80 Cereals

Analysis using R programming

The 80 cereals dataset likely provides a comprehensive overview of various cereals available in the market, encompassing a wide range of brands, flavors, and nutritional profiles. Through analyzing this dataset, one can gain valuable insights into consumer preferences, industry trends, and nutritional patterns. Firstly, the dataset likely includes information on cereal brands, allowing researchers to identify market leaders and emerging players. It may also delve into flavor variations, highlighting popular choices among consumers. Nutritional content is likely a significant aspect of this dataset, providing data on calories, sugar content, fiber, vitamins, and minerals per serving. This allows for the assessment of the healthiness of different cereals, catering to the needs of health-conscious consumers. Furthermore, the dataset may include pricing information, enabling researchers to explore correlations between price points and nutritional value or brand popularity. Analyzing this dataset could reveal interesting trends, such as the rise of healthier cereal options, the impact of marketing strategies on consumer choices, and the influence of cultural preferences on cereal flavors. Overall, the 80 cereals dataset serves as a valuable resource for understanding the dynamics of the cereal industry and informing consumer choices.

here's a general interpretation of columns in the dataset :

- ***Name:*** *This column likely contains the names of the cereals included in the dataset, allowing for easy identification.*
- ***mfr:*** *This column could indicate the company or brand that produces each cereal, providing insights into market share and brand diversity.*
  A = American Home Food Products;
  G = General Mills
  K = Kelloggs
  N = Nabisco
  P = Post
  Q = Quaker Oats
  R = Ralston Purina
- ***type:*** *This column provide an insight whether the item is hot served or cold served, majority of the entries are cold served type*
  cold
  hot
- ***calories:*** *This column would likely specify the number of calories per serving of each cereal, offering information on its energy content.*
- ***protein:*** *This column might detail the amount of protein in each serving of cereal, an essential nutrient for muscle growth and repair.*
- ***Fat:*** *Here, the dataset would likely list the quantity of fat present in each serving of cereal, including information on saturated, unsaturated, and trans fats if available.*
- *Sodium: This column would probably contain data on the sodium content per serving of each cereal, an important consideration for individuals monitoring their salt intake.*
- ***Fiber:*** *It's likely that this column provides information on the fiber content in each serving of cereal, which is crucial for digestive health and satiety.*
- ***Sugar:*** *This column may specify the amount of sugar present in each serving of cereal, aiding consumers in making informed decisions about their sugar consumption.*
- *Vitamins and Minerals: This section might include columns for various vitamins (e.g., vitamin A, vitamin C) and minerals (e.g., calcium, iron) present in the cereals, offering insights into their nutritional value.*
- ***Rating:*** *This column could contain ratings or scores assigned to each cereal based on factors such as taste, nutritional quality, or consumer satisfaction.*
- ***Cups:*** *This column would likely indicate the weight of a standard serving size for each cereal, facilitating comparisons between different products.*

 # Import the dataset and mutate the manufacturer and type column to proper name

```
cereal_data<-read.csv("C:/Users/Rama/Downloads/cereal.csv")
cereal_data
library(dplyr)
 cereal_data_1 <- cereal_data %>%
mutate(mfr = case_when(
  mfr == "A" ~ "American Home Food Products",
  mfr == "G" ~ "General Mills",
  mfr == "K" ~ "Kelloggs",
  mfr == "N" ~ "Nabisco",
  mfr == "P" ~ "Post",
  mfr == "Q" ~ "Quaker Oats",
  mfr == "R" ~ " Ralston Purina",
  TRUE ~ mfr
))
 cereal_data_1<-cereal_data%>%
   mutate(type=case_when(
     type == "C" ~ "Cold",
     type == "H" ~ "Hot",
     TRUE ~ type
   ))
 cereal_data_1
```

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
1	100% Bran	Nabisco	cold	70	4	1	130	10.0	5.0	6	280	25	3	1.00	0.33	68.40297
2	100% Natural Bran	Quaker Oats	cold	120	3	5	15	2.0	8.0	8	135	0	3	1.00	1.00	33.98368
3	All-Bran	Kelloggs	cold	70	4	1	260	9.0	7.0	5	320	25	3	1.00	0.33	59.42551
4	All-Bran with Extra Fiber	Kelloggs	cold	50	4	0	140	14.0	8.0	0	330	25	3	1.00	0.50	93.70491
5	Almond Delight	Ralston Purina	cold	110	2	2	200	1.0	14.0	8	-1	25	3	1.00	0.75	34.38484
6	Apple Cinnamon Cheerios	General Mills	cold	110	2	2	180	1.5	10.5	10	70	25	1	1.00	0.75	29.50954
7	Apple Jacks	Kelloggs	cold	110	2	0	125	1.0	11.0	14	30	25	2	1.00	1.00	33.17409
8	Basic 4	General Mills	cold	130	3	2	210	2.0	18.0	8	100	25	3	1.33	0.75	37.03856
9	Bran Chex	Ralston Purina	cold	90	2	1	200	4.0	15.0	6	125	25	1	1.00	0.67	49.12025
10	Bran Flakes	post	cold	90	3	0	210	5.0	13.0	5	190	25	3	1.00	0.67	53.31381
11	Cap'n'Crunch	Quaker Oats	cold	120	1	2	220	0.0	12.0	12	35	25	2	1.00	0.75	18.04285
12	Cheerios	General Mills	cold	110	6	2	290	2.0	17.0	1	105	25	1	1.00	1.25	50.76500
13	Cinnamon Toast Crunch	General Mills	cold	120	1	3	210	0.0	13.0	9	45	25	2	1.00	0.75	19.82357
14	Clusters	General Mills	cold	110	3	2	140	2.0	13.0	7	105	25	3	1.00	0.50	40.40021
15	Cocoa Puffs	General Mills	cold	110	1	1	180	0.0	12.0	13	55	25	2	1.00	1.00	22.73645
16	Corn Chex	Ralston Purina	cold	110	2	0	280	0.0	22.0	3	25	25	1	1.00	1.00	41.44502
17	Corn Flakes	Kelloggs	cold	100	2	0	290	1.0	21.0	2	35	25	1	1.00	1.00	45.86332
18	Corn Pops	Kelloggs	cold	110	1	0	90	1.0	13.0	12	20	25	2	1.00	1.00	35.78279
19	Count Chocula	General Mills	cold	110	1	1	180	0.0	12.0	13	65	25	2	1.00	1.00	22.39651
20	Cracklin' Oat Bran	Kelloggs	cold	110	3	3	140	4.0	10.0	7	160	25	3	1.00	0.50	40.44877
21	Cream of Wheat (Quick)	Nabisco	Hot	100	3	0	80	1.0	21.0	0	-1	0	2	1.00	1.00	64.53382
22	Crispix	Kelloggs	cold	110	2	0	220	1.0	21.0	3	30	25	3	1.00	1.00	46.89564
23	Crispy Wheat & Raisins	General Mills	cold	100	2	1	140	2.0	11.0	10	120	25	3	1.00	0.75	36.17620
24	Double Chex	Ralston Purina	cold	100	2	0	190	1.0	18.0	5	80	25	3	1.00	0.75	44.33086
25	Froot Loops	Kelloggs	cold	110	2	1	125	1.0	11.0	13	30	25	2	1.00	1.00	32.20758
26	Frosted Flakes	Kelloggs	cold	110	1	0	200	1.0	14.0	11	25	25	1	1.00	0.75	31.43597
27	Frosted Mini-Wheats	Kelloggs	cold	100	3	0	0	3.0	14.0	7	100	25	2	1.00	0.80	58.34514
28	Fruit & Fibre Dates; Walnuts; and Oats	post	cold	120	3	2	160	5.0	12.0	10	200	25	3	1.25	0.67	40.91705
29	Fruitful Bran	Kelloggs	cold	120	3	0	240	5.0	14.0	12	190	25	3	1.33	0.67	41.01549
30	Fruity Pebbles	post	cold	110	1	1	135	0.0	13.0	12	25	25	2	1.00	0.75	28.02576

- *cereal_data <- read.csv("C:/Users/Rama/Downloads/cereal.csv"): This line of code reads a CSV file named "cereal.csv" located at the specified file path and stores the data in a dataframe called cereal_data.*
- *cereal_data: This line simply prints out the contents of the cereal_data dataframe, allowing the user to inspect the data.*
- *library(dplyr): This line loads the dplyr package, which provides a set of functions for data manipulation.*
- *cereal_data_1 <- cereal_data %>% mutate(...): This code creates a new dataframe called cereal_data_1 by taking the original cereal_data dataframe and applying transformations using the %>% operator (pipe operator), which passes the dataframe to the next function (mutate() in this case) for further processing.*
- *mutate(mfr = case_when(...)): This line creates a new column called mfr in the cereal_data_1 dataframe. It uses case_when() to replace the values in the original mfr column with more descriptive labels. For example, "A" is replaced with "American Home Food Products", "G"*

with "General Mills", and so on. The TRUE ~ mfr part ensures that values not matched by the specified conditions remain unchanged.

- *cereal_data_1 <- cereal_data %>% mutate(...): This line of code performs a similar operation to the previous mutate() call but creates a new column called type. It replaces the values in the original type column with more descriptive labels: "C" is replaced with "Cold", "H" with "Hot", and other values remain unchanged.*
- *cereal_data_1: Finally, this line prints out the contents of the cereal_data_1 dataframe, which now includes the modified mfr and type columns with more interpretable labels.*

 # provide a detailed summary of each column of the dataset

summary(cereal_data_1)

```
     name               mfr                type               calories    
 Length:77          Length:77          Length:77          Min.   : 50.0  
 Class :character   Class :character   Class :character   1st Qu.:100.0  
 Mode  :character   Mode  :character   Mode  :character   Median :110.0  
                                                          Mean   :106.9  
                                                          3rd Qu.:110.0  
                                                          Max.   :160.0  
```

```
    protein            fat              sodium            fiber       
 Min.   :1.000   Min.   :0.000   Min.   :  0.0   Min.   : 0.000  
 1st Qu.:2.000   1st Qu.:0.000   1st Qu.:130.0   1st Qu.: 1.000  
 Median :3.000   Median :1.000   Median :180.0   Median : 2.000  
 Mean   :2.545   Mean   :1.013   Mean   :159.7   Mean   : 2.152  
 3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:210.0   3rd Qu.: 3.000  
 Max.   :6.000   Max.   :5.000   Max.   :320.0   Max.   :14.000  
     carbo           sugars           potass          vitamins      
 Min.   :-1.0   Min.   :-1.000   Min.   : -1.00   Min.   :  0.00  
 1st Qu.:12.0   1st Qu.: 3.000   1st Qu.: 40.00   1st Qu.: 25.00  
 Median :14.0   Median : 7.000   Median : 90.00   Median : 25.00  
 Mean   :14.6   Mean   : 6.922   Mean   : 96.08   Mean   : 28.25  
 3rd Qu.:17.0   3rd Qu.:11.000   3rd Qu.:120.00   3rd Qu.: 25.00  
 Max.   :23.0   Max.   :15.000   Max.   :330.00   Max.   :100.00  
     shelf           weight           cups            rating      
 Min.   :1.000   Min.   :0.50   Min.   :0.250   Min.   :18.04  
 1st Qu.:1.000   1st Qu.:1.00   1st Qu.:0.670   1st Qu.:33.17  
 Median :2.000   Median :1.00   Median :0.750   Median :40.40  
 Mean   :2.208   Mean   :1.03   Mean   :0.821   Mean   :42.67  
 3rd Qu.:3.000   3rd Qu.:1.00   3rd Qu.:1.000   3rd Qu.:50.83  
 Max.   :3.000   Max.   :1.50   Max.   :1.500   Max.   :93.70  
```

- *The code summary(cereal_data_1) provides a summary of the dataframe cereal_data_1, offering a concise overview of its structure and contents. Here's how to interpret the output:*
- *Min: This row displays the minimum value for each numerical column in the dataframe.*
- *1st Qu: This row shows the first quartile (25th percentile) for numerical columns, indicating the value below which 25% of the data falls.*
- *Median: This row displays the median value (50th percentile) for numerical columns, representing the middle value of the dataset.*
- *Mean: This row presents the mean (average) value for numerical columns.*
- *3rd Qu: This row shows the third quartile (75th percentile) for numerical columns, indicating the value below which 75% of the data falls.*
- *Max: This row displays the maximum value for each numerical column.*
- *For categorical or factor columns, the summary typically includes:*
- *Counts: The number of occurrences of each unique value in the column.*
- *Most frequent: The most frequently occurring value in the column.*

