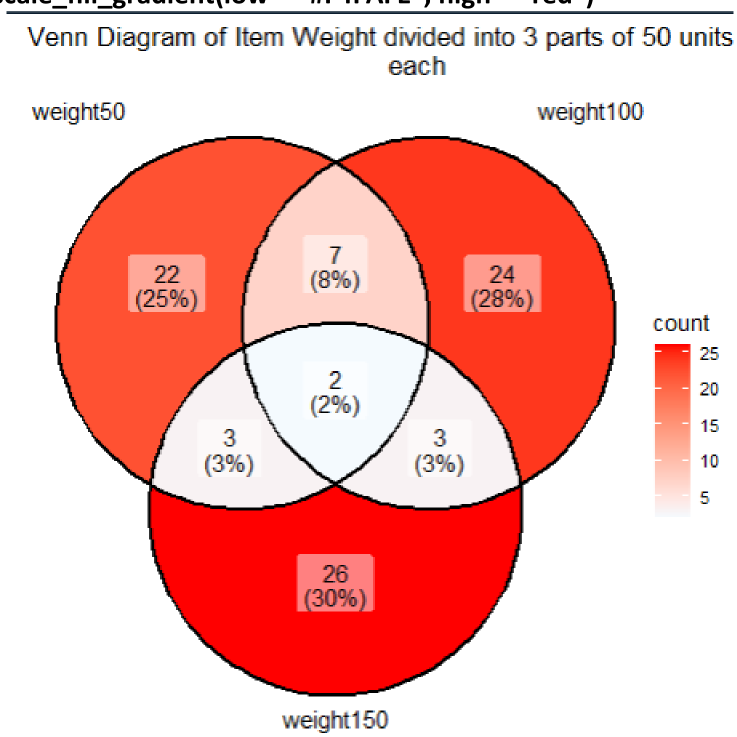
- *ru <- data %>% filter(Outlet_Identifier == "OUT049" & Outlet_Establishment_Year == 1999): This line filters the data stored in the dataframe data to only include rows where the Outlet*

Identifier is "OUT049" and the Outlet Establishment Year is 1999. The filtered data is stored in the dataframe ru.

- *ru <- ru[1:50,]: This line subsets the ru dataframe to include only the first 50 rows.*
- *Item_MRP1 <- ceiling(ru$Item_MRP): This line calculates the ceiling (rounds up) of the Item MRP (Maximum Retail Price) values in the ru dataframe and stores the result in a new variable Item_MRP1.*
- *install.packages("treemapify"): This line installs the "treemapify" package if it's not already installed.*
- *library(treemapify): This line loads the "treemapify" package into the R environment, enabling the use of its functions.*
- *ggplot(ru, aes(area = Item_MRP, fill = Item_Type, label = Item_MRP1)) + geom_treemap() + geom_treemap_text(size = 10) + ggtitle("Tree Map of Item Type and Item MRP for year 1999 & outlet no OUT049"): This line creates a treemap visualization using ggplot2. It maps the Item_MRP values to the area of each rectangle, colors the rectangles based on Item_Type, and labels each rectangle with rounded Item_MRP values (Item_MRP1). The geom_treemap() function is used to create the treemap, and geom_treemap_text() is used to add text labels to each rectangle. Finally, ggtitle() sets the title of the plot to "Tree Map of Item Type and Item MRP for year 1999 & outlet no OUT049".*
- *Here Fruits & Vegetables have the highest Item MRP*

 # utilize the `ggVennDiagram` package to create a Venn diagram visualizing overlaps between sets of data.

```
install.packages("ggVennDiagram")
                        library(ggVennDiagram)
# List of items
                        x <- list(weight50=data$Item_Weight[1:50],weight100=
data$Item_Weight[51:100],weight150= data$Item_Weight[101:150])

                        # 3D Venn diagram
                        ggVennDiagram(x,color = "black", lwd = 0.8, lty = 1)+
                         ggtitle("Venn Diagram of Item Weight divided into 3 parts of 50 units
                               each")+
                         scale_fill_gradient(low = "#F4FAFE", high = "red")
```