- *NA's: The number of missing values, if any, in the column.*
- *Interpreting this summary allows users to quickly understand the range, central tendency, and distribution of numerical variables, as well as the frequency and diversity of categorical variables within the dataframe cereal_data_1.*

 # Calculate the number of missing values in each column of the dataframe

```
missing_values<-colSums(is.na(cereal_data_1))
missing_values
```

```
> missing_values<-colSums(is.na(cereal_data_1))
> missing_values
    name      mfr     type calories  protein      fat   sodium    fiber
       0        0        0        0        0        0        0        0
   carbo   sugars   potass vitamins    shelf   weight     cups   rating
       0        0        0        0        0        0        0        0
```

- *is.na(cereal_data_1): This part of the code generates a logical matrix of the same dimensions as cereal_data_1, where each element is TRUE if the corresponding element in cereal_data_1 is NA (missing), and FALSE otherwise.*
- *colSums(...): This function calculates the sum of TRUE values for each column in the logical matrix generated by is.na(cereal_data_1). Since TRUE is treated as 1 and FALSE as 0 in arithmetic operations, summing the logical values effectively counts the number of missing values in each column.*
- *missing_values: This assigns the resulting column-wise sums of missing values to the variable missing_values.*
- *This result shows that there is no missing value that need to be removed and we can continue to work with the same dataset*

 # calculate the correlation between the different nutritional composition of the cereals

```
cor(cereal_data_1[,c(4:16)])
```

```
> cor(cereal_data_1[,c(4:16)])
            calories      protein          fat       sodium       fiber
calories  1.00000000  0.019066068  0.498609814  0.300649227 -0.29341275
protein   0.01906607  1.000000000  0.208430990 -0.054674348  0.50033004
fat       0.49860981  0.208430990  1.000000000 -0.005407464  0.01671924
sodium    0.30064923 -0.054674348 -0.005407464  1.000000000 -0.07067501
fiber    -0.29341275  0.500330043  0.016719237 -0.070675009  1.00000000
carbo     0.25068091 -0.130863648 -0.318043492  0.355983473 -0.35608274
sugars    0.56234029 -0.329141777  0.270819175  0.101451381 -0.14120539
potass   -0.06660886  0.549407400  0.193278602 -0.032603467  0.90337367
vitamins  0.26535630  0.007335371 -0.031156266  0.361476688 -0.03224268
shelf     0.09723437  0.133864789  0.263691089 -0.069719015  0.29753906
weight    0.69609108  0.216158486  0.214625033  0.308576451  0.24722563
cups      0.08719955 -0.244469158 -0.175892142  0.119664615 -0.51306093
rating   -0.68937603  0.470618465 -0.409283660 -0.401295204  0.58416042
```

	carbo	sugars	potass	vitamins	shelf
calories	0.25068091	0.56234029	-0.06660886	0.265356298	0.09723437
protein	-0.13086365	-0.32914178	0.54940740	0.007335371	0.13386479
fat	-0.31804349	0.27081918	0.19327860	-0.031156266	0.26369109
sodium	0.35598347	0.10145138	-0.03260347	0.361476688	-0.06971902
fiber	-0.35608274	-0.14120539	0.90337367	-0.032242679	0.29753906
carbo	1.00000000	-0.33166538	-0.34968522	0.258147549	-0.10179030
sugars	-0.33166538	1.00000000	0.02169581	0.125137260	0.10043789
potass	-0.34968522	0.02169581	1.00000000	0.020698687	0.36066341
vitamins	0.25814755	0.12513726	0.02069869	1.000000000	0.29926167
shelf	-0.10179030	0.10043789	0.36066341	0.299261665	1.00000000
weight	0.13513642	0.45064760	0.41630315	0.320324059	0.19076197
cups	0.36393247	-0.03235762	-0.49519495	0.128404543	-0.33526876
rating	0.05205466	-0.75967466	0.38016537	-0.240543611	0.02515882

	weight	cups	rating
calories	0.6960911	0.08719955	-0.68937603
protein	0.2161585	-0.24446916	0.47061846
fat	0.2146250	-0.17589214	-0.40928366
sodium	0.3085765	0.11966461	-0.40129520
fiber	0.2472256	-0.51306093	0.58416042
carbo	0.1351364	0.36393247	0.05205466
sugars	0.4506476	-0.03235762	-0.75967466
potass	0.4163032	-0.49519495	0.38016537
vitamins	0.3203241	0.12840454	-0.24054361
shelf	0.1907620	-0.33526876	0.02515882
weight	1.0000000	-0.19958272	-0.29812398
cups	-0.1995827	1.00000000	-0.20316006
rating	-0.2981240	-0.20316006	1.00000000

- *cereal_data_1[, c(4:16)]: This part of the code selects columns 4 through 16 from the dataframe cereal_data_1. It uses the syntax [, c(4:16)] to subset the dataframe, indicating that we want to include all rows ([,]) and columns 4 through 16 (c(4:16)).*
- *cor(...): This function calculates the correlation matrix for the specified subset of columns. The correlation matrix shows the pairwise correlations between all pairs of columns in the selected subset. Each cell in the matrix represents the correlation coefficient between two variables, indicating the strength and direction of their linear relationship. The correlation coefficient ranges from -1 to 1, where:*
- *1 indicates a perfect positive linear relationship,*
- *0 indicates no linear relationship, and*
- *-1 indicates a perfect negative linear relationship.*

***Calories**:*

- *Positively correlated with fat (0.499), sugars (0.562), weight (0.696).*
- *Negatively correlated with fiber (-0.293).*
- *Moderate negative correlation with rating (-0.689).*

***Protein**:*

- *Positively correlated with fiber (0.500), potassium (0.549).*
- *Weak positive correlation with fat (0.208).*
- *Weak negative correlation with sodium (-0.055).*

***Fat**:*

- *Positively correlated with calories (0.499), sugars (0.271), weight (0.215).*
- *Weak positive correlation with protein (0.208).*
- *Weak negative correlation with sodium (-0.005).*

***Sodium**:*

- *Positively correlated with carbo (0.356).*
- *Weak positive correlation with calories (0.301).*
- *Weak negative correlation with fiber (-0.071).*

***Fiber**:*
- *Positively correlated with protein (0.500), potassium (0.903).*
- *Negatively correlated with calories (-0.293), carbo (-0.356), cups (-0.513).*
- *Moderate positive correlation with rating (0.584).*

***Carbohydrates** (Carbo):*
- *Positively correlated with sodium (0.356).*
- *Negatively correlated with fiber (-0.356), sugars (-0.332).*

***Sugars**:*
- *Positively correlated with calories (0.562), shelf (0.100).*
- *Negatively correlated with fiber (-0.141), protein (-0.329), and vitamins (-0.125).*

***Potassium** (**Potass**):*
- *Positively correlated with fiber (0.903), protein (0.549).*
- *Weak positive correlation with fat (0.193).*
- *Weak negative correlation with sodium (-0.033).*

***Vitamins**:*
- *Positively correlated with sodium (0.361), shelf (0.299).*
- *Weak positive correlation with weight (0.320).*
- *Weak negative correlation with fat (-0.031), rating (-0.241).*

***Shelf**:*
- *Positively correlated with vitamins (0.299), potassium (0.361).*
- *Weak positive correlation with fat (0.264), calories (0.097).*
- *Weak negative correlation with fiber (0.298), cups (-0.335).*

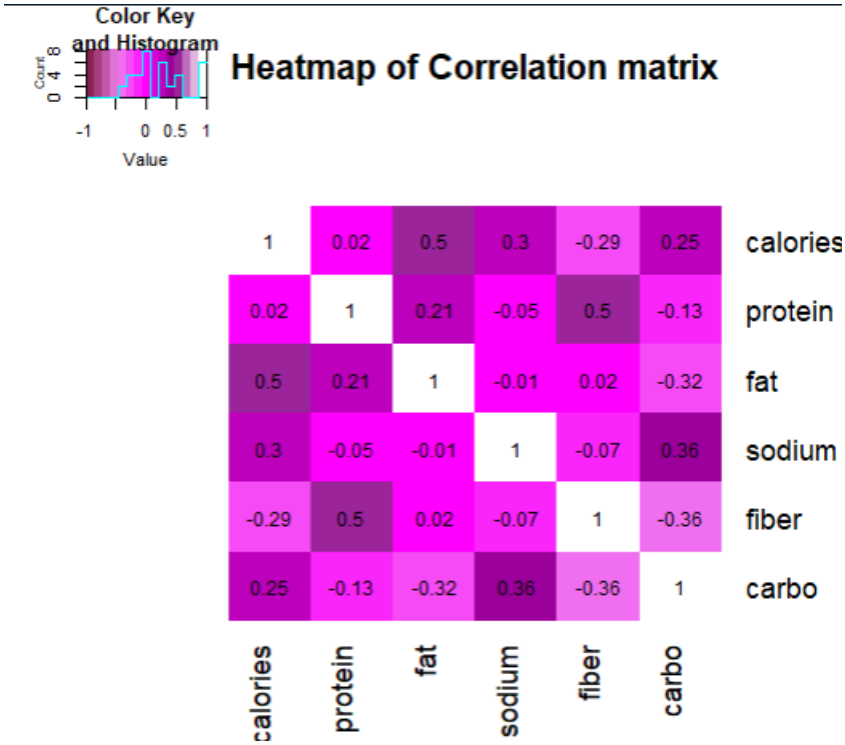
***Weight**:*
- *Positively correlated with calories (0.696), sugars (0.451).*
- *Weak positive correlation with fat (0.215), potassium (0.416).*
- *Weak negative correlation with rating (-0.298), cups (-0.200).*

***Cups**:*
- *Positively correlated with sodium (0.120).*
- *Negatively correlated with fiber (-0.513), shelf (-0.335).*
- *Weak negative correlation with weight (-0.200), rating (-0.203).*

***Rating**:*
- *Positively correlated with fiber (0.584), protein (0.471).*
- *Negatively correlated with sugars (-0.760), fat (-0.409).*
- *Moderate negative correlation with calories (-0.689), weight (-0.298).*

 # calculate the correlation matrix for a subset of variables (calories, protein, fat, sodium, fiber, and carbo) from the `cereal_data_1` dataframe, rounds the correlation values to two decimal places, and then visualizes the correlation matrix using a heatmap.

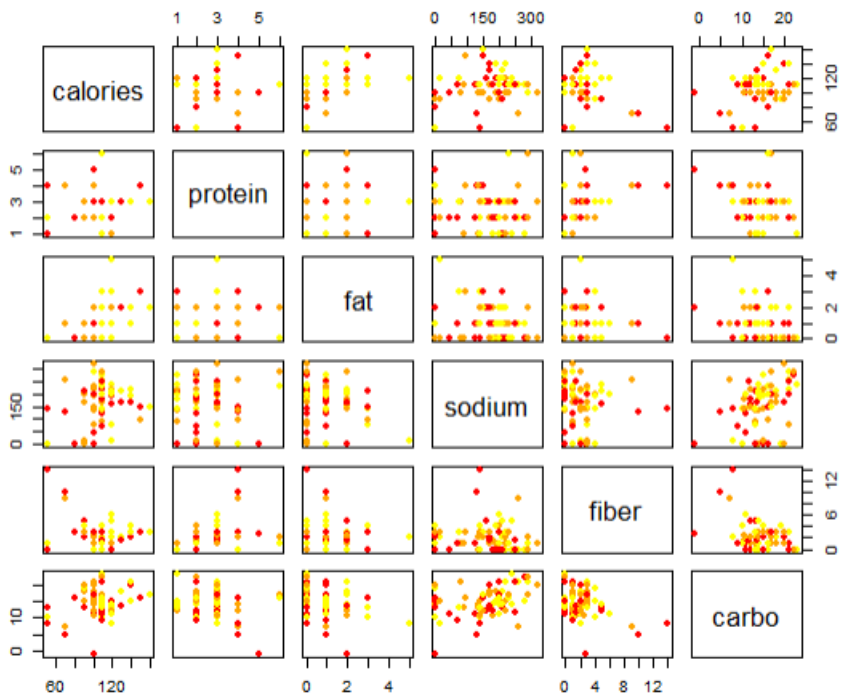
```
correlation<-cor(cereal_data_1[,c(4:9)])
correlation_round<-round(correlation,2)
library(corrplot)
install.packages("gplots")
library(gplots)
heatmap.2(correlation,Rowv =
FALSE,col=colorRampPalette(c("violetred4","violet","magenta","magenta4" ,"white")),Colv =
FALSE,dendrogram = "none",
      scale = "none",trace = "none", cellnote = correlation_round, notecol = "black", notecex = 1,
      main ="Heatmap of Correlation matrix")
```



- *correlation <- cor(cereal_data_1[, c(4:9)]): This line computes the correlation matrix for a subset of columns from the dataframe cereal_data_1. Specifically, it calculates the pairwise correlations between the variables represented by columns 4 to 9 (calories, protein, fat, sodium, fiber, and carbo).*
- *correlation_round <- round(correlation, 2): This line rounds the correlation coefficients to two decimal places. Rounding the coefficients makes the heatmap cleaner and easier to interpret.*
- *library(corrplot): This line loads the corrplot package, which provides functions for visualizing correlation matrices.*
- *install.packages("gplots"): This line installs the gplots package if it's not already installed. gplots is required for the heatmap.2 function used in the code.*
- *library(gplots): This line loads the gplots package, which provides additional plotting functions, including heatmap.2.*
- *heatmap.2(...): This function creates a heatmap visualization of the correlation matrix using the gplots package.*
- *correlation: This argument specifies the correlation matrix to be visualized.*
- *Various arguments (Rowv, col, dendrogram, etc.) are used to customize the appearance of the heatmap. For example:*
- *Rowv = FALSE and Colv = FALSE indicate that no row or column dendrograms should be included.*
- *col specifies the color palette to be used for the heatmap.*
- *cellnote specifies that the rounded correlation coefficients should be displayed within each cell of the heatmap.*
- *main sets the main title of the heatmap.*

 # generates a pairs plot for the dataframe `cereal_data_comp`.

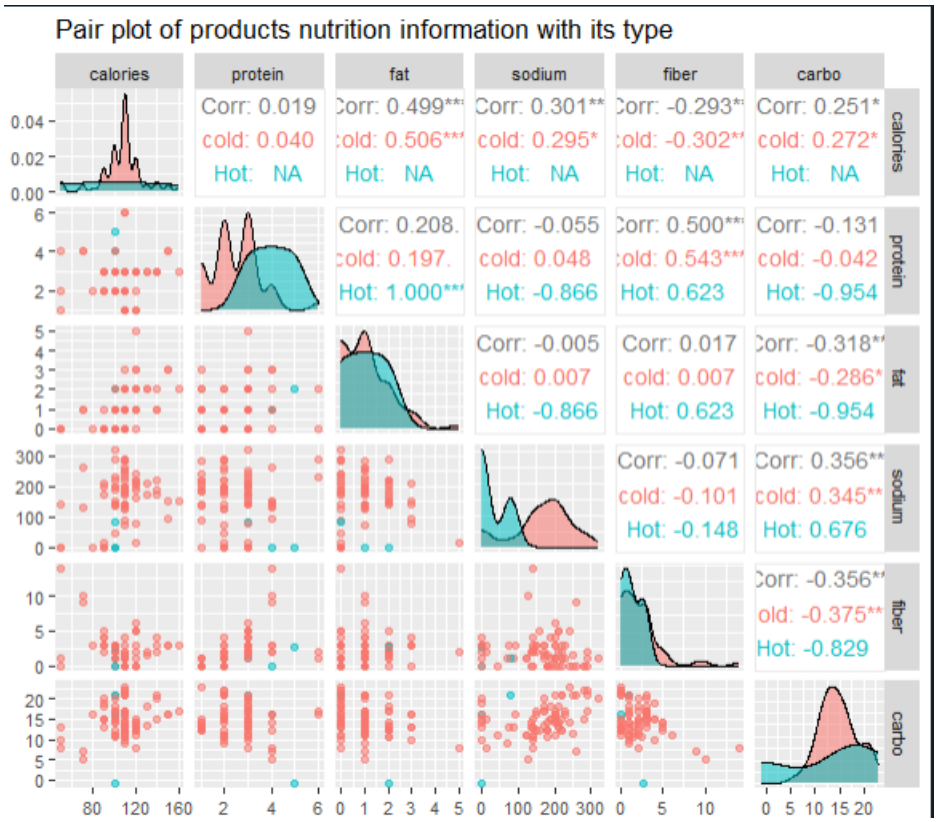
```
pairs(cereal_data_comp,col = c("red", "yellow", "orange"),pch = 16,
    main = "Pairs Plot of Cereals Data")
```



- *pairs() Function:*
- *pairs(): This function creates a pairs plot, also known as a scatterplot matrix. In a pairs plot, each variable in the dataframe is plotted against every other variable, resulting in a grid of scatterplots.*
- *Dataframe:*
- *cereal_data_comp: This is the dataframe used for creating the pairs plot. It likely contains multiple variables (columns), each representing different characteristics or attributes of cereals.*
- *Plot Customization:*
- *col = c("red", "yellow", "orange"): This argument specifies the colors to be used for the points in the scatterplots. In this case, the first variable will be plotted in red, the second in yellow, and the third in orange. If there are more than three variables, the colors will cycle through this sequence.*
- *pch = 16: This argument sets the symbol or glyph used for plotting the points. In this case, it's set to 16, which corresponds to filled circles.*
- *main = "Pairs Plot of Cereals Data": This argument sets the main title of the plot, which indicates that it's a pairs plot of cereals data.*
- *Interpretation:*
- *The pairs plot allows for visual examination of the relationships between different variables in the dataset. Each scatterplot in the grid represents the relationship between two variables, with one variable plotted on the x-axis and the other on the y-axis.*
- *By examining the scatterplots, patterns such as correlations, clusters, or outliers between variables can be identified. This helps in understanding the overall structure and relationships within the dataset.*

 # utilize the `ggpairs()` function from the `GGally` package to create a pair plot of nutrition information for cereal products, with each plot colored by the type of cereal.

```
ggpairs(cereal_data,              # Data frame
      columns = 4:9,       # Columns
      aes(color = type,  # Color by group (cat. variable)
          alpha = 0.5),
      title = "Pair plot of products nutrition information with its type")
```



- *ggpairs() Function:*
- *ggpairs(): This function generates a matrix of scatterplots for each pair of variables in the dataframe. It's part of the GGally package, an extension of ggplot2 for creating beautiful visualizations of data.*
- *Arguments:*
- *cereal_data: This is the dataframe containing the data on cereal products.*
- *columns = 4:9: Specifies which columns of the dataframe to include in the pair plot. In this case, columns 4 to 9 are selected, presumably representing nutritional information such as calories, protein, fat, sodium, fiber, and carbo.*
- *aes(color = type, alpha = 0.5): This argument specifies aesthetics mappings (aes) for the pair plot. It sets the color of the points based on the type column, which likely represents different types or categories of cereals. The alpha parameter controls the transparency of the points, making them slightly transparent (alpha = 0.5) to help visualize overlapping points.*
- *title = "Pair plot of products nutrition information with its type": Sets the main title of the pair plot to provide a brief description of the visualization.*
- *Interpretation:*
- *The pair plot allows for the visual exploration of relationships between different pairs of nutritional variables across cereal products.*
- *Each scatterplot in the matrix represents the relationship between two variables, with one variable plotted on the x-axis and the other on the y-axis.*
- *Points in each scatterplot are colored based on the type of cereal product, providing insight into how different types of cereals are distributed across the nutritional space.*
- *By examining patterns and trends in the scatterplots, such as clusters or correlations, one can gain insights into how different nutritional variables interact with each other and how they vary across different types of cereals.*
- *In summary, this code generates a visually appealing pair plot to explore the relationships between nutritional variables of cereal products, while also considering the cereal type as a categorical variable for color differentiation.*

 # `rcorr()` function from the `Hmisc` package to compute the correlation coefficients and corresponding p-values for a subset of columns (presumably representing nutritional information) in the `cereal_data` dataframe.

```
library(Hmisc)
#create matrix of correlation coefficients and matrix of p-values
rcorr(as.matrix(cereal_data[,c(4:9)]))
```

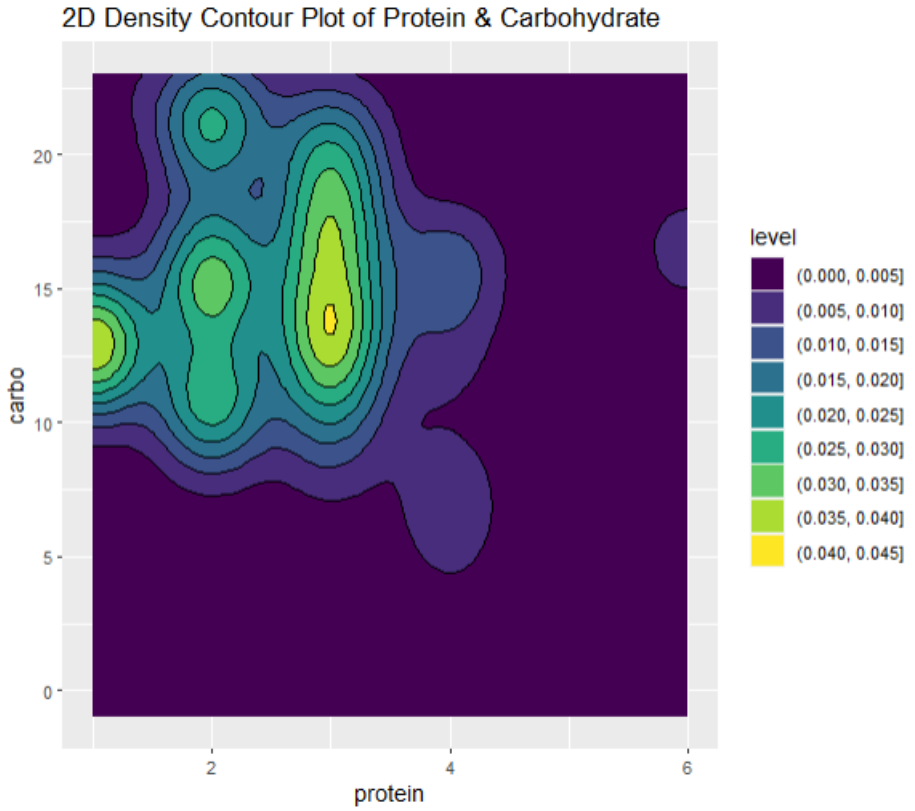
```
          calories protein   fat sodium fiber carbo
calories      1.00    0.02  0.50   0.30 -0.29  0.25
protein       0.02    1.00  0.21  -0.05  0.50 -0.13
fat           0.50    0.21  1.00  -0.01  0.02 -0.32
sodium        0.30   -0.05 -0.01   1.00 -0.07  0.36
fiber        -0.29    0.50  0.02  -0.07  1.00 -0.36
carbo         0.25   -0.13 -0.32   0.36 -0.36  1.00

n= 77


P
          calories protein fat    sodium fiber  carbo
calories           0.8693  0.0000 0.0079 0.0096 0.0279
protein   0.8693           0.0689 0.6367 0.0000 0.2566
fat       0.0000   0.0689         0.9628 0.8852 0.0048
sodium    0.0079   0.6367  0.9628        0.5413 0.0015
fiber     0.0096   0.0000  0.8852 0.5413        0.0015
carbo     0.0279   0.2566  0.0048 0.0015 0.0015
>
```