- *install.packages("ggVennDiagram"): This line installs the "ggVennDiagram" package if it's not already installed in the R environment.*
- *library(ggVennDiagram): This line loads the "ggVennDiagram" package, enabling the use of its functions.*
- *x <- list(weight50=data$Item_Weight[1:50], weight100= data$Item_Weight[51:100], weight150= data$Item_Weight[101:150]): This line creates a list x containing three sets of data, each representing 50 units of item weight from the dataframe data. The sets are named weight50, weight100, and weight150.*
- *ggVennDiagram(x, color = "black", lwd = 0.8, lty = 1): This line generates a 3D Venn diagram using the data in the list x. Parameters such as color, lwd, and lty define the color, line width, and line type of the diagram, respectively.*
- *ggtitle("Venn Diagram of Item Weight divided into 3 parts of 50 units each"): This line sets the title of the Venn diagram to "Venn Diagram of Item Weight divided into 3 parts of 50 units each".*
- *scale_fill_gradient(low = "#F4FAFE", high = "red"): This line sets the color gradient for the Venn diagram. It maps the lowest value to a light blue color (#F4FAFE) and the highest value to red.*
- *Weight between 0- 50 & 51-100 have 7 values in common whereas weight between 100-150 & 50-100 and 0-50 have 3-3 entries in common. All the 3 sets have only 2 elements in common that is 12.9 kg & 13.8 kg*

 # generates a table that counts the occurrences of each unique value in the "Outlet_Identifier" column of the dataframe `data`.

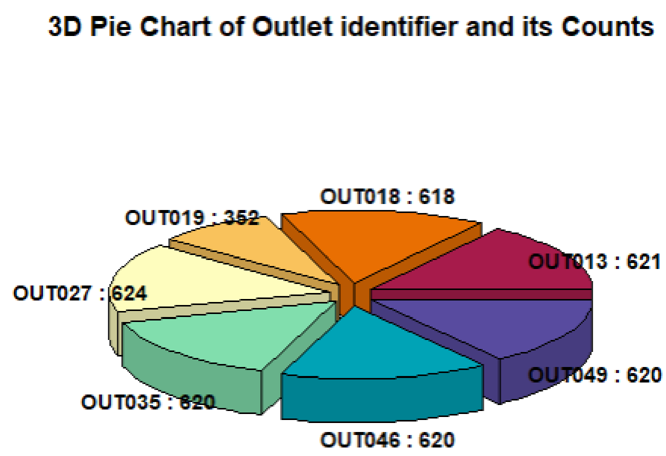
identifier_counts<-table(data$Outlet_Identifier)
identifier_counts

```
OUT013 OUT018 OUT019 OUT027 OUT035 OUT046 OUT049
   621    618    352    624    620    620    620
>
```

- *table(data$Outlet_Identifier): This part of the code creates a table of counts for each unique value in the "Outlet_Identifier" column of the dataframe data. It counts how many times each unique value appears in the column.*
- *identifier_counts: This line assigns the table generated by table(data$Outlet_Identifier) to the variable identifier_counts. This variable will contain the counts of each unique value in the "Outlet_Identifier" column.*
- *identifier_counts: When you print or inspect identifier_counts, it will display the table of counts showing how many times each unique value occurs in the "Outlet_Identifier" column.*
- *Here OUT037 has the highest stores where OUT035,OUT046,OUT049 have the same values*

 # create a 3D pie chart visualization using the `pie3D()` function from the "plotrix" package.

library(plotrix)
pie3D(identifier_counts, explode=0.1,col = hcl.colors(length(identifier_counts), "Spectral"),
labels = paste(names(identifier_counts),":",identifier_counts), labelcex = 0.9, labelcol="black", main = "3D Pie Chart of Outlet identifier and its Counts")



- *ibrary(plotrix): This line loads the "plotrix" package, which provides various plotting functions, including pie3D() for creating 3D pie charts.*
- *pie3D(identifier_counts, explode = 0.1, col = hcl.colors(length(identifier_counts), "Spectral"), labels = paste(names(identifier_counts), ":", identifier_counts), labelcex = 0.9, labelcol = "black", main = "3D Pie Chart of Outlet identifier and its Counts"):*
- *identifier_counts: This argument specifies the data to be visualized in the pie chart. It likely contains the counts of each unique value in the "Outlet_Identifier" column, as generated by the table() function.*
- *explode = 0.1: This parameter determines how much each slice of the pie chart should be exploded (i.e., pulled out from the center). Here, 0.1 means a slight explosion for each slice.*
- *col = hcl.colors(length(identifier_counts), "Spectral"): This parameter specifies the colors to be used for each slice of the pie chart. It generates a sequence of colors based on the number of unique identifiers.*
- *labels = paste(names(identifier_counts), ":", identifier_counts): This parameter specifies the labels for each slice of the pie chart. It concatenates the names of the identifiers with their corresponding counts.*
- *labelcex = 0.9: This parameter sets the size of the labels in the pie chart. Here, 0.9 means the labels are 90% of their default size.*
- *labelcol = "black": This parameter sets the color of the labels to black.*
- *main = "3D Pie Chart of Outlet identifier and its Counts": This parameter specifies the main title of the pie chart.*