- *rcorr(): This function computes the correlation matrix for the specified subset of columns and returns both the correlation coefficients and their associated p-values. It's part of the Hmisc package, which provides various functions for statistical analysis.*
- *Arguments:*
- *as.matrix(cereal_data[, c(4:9)]): This specifies the subset of columns from the cereal_data dataframe to be included in the correlation analysis. Columns 4 to 9 are selected, presumably representing nutritional information such as calories, protein, fat, sodium, fiber, and carbo.*
- *Interpretation:*
- *The rcorr() function computes both the correlation coefficients and their corresponding p-values for pairwise comparisons between the selected columns.*
- *The correlation coefficients quantify the strength and direction of the linear relationship between pairs of variables. They range from -1 to 1, where -1 indicates a perfect negative linear relationship, 1 indicates a perfect positive linear relationship, and 0 indicates no linear relationship.*
- *The p-values associated with each correlation coefficient indicate the probability of observing such a correlation by chance, assuming the null hypothesis that there is no correlation between the variables.*
- *This information helps in understanding the statistical significance of the observed correlations. Low p-values (typically < 0.05) indicate that the correlation is unlikely to have occurred by chance and is statistically significant.*
- *By examining both the correlation coefficients and p-values, one can assess both the strength and significance of the relationships between nutritional variables in the dataset.*
- *The correlation coefficient between "calories" and "fat" is 0.50, suggesting a moderate positive linear relationship between the two variables.*
- *The correlation coefficient between "fiber" and "carbo" is -0.36, indicating a moderate negative linear relationship between the two variables.*
- *The p-value associated with the correlation coefficient between "calories" and "fat" is 0.0000, indicating that the correlation is statistically significant.*

- *The p-value associated with the correlation coefficient between "protein" and "sodium" is 0.6367, suggesting that the correlation is not statistically significant at the 0.05 significance level.*

 # utilize the `ggplot2` package in R to create a 2D density plot of the variables "protein" and "carbo" from the `cereal_data` dataframe.

```
Library(ggplot2)
ggplot(cereal_data, aes(x = protein, y = carbo)) +
  geom_density_2d_filled() +
  geom_density_2d(colour = "black")+
  ggtitle("2D Density Plot of Protein & Carbohydrate")
```



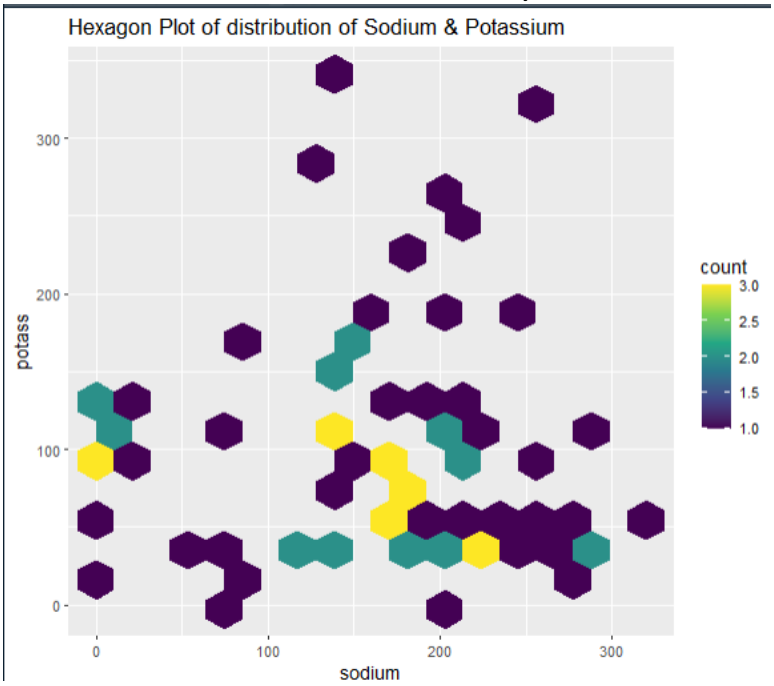
- *library(ggplot2): This line imports the ggplot2 package, which is a powerful and flexible package for creating visualizations in R.*
- *ggplot(cereal_data, aes(x = protein, y = carbo)): This function initializes a ggplot object with the cereal_data dataframe as the dataset and specifies the aesthetics (aes) mapping.*
- *x = protein, y = carbo: This specifies that the variable "protein" should be mapped to the x-axis, and the variable "carbo" should be mapped to the y-axis.*
- *geom_density_2d_filled(): This function adds a filled 2D density plot to the plot. It visualizes the density of points in the x-y space.*
- *geom_density_2d(colour = "black"): This function adds contours to the density plot, outlining the regions of higher density. The contours are colored black.*
- *ggtitle("2D Density Plot of Protein & Carbohydrate"): This function adds a title to the plot, indicating that it is a 2D density plot of the variables "protein" and "carbo".*

Interpretation:

- *The code generates a visualization that represents the joint distribution of the "protein" and "carbo" variables.*
- *The density plot displays areas of higher density in the x-y space, indicating where there is a higher concentration of data points.*
- *The filled regions show the overall density of points, while the contours provide additional information about the shape and spread of the distribution.*
- *Analysts can use this plot to visually assess the relationship between protein and carbohydrate content in the cereal dataset and identify any patterns or trends present in the data.*

 # Create a hexagon Plot of sodium & potassium

```
ggplot(cereal_data, aes(x =sodium , y = potass)) +
 geom_hex(bins=15)+
 scale_fill_viridis_c()+
 ggtitle("Hexagon Plot of distribution of Sodium & Potassium")
```

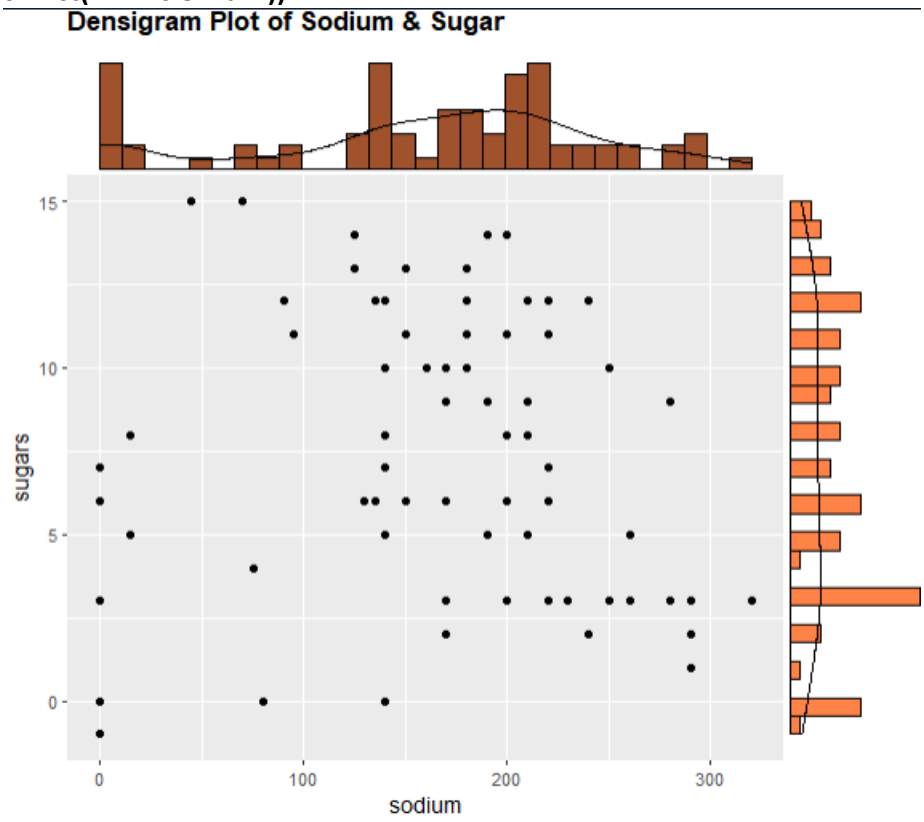


- *ggplot(cereal_data, aes(x = sodium, y = potass))*: This initializes a ggplot *object with the cereal_data dataframe as the dataset and specifies the aesthetics (aes) mapping.*
- *x = sodium, y = potass: This maps the variable "sodium" to the x-axis and the variable "potass" to the y-axis.*
- *geom_hex(bins = 15): This adds a layer of hexagons to the plot, where each hexagon represents a bin containing data points. The number of bins is specified as 15, controlling the granularity of the hexagon plot.*
- *scale_fill_viridis_c(): This function sets the color scale for filling the hexagons. The colors are based on the viridis color palette, which provides visually appealing and perceptually uniform color gradients.*
- *ggtitle("Hexagon Plot of distribution of Sodium & Potassium"): This adds a title to the plot, indicating that it is a hexagon plot visualizing the distribution of sodium and potassium variables.*
- *The code generates a hexagon plot, which is a type of 2D histogram, showing the density of points in the x-y space formed by the "sodium" and "potass" variables.*
- *Each hexagon represents a bin containing data points. The color intensity of the hexagons indicates the density of points in that region.*
- *The plot provides insight into the distribution and relationship between sodium and potassium content in the cereal dataset.*
- *By examining the plot, analysts can identify any patterns or trends in the distribution of these variables, such as clusters or outliers.*
- *There majority of the values of sodium & potassium are in the count of 1-2.*

 # create a densigram plot of the variables "sodium" and "sugars" from the `cereal_data` dataframe using the `ggplot2` and `ggExtra` packages.

```
install.packages("ggExtra")
library(ggExtra)
p <- ggplot(cereal_data, aes(x = sodium, y = sugars)) +
 geom_point()+
 ggtitle("Densigram Plot of Sodium & Sugar ")+
```

```
  theme(plot.title = element_text(face = "bold")
# Arguments for each marginal histogram
ggMarginal(p, type = "densigram",
        xparams = list(fill = "sienna"),
        yparams = list(fill = "sienna1"))
```



- *install.packages("ggExtra"): This command installs the ggExtra package if it's not already installed.*
- *library(ggExtra): This loads the ggExtra package into the R session, allowing us to use its functions.*
- *ggplot(cereal_data, aes(x = sodium, y = sugars)) + geom_point() + ggtitle("Densigram Plot of Sodium & Sugar "): This initializes a ggplot object with the cereal_data dataframe as the dataset and specifies the aesthetics (aes) mapping. It also adds a scatterplot layer using geom_point() and sets the plot title to "Densigram Plot of Sodium & Sugar".*
- *theme(plot.title = element_text(face = "bold")): This line customizes the appearance of the plot title, making it bold.*
- *ggMarginal(p, type = "densigram", xparams = list(fill = "sienna"), yparams = list(fill = "sienna1")): This function adds marginal density histograms (densigrams) to the scatterplot p.*
- *type = "densigram": Specifies that densigrams should be added to the plot.*
- *xparams and yparams: These arguments specify the aesthetics (e.g., fill color) for the marginal histograms. In this case, the fill colors for the x and y marginal histograms are set to "sienna" and "sienna1", respectively.*
- *The code generates a scatterplot of the variables "sodium" and "sugars", showing the relationship between these two variables.*
- *Additionally, densigrams are added as marginal histograms on the x and y axes, representing the distribution of data along each axis.*
- *The densigrams provide insight into the univariate distributions of "sodium" and "sugars" while still allowing the viewer to see their bivariate relationship in the scatterplot.*

 # compute the total calories for each manufacturer in the `cereal_data` dataframe using dplyr package

```
library(dplyr)
cereal_calories<-cereal_data%>%
```

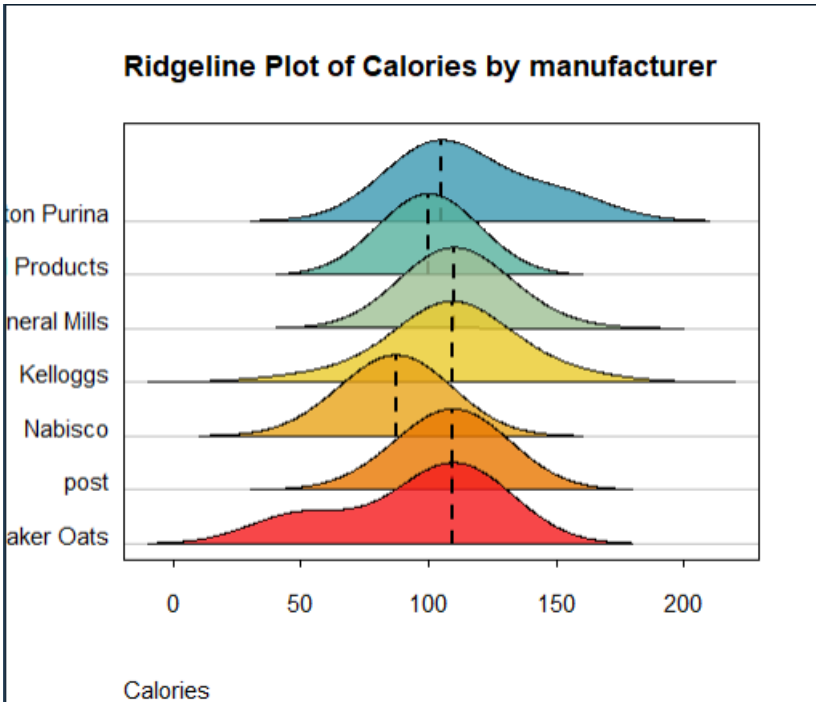
```
 group_by(mfr)%>%
summarise(calories)
cereal_calories
```

```
   mfr                              calories
   <chr>                               <int>
 1 " Ralston Purina"                     110
 2 " Ralston Purina"                      90
 3 " Ralston Purina"                     110
 4 " Ralston Purina"                     100
 5 " Ralston Purina"                     150
 6 " Ralston Purina"                     150
 7 " Ralston Purina"                     110
 8 " Ralston Purina"                     100
 9 "American Home Food Products"         100
10 "General Mills"                       110
# i 67 more rows
# i Use `print(n = ...)` to see more rows
```

- *library(dplyr): This loads the dplyr package, which provides a set of functions for data manipulation and transformation.*
- *%>%: This is the pipe operator, which allows you to chain together multiple operations.*
- *group_by(mfr): This groups the data by the "mfr" (manufacturer) column. This means that subsequent operations will be applied within each group separately.*
- *summarise(calories): This summarizes the grouped data by calculating the total calories within each group (manufacturer). The summarise() function creates a new data frame with one row for each group and the specified summary statistic(s). In this case, it computes the sum of "calories" within each group.*
- *The result is stored in a new dataframe called cereal_calories.*
- *Each row of cereal_calories represents a different manufacturer.*
- *The column "calories" contains the total calories for the cereals produced by each manufacturer.*

 # utilizes the `ridgeline` package, which provides functions to create ridge plots

```
install.packages("remotes")
remotes::install_github("R-CoderDotCom/ridgeline@main")
library(ridgeline)
ridgeline(cereal_calories$calories,cereal_calories$mfr, bw = 20,mode =
TRUE,xlab="Calories",main="Ridgeline Plot of Calories by manufacturer")
```



- *install.packages("remotes"): This installs the remotes package, which is used for installing packages from GitHub.*
- *remotes::install_github("R-CoderDotCom/ridgeline@main"): This installs the ridgeline package directly from GitHub.*
- *library(ridgeline): This loads the ridgeline package, making its functions available for use.*
- *ridgeline(cereal_calories$calories, cereal_calories$mfr, bw = 20, mode = TRUE, xlab = "Calories", main = "Ridgeline Plot of Calories by Manufacturer"): This function generates a ridge plot.*
- *cereal_calories$calories: This specifies the numeric data (total calories) to be plotted.*
- *cereal_calories$mfr: This specifies the categorical data (manufacturer) used for grouping the ridges.*
- *bw = 20: This sets the bandwidth for density estimation. Higher values result in smoother ridges.*
- *mode = TRUE: This determines whether the mode of each ridge is highlighted.*
- *xlab = "Calories": This sets the label for the x-axis.*
- *main = "Ridgeline Plot of Calories by Manufacturer": This sets the main title of the plot.*
- *The code creates a ridge plot showing the distribution of total calories across different manufacturers.*
- *Each ridge represents the density distribution of calorie values for a specific manufacturer.*
- *The plot allows for easy comparison of the calorie distributions among manufacturers.*
- *The highlighted mode of each ridge provides insight into the most common calorie value for each manufacturer.*
- *This visualization helps to identify patterns and variations in calorie content among different cereal manufacturers.*

 # converts a subset of the `cereal_data` dataframe into a matrix named `matrix_data`

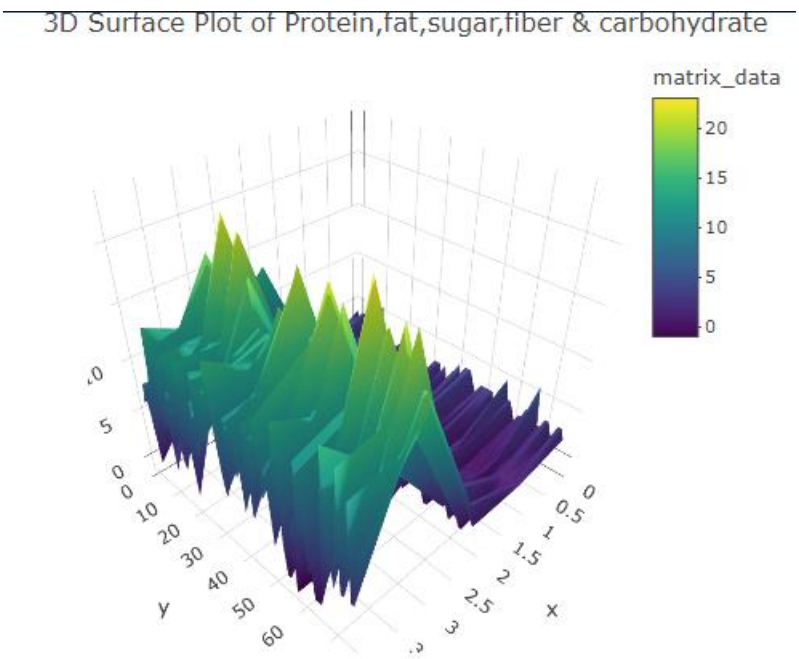
```
matrix_data<-as.matrix(cereal_data[,c(5,6,8,9,10)])
matrix_data
```

```
> matrix_data<-as.matrix(cereal_data[,c(5,6,8,9,10)])
> matrix_data
      protein fat fiber carbo sugars
 [1,]       4   1  10.0   5.0      6
 [2,]       3   5   2.0   8.0      8
 [3,]       4   1   9.0   7.0      5
 [4,]       4   0  14.0   8.0      0
 [5,]       2   2   1.0  14.0      8
 [6,]       2   2   1.5  10.5     10
 [7,]       2   0   1.0  11.0     14
 [8,]       3   2   2.0  18.0      8
 [9,]       2   1   4.0  15.0      6
[10,]       3   0   5.0  13.0      5
[11,]       1   2   0.0  12.0     12
[12,]       6   2   2.0  17.0      1
[13,]       1   3   0.0  13.0      9
[14,]       3   2   2.0  13.0      7
```

- *cereal_data[, c(5, 6, 8, 9, 10)]: This selects a subset of columns from the cereal_data dataframe. The selected columns are 5th, 6th, 8th, 9th, and 10th columns.*
- *as.matrix(): This function converts the selected subset of the dataframe into a matrix.*
- *The resulting matrix is assigned to the variable matrix_data.*
- *matrix_data: This variable contains the selected subset of data from the cereal_data dataframe, now represented as a matrix.*
- *Each row of the matrix corresponds to a different observation (cereal), and each column represents a different variable.*
- *The matrix contains values from the selected columns: fiber, carbo, sugars, potass, and vitamins.*
- *By converting the subset of the dataframe into a matrix, you can perform matrix operations and calculations on this data.*
- *This transformation may be useful for certain types of analyses or computations that require matrix input.*

 # utilize the `plotly` package in R to create a 3D surface plot of the data stored in the `matrix_data`

```
library(plotly)
fig <- plot_ly(z = ~matrix_data)
fig <- fig %>% add_surface()
fig
fig<-fig%>%
  layout(title="3D Surface Plot of Protein,fat,sugar,fiber & carbohydrate")
fig
```



- *library(plotly): This loads the plotly package, which provides functions for creating interactive plots.*
- *fig <- plot_ly(z = ~matrix_data): This initializes a plotly figure object using the data from the matrix_data matrix. The z argument specifies the data to be represented on the z-axis.*
- *fig <- fig %>% add_surface(): This adds a 3D surface plot to the figure object created in the previous step. The surface plot visualizes the values from the matrix_data matrix in a three-dimensional space.*
- *fig: This displays the plotly figure object, showing the 3D surface plot.*
- *fig <- fig %>% layout(title="3D Surface Plot of Protein, Fat, Sugar, Fiber & Carbohydrate"): This sets the title of the plot to "3D Surface Plot of Protein, Fat, Sugar, Fiber & Carbohydrate".*
- *The code generates an interactive 3D surface plot visualizing the data stored in the matrix_data matrix.*
- *The x-axis, y-axis, and z-axis of the plot represent different variables or dimensions of the data.*
- *The plot provides a comprehensive visualization of the relationships between the variables represented by the rows and columns of the matrix_data matrix in a three-dimensional space.*

 # Group the data by manufacturer and type and summarize the rating by its range

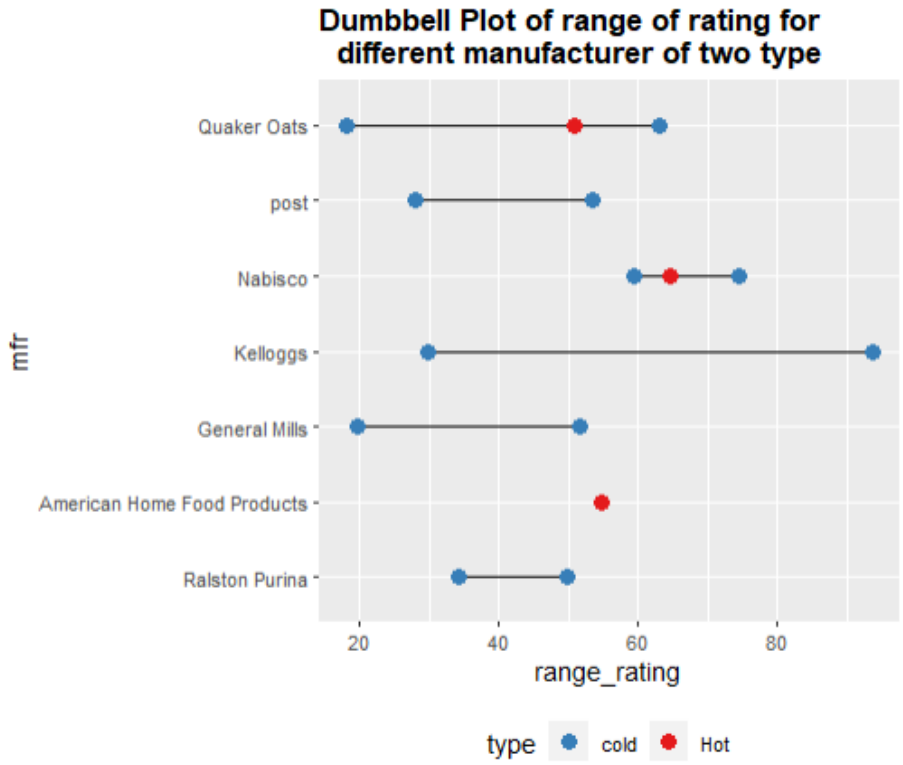
```
df<-cereal_data%>%
  group_by(mfr,type)%>%
  summarise(range_rating=range(rating))
df
```

```
# Groups:   mfr, type [9]
   mfr                          type  range_rating
   <chr>                        <chr>        <dbl>
 1 " Ralston Purina"            cold          34.1
 2 " Ralston Purina"            cold          49.8
 3 "American Home Food Products" Hot          54.9
 4 "American Home Food Products" Hot          54.9
 5 "General Mills"              cold          19.8
 6 "General Mills"              cold          51.6
 7 "Kelloggs"                   cold          29.9
 8 "Kelloggs"                   cold          93.7
 9 "Nabisco"                    Hot           64.5
10 "Nabisco"                    Hot           64.5
11 "Nabisco"                    cold          59.4
12 "Nabisco"                    cold          74.5
13 "Quaker Oats"                Hot           50.8
14 "Quaker Oats"                Hot           50.8
15 "Quaker Oats"                cold          18.0
16 "Quaker Oats"                cold          63.0
17 "post"                       cold          28.0
18 "post"                       cold          53.4
```

- *cereal_data is first grouped by two variables: mfr (manufacturer) and type (type of cereal).*
- *Within each group, the code calculates the range of the rating variable using the range() function.*
- *The summarise() function is used to create a summary dataframe (df) with three columns: mfr, type, and range_rating.*
- *Each row of the df dataframe represents a unique combination of manufacturer and type of cereal, with the corresponding range of ratings for that group.*
- *In essence, this code is summarizing the range of ratings for each combination of manufacturer and type of cereal in the cereal_data dataframe.*