 # Make_long of Item_MRP

```
df_333 <- data %>%
                    make_long(Item_MRP,Item_Weight,Item_Visibility)
                   df_333
```

```
>                                          df_333
# A tibble: 12,225 × 4
   x                      node next_x            next_node
   <fct>                 <dbl> <fct>                 <dbl>
 1 Item_MRP           108.     Item_Weight        20.8
 2 Item_Weight         20.8    Item_Visibility     0.00756
 3 Item_Visibility      0.00756 NA                NA
 4 Item_MRP           234.     Item_Weight        12.7
 5 Item_Weight         12.7    Item_Visibility     0.119
 6 Item_Visibility      0.119  NA                 NA
 7 Item_MRP           117.     Item_Weight         9.8
 8 Item_Weight          9.8    Item_Visibility     0.0638
 9 Item_Visibility      0.0638 NA                 NA
10 Item_MRP            50.1    Item_Weight        19.4
# ℹ 12,215 more rows
# ℹ Use `print(n = ...)` to see more rows
```

- *data %>% make_long(Item_MRP, Item_Weight, Item_Visibility): This code is using the pipe operator %>%, which is part of the dplyr package, to perform data manipulation tasks in a sequence. The make_long() function is likely a custom function or one from another package (not from dplyr). It is applied to the data dataframe to convert it into a long format, where the variables Item_MRP, Item_Weight, and Item_Visibility are stacked into a single column. The resulting dataframe is assigned to df_333.*
- *df_333: This line simply prints the resulting dataframe df_333*

#Compute the total number of missing values in each column of the dataframe `df_333` and stores the result in the variable `df_4`

```
df_4<-colSums(is.na(df_333))
df_4
```

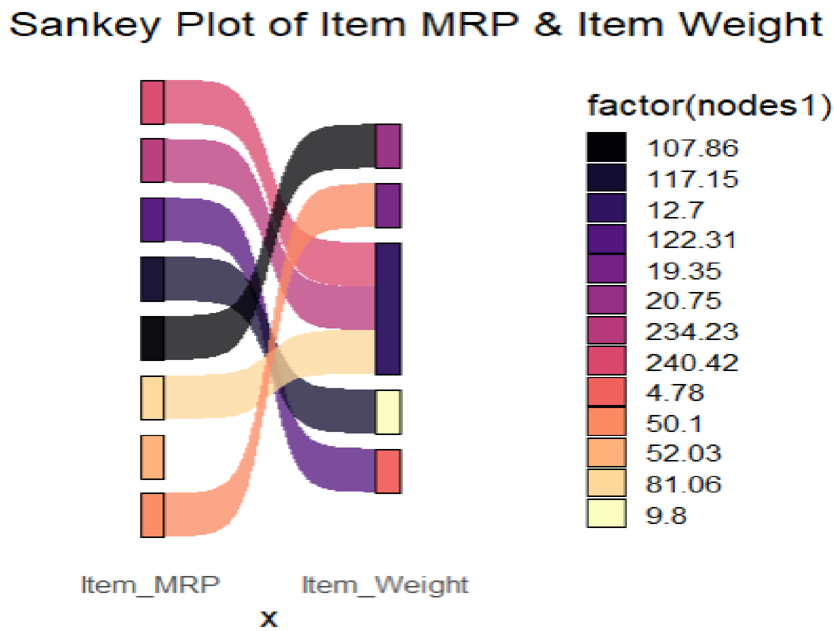
```
>                                       df_4
         x        node      next_x next_node
         0           0        4075      4075
```

- *is.na(df_333): This part of the code generates a logical matrix of the same dimensions as df_333, where each element is TRUE if the corresponding element in df_333 is NA (missing), and FALSE otherwise.*
- *colSums(): This function calculates the sum of the logical values for each column of the matrix generated by is.na(df_333). Since TRUE is treated as 1 and FALSE as 0 when you sum logical values, colSums() effectively counts the number of missing values in each column.*
- *This function shows that next_x, next_node has 4075 NA values, now we will omit them.*