 # create a dumbbell plot using the `ggplot2` package, visualizing the range of ratings for different manufacturers of two types of cereal.

```
ggplot(df, aes(x = range_rating, y = mfr)) +
  geom_line() +
  geom_point(aes(color = type), size = 3) +
  scale_color_brewer(palette ="Set1" , direction = -1) +
  ggtitle("Dumbbell Plot of range of rating for
  different manufacturer of two type")+
  theme(legend.position = "bottom",plot.title = element_text(face = "bold"))
```



- *ggplot(df, aes(x = range_rating, y = mfr)): This initializes the ggplot object, specifying the dataframe df as the data source and mapping the x-axis to range_rating (the range of ratings) and the y-axis to mfr (manufacturer).*
- *geom_line(): This adds lines connecting the points. Since there's no explicit y aesthetic defined for this layer, it will connect points based on the order in the dataframe.*
- *geom_point(aes(color = type), size = 3): This adds points to the plot, representing each combination of manufacturer and type. The points are colored based on the type of cereal, and their size is set to 3.*
- *scale_color_brewer(palette ="Set1" , direction = -1): This sets the color palette for the points. It uses the Brewer color palette "Set1" and reverses the direction of the colors.*
- *ggtitle("Dumbbell Plot of range of rating for different manufacturer of two type"): This adds a title to the plot.*
- *theme(legend.position = "bottom", plot.title = element_text(face = "bold")): This adjusts the theme of the plot. It sets the position of the legend at the bottom and makes the plot title bold.*
- *Here the Dumbbell plot represent that majority of the manufacturer are cold type and kelloggs has the highest range followed by Quaker Oats and least by Raiston Purina*

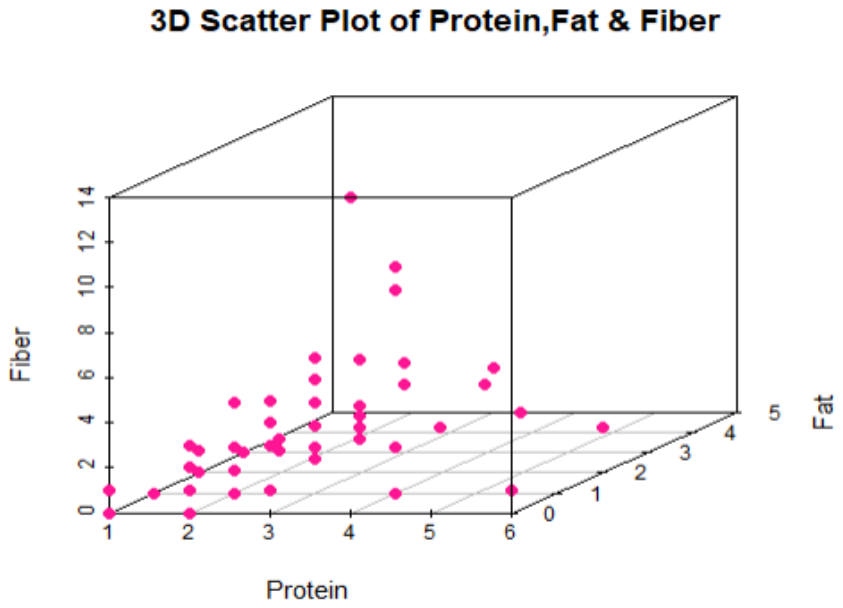
 # generate a histogram using the `ggplot2` package, visualizing the distribution of calorie values in the `cereal_data` dataframe.

```
ggplot(cereal_data, aes(x = calories, fill = ..x..)) +
  geom_histogram(binwidth = 10) +
  scale_fill_gradient(low = "red", high = "yellow") +
  geom_text(stat = "count", aes(label = ..count..), vjust = -0.5) +
  labs(title = "Histogram with Calories Count", x = "Value", y = "Frequency")+
  theme(plot.title = element_text(face = "bold"))
```

- *ggplot(cereal_data, aes(x = calories, fill = ..x..)): This initializes the ggplot object, specifying the dataframe cereal_data as the data source and mapping the x-axis to the calories variable. The fill aesthetic is set to ..x.., which represents the x-coordinate of each bar in the histogram.*
- *geom_histogram(binwidth = 10): This adds a histogram layer to the plot, with each bar representing a range of calorie values. The binwidth parameter sets the width of each bin to 10 units.*
- *scale_fill_gradient(low = "red", high = "yellow"): This sets the color gradient for the fill of the bars in the histogram. The gradient ranges from "red" (low values) to "yellow" (high values).*
- *geom_text(stat = "count", aes(label = ..count..), vjust = -0.5): This adds text labels to each bar of the histogram, displaying the count of observations within each bin. The vjust parameter adjusts the vertical justification of the text labels.*
- *labs(title = "Histogram with Calories Count", x = "Value", y = "Frequency"): This sets the title and axis labels for the plot.*
- *theme(plot.title = element_text(face = "bold")): This adjusts the theme of the plot, making the plot title bold.*
- *This histogram shows that value between 90-120 has the highest frequency, while less than or more than these value have very less frequency*

 # use the `scatterplot3d` package to create a 3D scatter plot of protein, fiber & fat

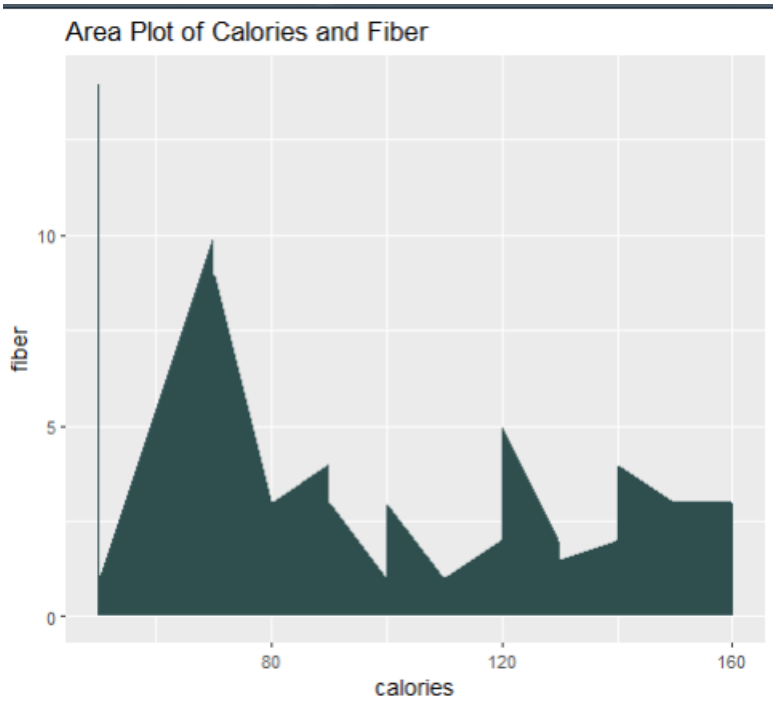
```
install.packages("scatterplot3d") # Install
library("scatterplot3d")
composition<-cereal_data[,c(5,6,8)]
scatterplot3d(composition, pch = 16, color="deeppink",main="3D Scatter Plot of Protein,Fat & Fiber",
              xlab = "Protein ",
              ylab = "Fat",
              zlab = "Fiber")
```



- *install.packages("scatterplot3d"): This command installs the scatterplot3d package if it's not already installed in the R environment.*
- *library("scatterplot3d"): This loads the scatterplot3d package into the current R session, making its functions available for use.*
- *composition <- cereal_data[,c(5,6,8)]: This line selects columns 5, 6, and 8 from the cereal_data dataframe, which likely correspond to variables related to protein, fat, and fiber content of the cereals. It assigns these columns to a new dataframe called composition.*
- *scatterplot3d(composition, pch = 16, color="deeppink", main="3D Scatter Plot of Protein,Fat & Fiber", xlab = "Protein ", ylab = "Fat", zlab = "Fiber"): This function call creates the 3D scatter plot using the composition dataframe.*
- *pch = 16 sets the type of point marker to be used.*
- *color = "deeppink" sets the color of the points in the plot.*
- *main = "3D Scatter Plot of Protein, Fat & Fiber" sets the title of the plot.*
- *xlab = "Protein ", ylab = "Fat", zlab = "Fiber" set the labels for the x, y, and z axes respectively, indicating the variables represented on each axis.*
- *This plot shows that majority of the values lies in the lower axis between 0 to 6*

 # utilize `ggplot2` to create an area plot based on the data provided in the dataframe `df_filtered`

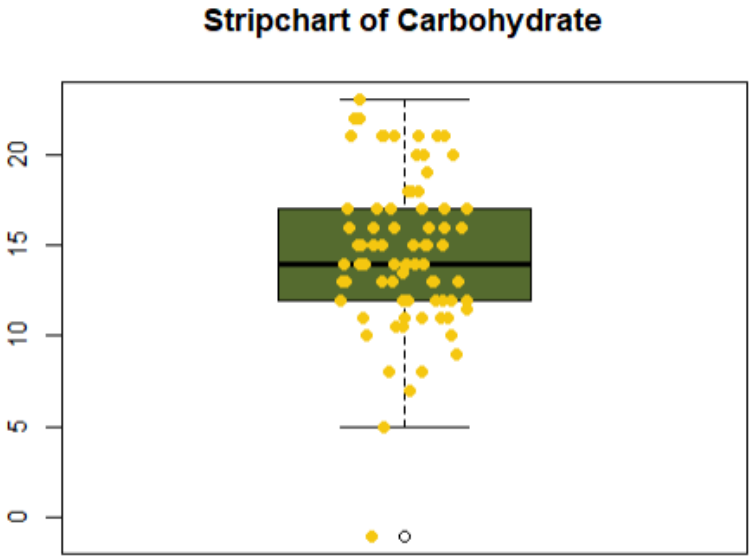
```
ggplot(df_filtered, aes(x =calories , y = fiber)) +
  geom_area(fill="darkslategray")+
  ggtitle("Area Plot of Calories and Fiber")
```



- *ggplot(df_filtered, aes(x = calories, y = fiber)): This initializes a ggplot object with the dataframe df_filtered as the data source and specifies the aesthetics mapping. It maps the x-axis to the calories variable and the y-axis to the fiber variable.*
- *geom_area(fill="darkslategray"): This adds an area layer to the plot. The fill parameter sets the color of the area to "darkslategray".*
- *ggtitle("Area Plot of Calories and Fiber"): This sets the title of the plot to "Area Plot of Calories and Fiber".*
- *Overall, this code generates an area plot representing the relationship between calories and fiber, with each point in the plot representing a combination of calories and fiber values from the df_filtered dataframe. The area under the curve formed by connecting these points is shaded in dark slate gray.*

 # create a boxplot and a stripchart (or jitter plot) for the variable "carbo" in the dataframe `cereal_data`.

```
boxplot(cereal_data$carbo, col = "darkolivegreen",main="Stripchart of Carbohydrate")
# Points
stripchart(cereal_data$carbo,           # Data
        method = "jitter", # Random noise
        pch = 19,         # Pch symbols
        col =7,          # Color of the symbol
        vertical = TRUE,   # Vertical mode
        add = TRUE)       # Add it over
```



- *boxplot(cereal_data$carbo, col = "darkolivegreen",main="Stripchart of Carbohydrate"): This line generates a boxplot for the variable "carbo" from the cereal_data dataframe.*
- *cereal_data$carbo selects the "carbo" column from the dataframe.*
- *col = "darkolivegreen" sets the color of the boxes in the boxplot to "darkolivegreen".*
- *main = "Stripchart of Carbohydrate" sets the main title of the plot to "Stripchart of Carbohydrate".*
- *stripchart(cereal_data$carbo, method = "jitter", pch = 19, col = 7, vertical = TRUE, add = TRUE): This line adds a stripchart (or jitter plot) on top of the previously created boxplot.*
- *cereal_data$carbo selects the "carbo" column from the dataframe.*
- *method = "jitter" specifies that the points should be jittered to avoid overlap.*
- *pch = 19 sets the symbol used for the points to a solid circle.*
- *col = 7 sets the color of the points to a numeric value (color palette index).*
- *vertical = TRUE specifies that the stripchart should be vertical.*
- *add = TRUE adds the stripchart to the existing plot.*
- *Together, these commands create a visualization that combines a boxplot and a stripchart for the "carbo" variable, providing both a summary of the distribution through the boxplot and a detailed view of individual data points through the stripchart.*

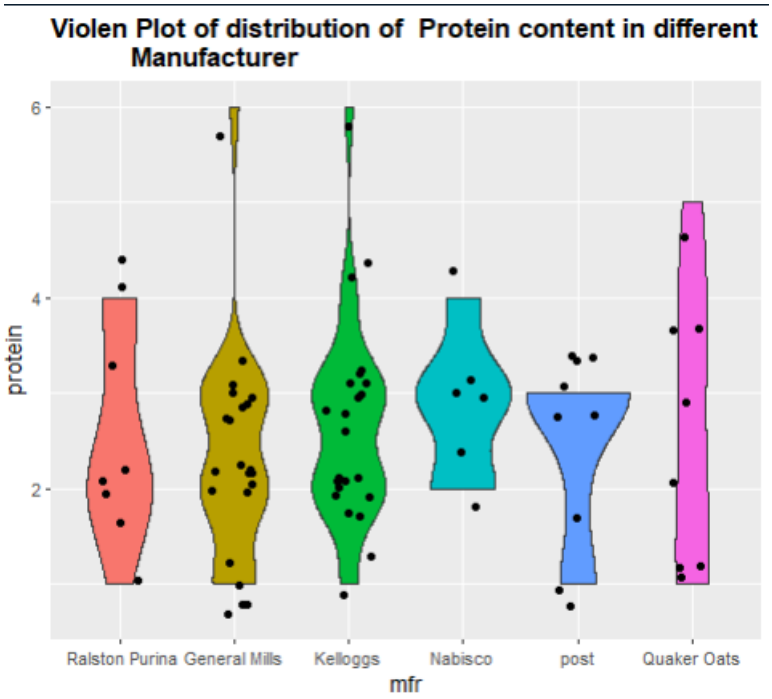
 # Remove rows with "American Home Food Products" in the mfr column

```
df_filtered <- cereal_data %>%
  filter(mfr != "American Home Food Products")
```

- *cereal_data %>%: This is a pipe operator (%>%), which allows you to perform operations sequentially on the dataframe cereal_data.*
- *filter(mfr != "American Home Food Products"): This function filters the dataframe based on a condition. In this case, it selects rows where the value in the mfr column is not equal to "American Home Food Products".*

 # create a violin plot to visualize the distribution of protein content in different manufacturers' cereals.

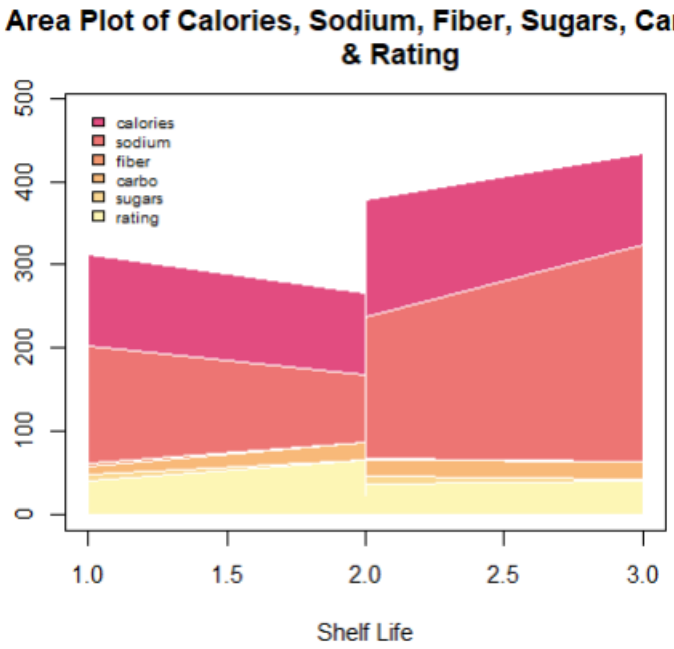
```
ggplot(df_filtered, aes(x = mfr, y = protein, fill = mfr)) +
  geom_violin(alpha = 1) +
  geom_point(position = position_jitter(seed = 1, width = 0.2)) +
  ggtitle("Violen Plot of distribution of  Protein content in different
          Manufacturer")+
  theme(legend.position = "none",plot.title = element_text(face = "bold"))
```



- *ggplot(df_filtered, aes(x = mfr, y = protein, fill = mfr)): This initializes a ggplot object with the dataframe df_filtered as the data source. It maps the x-axis to the manufacturer (mfr), the y-axis to the protein content, and the fill color aesthetic also to the manufacturer (mfr).*
- *geom_violin(alpha = 1): This adds a violin plot layer to the ggplot object. Violin plots provide a combination of box plot and density plot, showing the distribution of the data. The alpha = 1 argument sets the transparency of the violins to 1 (fully opaque).*
- *geom_point(position = position_jitter(seed = 1, width = 0.2)): This adds individual data points on top of the violin plot using jittering to avoid overlap. The seed = 1 ensures reproducibility of the random jittering, and width = 0.2 controls the amount of jittering.*
- *ggtitle("Violen Plot of distribution of Protein content in different Manufacturer"): This sets the title of the plot to "Violin Plot of distribution of Protein content in different Manufacturer".*
- *theme(legend.position = "none", plot.title = element_text(face = "bold")): This adjusts the theme of the plot. It removes the legend (legend.position = "none") and makes the plot title bold (plot.title = element_text(face = "bold")).*
- *Overall, this code generates a violin plot showing the distribution of protein content in cereals from different manufacturers, with individual data points overlaid for additional detail.*

 # utilize the "areaplot" package to create a stacked area plot visualizing the trends of various attributes (calories, sodium, fiber, sugars, carbs, and rating) across different shelf life categories.

```
install.packages("areaplot")
library(areaplot)
# Colors
cols <- hcl.colors(6, palette = "PinkYl")
#Stacked area chart with custom borders
areaplot(df_filtered$shelf, df_filtered[,c(4,7,8,9,10,16)], col = cols,
      border = "white",
      lwd = 1,
      lty = 1,
      xlab = "Shelf Life",
      ylab= "",
      main="Area Plot of Calories, Sodium, Fiber, Sugars, Carbs
      & Rating",
      legend = TRUE,
      args.legend = list(x = "topleft", cex = 0.65))
```



- *install.packages("areaplot"): This command installs the "areaplot" package from CRAN (the Comprehensive R Archive Network) if it's not already installed on your system.*
- *ibrary(areaplot): This command loads the "areaplot" package into your current R session, making its functions available for use.*
- *cols <- hcl.colors(6, palette = "PinkYl"): This line generates a vector of colors using the HCL color space. It creates 6 colors from the "PinkYl" palette. These colors will likely be used for filling the areas in the stacked area plot.*
- *areaplot(...): This function creates the stacked area plot. Here are the parameters passed to the function:*
- *df_filtered$shelf: This specifies the x-axis variable, which represents the shelf life categories.*
- *df_filtered[,c(4,7,8,9,10,16)]: This specifies the y-axis variables, which include columns 4, 7, 8, 9, 10, and 16 from the df_filtered dataframe. These columns likely represent attributes such as calories, sodium, fiber, sugars, carbs, and rating.*
- *col = cols: This parameter assigns the colors specified in the cols vector to the different areas in the plot.*
- *border = "white", lwd = 1, lty = 1: These parameters set the border color, width, and line type for the areas in the plot.*
- *xlab = "Shelf Life", ylab = "": These parameters set the labels for the x-axis and y-axis of the plot. In this case, the y-axis label is left blank.*
- *main = "Area Plot of Calories, Sodium, Fiber, Sugars, Carbs & Rating": This sets the main title of the plot.*
- *legend = TRUE: This parameter indicates that a legend should be displayed on the plot.*
- *args.legend = list(x = "topleft", cex = 0.65): This specifies the position and size of the legend. In this case, the legend is placed in the top left corner of the plot, and its size is set to 0.65 times the default size.*
- *Overall, this code generates a stacked area plot showing the trends of various attributes across different shelf life categories. Each area in the plot represents one of the attributes, and the colors differentiate between them.*

 # create and function of sin(sqrt(x^2)) and find the range of fat

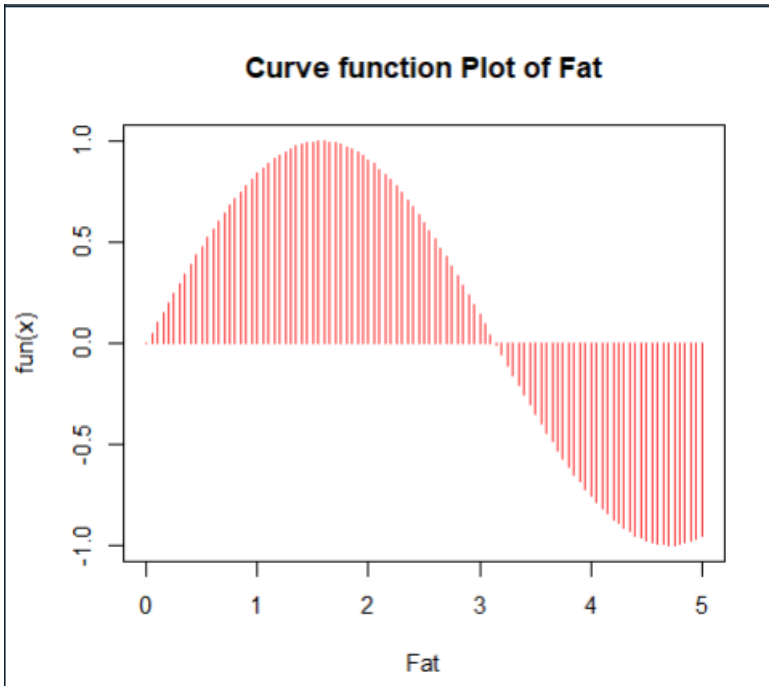
```
fun <- function(x) {
  sin(sqrt(x ^ 2))
}
range(cereal_data$fat)
```

```
> range(cereal_data$fat)
[1] 0 5
>
```

- *fun <- function(x) {sin(sqrt(x ^ 2)} This defines a function named fun.*
- *The function takes a single argument x.*
- *Inside the function, it calculates the square of x using x ^ 2, then takes the square root of the result using sqrt(), and finally takes the sine of the square root value using sin().*
- *Essentially, this function computes the sine of the square root of the square of the input x.*
- *range(cereal_data$fat)*
- *This code calculates the range of values in the "fat" column of the dataframe cereal_data.*
- *The range() function returns a vector containing the minimum and maximum values of the input vector.*
- *In this case, it calculates the range of values in the "fat" column of the cereal_data dataframe.*
- *So the range of fat is between 0 to 5*

 # generate a plot of the function defined by `fun(x)` over the interval from 0 to 5.

```
curve(fun, from = 0, to = 5,
    type = "h", col="firebrick1", xlab= fat,main="Curve function Plot of Fat")
```



- *curve(fun, from = 0, to = 5, type = "h"): This function plots a curve of the function fun(x) over the specified range.*
- *fun is the function to be plotted.*
- *from = 0 specifies the starting point of the x-axis.*
- *to = 5 specifies the ending point of the x-axis.*
- *type = "h" indicates that the plot should be generated using vertical lines (histogram-like).*
- *col = "firebrick1" sets the color of the lines in the plot to "firebrick1".*
- *xlab = "fat" sets the label for the x-axis to "fat".*
- *main = "Curve function Plot of Fat" sets the main title of the plot to "Curve function Plot of Fat".*
- *This code will plot the curve of the function fun(x) over the interval [0, 5], using vertical lines, with the x-axis labeled as "fat" and the plot title as "Curve function Plot of Fat".*