 # Create a Sankey plot of item MRP & Item weight

```
df_333<-na.omit(df_333)
df_333<-df_333[1:15,]
nodes1<-round(df_333$node,2)
ggplot(df_333, aes(x =x ,
       next_x = next_x,
       node = node,
```

```
                next_node = next_node,
                fill = factor(nodes1),
                label=node)) +
       geom_sankey(flow.alpha = 0.75, node.color = 1) +
       scale_fill_viridis_d(option = "A", alpha = 0.95) +
       theme_sankey(base_size = 16)+
       ggtitle("Sankey Plot of Item_MRP & Item_weight")
```



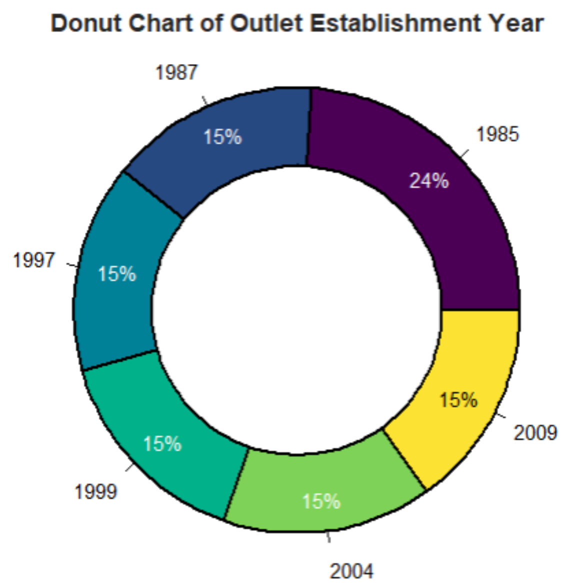
- *df_333 <- na.omit(df_333): This line removes rows with any missing values (NA) from the dataframe df_333. The resulting dataframe is stored back into df_333.*
- *df_333 <- df_333[1:15,]: This line subsets the dataframe df_333 to include only the first 15 rows. This might be done for visualization purposes, possibly to reduce the complexity of the plot or to focus on a specific subset of data.*
- *nodes1 <- round(df_333$node, 2): This line rounds the values in the "node" column of the dataframe df_333 to two decimal places and assigns the result to the variable nodes1. This variable might be used for coloring or labeling nodes in the Sankey plot.*
- *ggplot(df_333, aes(x = x, next_x = next_x, node = node, next_node = next_node, fill = factor(nodes1), label = node)) +: This line initializes a ggplot object with the dataframe df_333 as the data source. It sets up aesthetics for the plot, specifying the source (x), target (next_x), source node (node), target node (next_node), fill color (based on the factorized nodes1), and node label (label).*
- *geom_sankey(flow.alpha = 0.75, node.color = 1) +: This line adds a layer to the ggplot object for drawing the Sankey diagram. It specifies parameters such as the transparency of flow lines (flow.alpha) and the color of the nodes (node.color).*
- *scale_fill_viridis_d(option = "A", alpha = 0.95) +: This line sets the color scale for the nodes in the Sankey plot. It uses the Viridis color palette (viridis_d) with specified options and alpha value.*
- *theme_sankey(base_size = 16) +: This line applies a theme to the Sankey plot, adjusting the base font size.*
- *ggtitle("Sankey Plot of Item_MRP & Item_weight"): This line sets the title of the plot to "Sankey Plot of Item_MRP & Item_weight".*

 # Create a pie chart (or donut chart) of Outlet Establishment Year using a function called `PieChart()`.

```
PieChart(Outlet_Establishment_Year, data = data,
                    fill = "viridis",
                    color = "black", size=10,values_size = 1,values_color = c(rep("white",5 ), 1),
```

```
lwd = 2,
lty = 1,
main = "Donut Chart of Outlet Establishment Year")
```

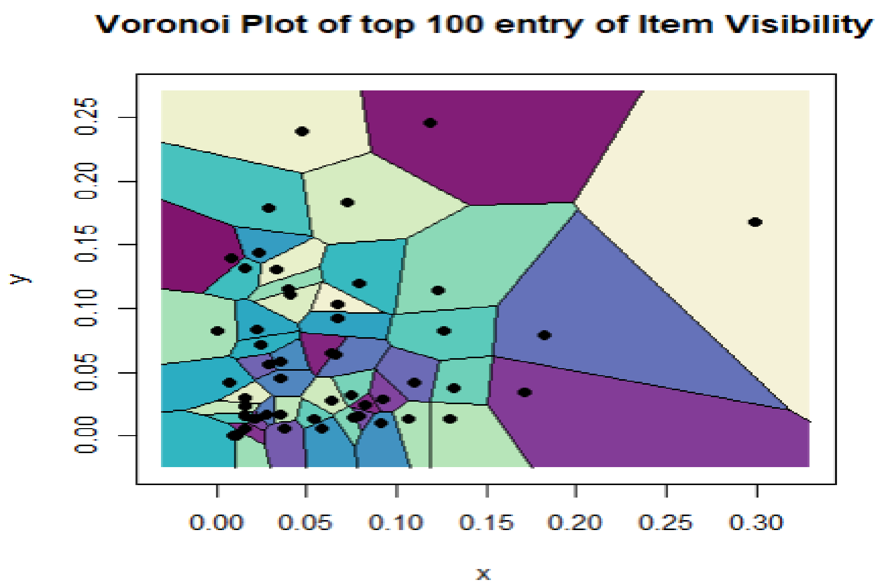
```
--- Outlet_Establishment_Year ---

                 1985   1987   1997   1999   2004   2009       Total
Frequencies:      976    621    620    620    620    618        4075
Proportions:    0.240  0.152  0.152  0.152  0.152  0.152       1.000

Chi-squared test of null hypothesis of equal probabilities
  Chisq = 155.686, df = 5, p-value = 0.000
```



- ***Outlet_Establishment_Year: This argument likely specifies the variable from the dataframe data that contains the data for the pie chart. It seems to represent the establishment year of outlets.***
- ***data = data: This argument specifies the dataframe data from which the data for the pie chart is taken.***
- ***fill = "viridis": This argument likely specifies the color palette to be used to fill the segments of the pie chart. It appears to be using the "viridis" color palette.***
- ***color = "black": This argument specifies the color of the lines separating the segments of the pie chart. It's set to black in this case.***
- ***size = 10: This argument seems unusual for a pie chart. It's possible it might control the size of the pie chart.***
- ***values_size = 1: This argument might control the size of the values displayed inside the segments of the pie chart.***
- ***values_color = c(rep("white",5), 1): This argument might specify the color of the values displayed inside the segments of the pie chart. It's set to white for the first 5 segments and 1 for the rest.***
- ***lwd = 2: This argument likely specifies the line width of the lines separating the segments of the pie chart.***
- ***lty = 1: This argument likely specifies the line type of the lines separating the segments of the pie chart.***
- ***main = "Donut Chart of Outlet Establishment Year": This argument specifies the main title of the pie chart, indicating it's a donut chart representing the establishment year of outlets.***
- **Here in donut chart year 1985 has the highest count while other have the same count**

#utilize the `deldir` package to generate a Voronoi plot based on the item visibility data from a dataframe `data`.

```
install.packages("deldir")
            library(deldir)
            x_visi <-data$Item_Visibility[1:50]
            y_visi <- data$Item_Visibility[51:100]
            # Calculate Voronoi Tesselation and tiles
            tesselation <- deldir(x_visi, y_visi)
            tiles <- tile.list(tesselation)
          plot(tiles, pch =19,
               fillcol = hcl.colors(50, "Purple-Yellow"),main="Voronoi Plot of top 100 entry of
Item Visibility")
```



- *install.packages("deldir"): This line installs the "deldir" package if it's not already installed in your R environment.*
- *library(deldir): This line loads the "deldir" package, enabling the use of its functions for computing Voronoi tessellations.*
- *x_visi <- data$Item_Visibility[1:50]: This line extracts the first 50 values of item visibility from the dataframe data and stores them in the variable x_visi.*
- *y_visi <- data$Item_Visibility[51:100]: This line extracts the next 50 values of item visibility (from 51st to 100th) from the dataframe data and stores them in the variable y_visi.*
- *tesselation <- deldir(x_visi, y_visi): This line computes the Voronoi tessellation based on the points defined by x_visi and y_visi using the deldir() function.*
- *tiles <- tile.list(tesselation): This line generates a list of tiles representing the Voronoi regions derived from the computed tessellation using the tile.list() function.*
- *plot(tiles, pch = 19, fillcol = hcl.colors(50, "Purple-Yellow"), main = "Voronoi Plot of top 100 entry of Item Visibility"): This line plots the Voronoi diagram represented by tiles. It sets the plotting character (pch) to 19, which is a solid circle, and fills the tiles with colors from a Purple-Yellow gradient created by hcl.colors(). The title of the plot is set to "Voronoi Plot of top 100 entry of Item Visibility".*
- *This voronoi plot show that majority of the entries are centered around 0.00 to 0.15.*