 # calculate the maximum sodium content for each manufacturer in the `cereal_data` dataframe.

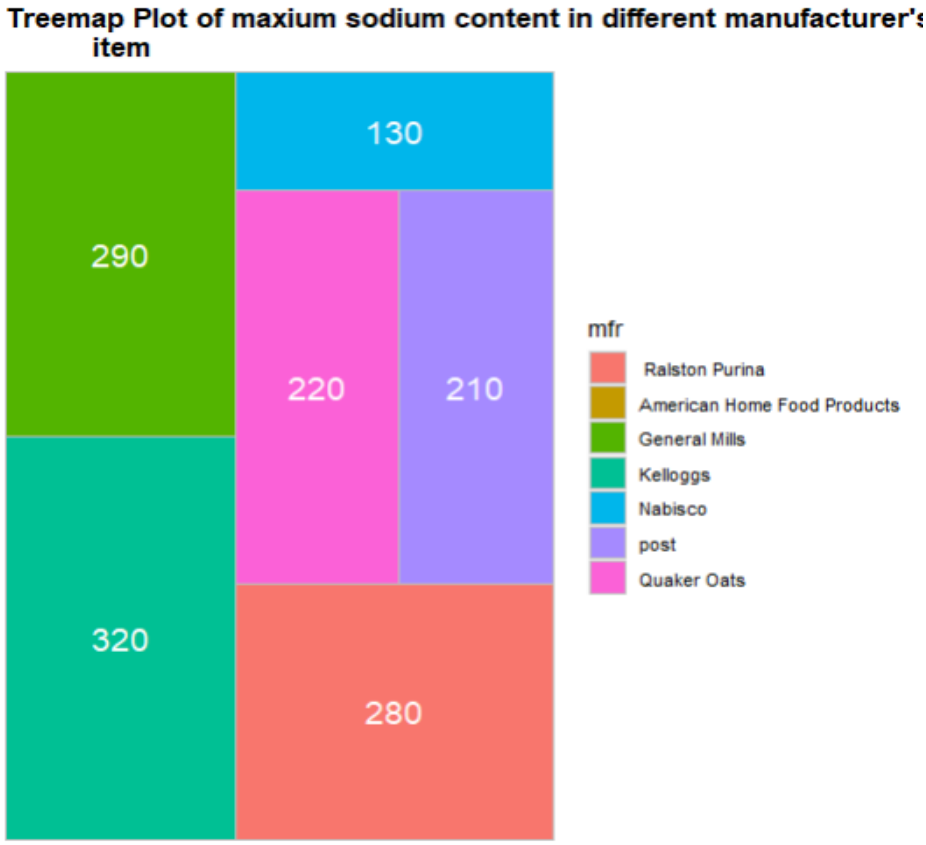
```
x_234<-cereal_data%>%
  group_by(mfr)%>%
  summarize(max_sodium=max(sodium))
x_234
```

```
# A tibble: 7 × 2
  mfr                          max_sodium
  <chr>                             <int>
1 " Ralston Purina"                   280
2 "American Home Food Products"         0
3 "General Mills"                     290
4 "Kelloggs"                          320
5 "Nabisco"                           130
6 "Quaker Oats"                       220
7 "post"                              210
>
```

- *group_by(mfr)%>% This code first takes the cereal_data dataframe.*
- *It then groups the data by the mfr (manufacturer) variable using the group_by() function from the dplyr package. This means that subsequent operations will be performed within each group defined by unique manufacturers.*
- *summarize(max_sodium=max(sodium)) Within each group defined by manufacturer, this code calculates the maximum sodium content using the max() function applied to the sodium variable.*
- *The summarize() function from the dplyr package creates a summary dataframe where each row represents a unique manufacturer, and the max_sodium column contains the maximum sodium content for that manufacturer.*
- *x_234 This code simply displays the resulting dataframe x_234 containing the maximum sodium content for each manufacturer.*
- *Here in this grouping kelloggs has the highest amount of sodium content whereas Nabisco has the least content*

#utilize `ggplot2` to create a treemap plot visualizing the maximum sodium content for different manufacturers' items.

```
ggplot(x_234, aes(area = max_sodium, fill = mfr, label = max_sodium)) +
  geom_treemap() +
  ggtitle("Treemap Plot of maxium sodium content in different manufacturer's
        item")+
  theme(plot.title =element_text(face = "bold"))+
  geom_treemap_text(colour = "white",
                    place = "centre",
                    size = 15)
```

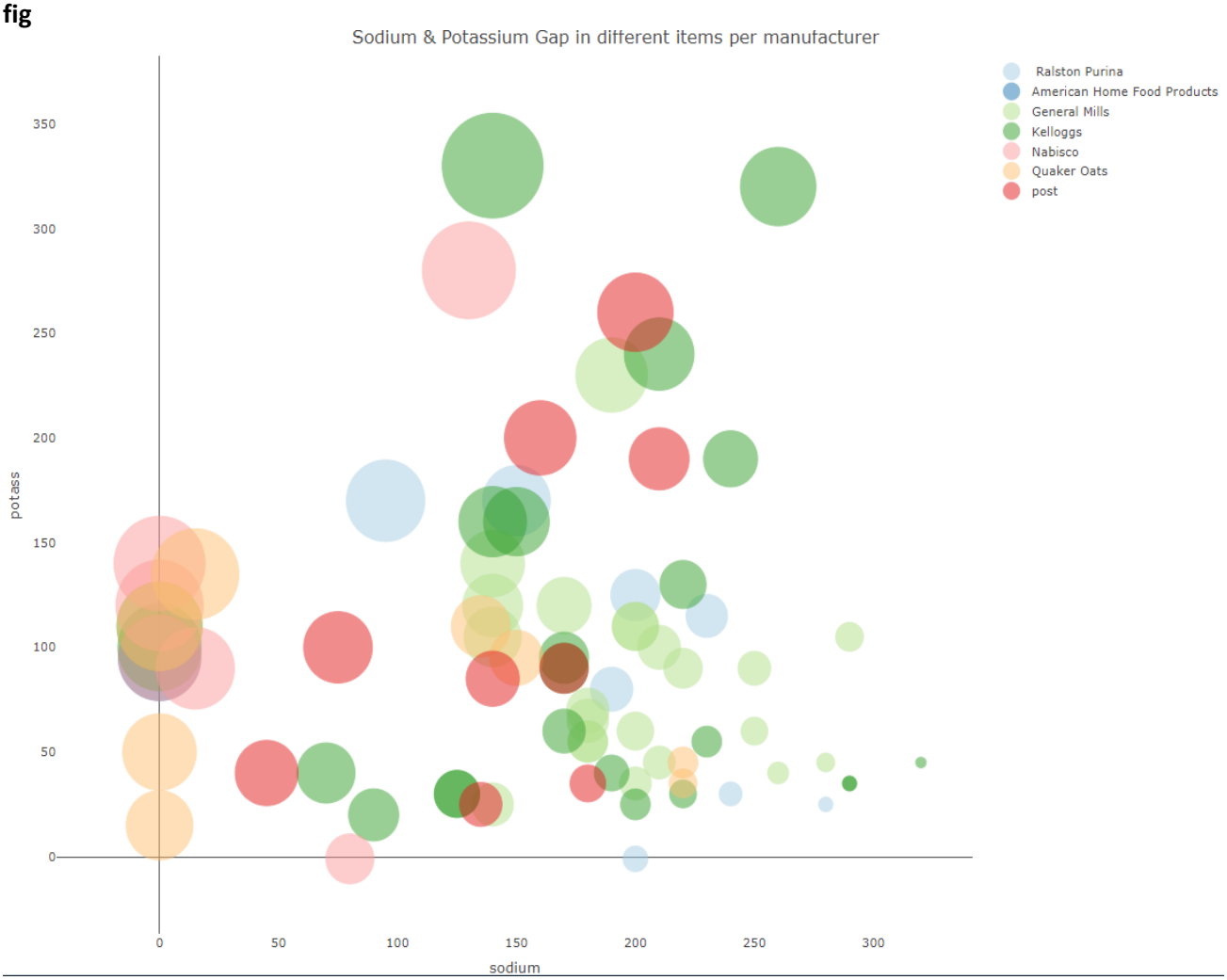


- *ggplot(x_234, aes(area = max_sodium, fill = mfr, label = max_sodium)) This initializes a ggplot object with the dataframe x_234 as the data source.*
- *The area aesthetic is mapped to the max_sodium variable, which will determine the area of each rectangle in the treemap.*
- *The fill aesthetic is mapped to the mfr variable, which will determine the color of each rectangle representing a manufacturer.*
- *The label aesthetic is mapped to the max_sodium variable as well, which means the labels on the treemap will display the maximum sodium content for each manufacturer's item.*
- *geom_treemap()This adds the treemap geometry to the plot. Each rectangle in the treemap represents a manufacturer's item, with the size of the rectangle proportional to the maximum sodium content.*
- *ggtitle("Treemap Plot of maxium sodium content in different manufacturer's*
- *item") This sets the title of the plot to "Treemap Plot of maximum sodium content in different manufacturer's items" with bold font face.*
- *The theme() function allows customization of various plot elements. In this case, it customizes the appearance of the plot title to be bold.*
- *geom_treemap_text(colour = "white", place = "centre", size = 15) This adds text labels to the treemap. The labels display the maximum sodium content for each manufacturer's item.*
- *colour = "white" sets the color of the text labels to white for better contrast against the colored rectangles.*
- *place = "centre" positions the labels at the center of each rectangle.*
- *size = 15 sets the size of the text labels to 15.*

 # Create a bubble chart of sodium and potassium gap of various products of different manufacturers

```
attach(cereal_data)
gap<-potass-sodium
fig <- plot_ly(cereal_data, x = ~sodium, y = ~potass, text = ~name, type = 'scatter', mode = 'markers',
size = ~gap, color = ~mfr, colors = 'Paired',
        marker = list(opacity = 0.5, sizemode = 'diameter'))
```

```
fig <- fig %>% layout(title = "Sodium & Potassium Gap in different items per manufacturer",
                      xaxis = list(showgrid = FALSE),
                      yaxis = list(showgrid = FALSE),
                      showlegend = TRUE)
fig
```

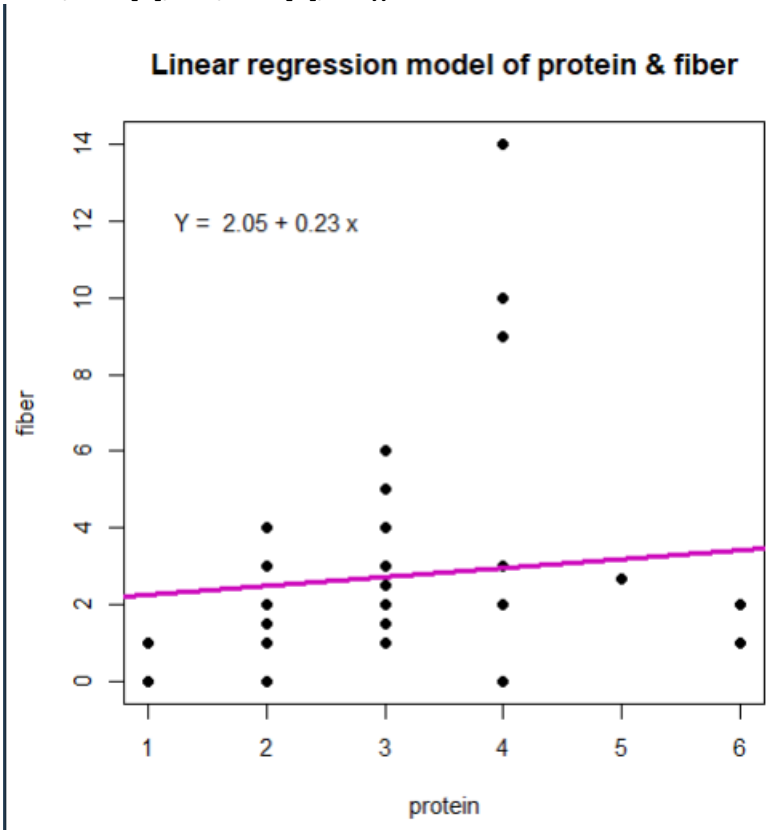


- *attach(cereal_data) This attaches the dataframe cereal_data to the search path, making its variables directly accessible by their names.*
- *gap<-potass-sodium This calculates the difference between the potassium (potass) and sodium (sodium) content for each cereal item, storing the result in a variable named gap.*
- *fig <- plot_ly(cereal_data, x = ~sodium, y = ~potass, text = ~name, type = 'scatter', mode = 'markers', size = ~gap, color = ~mfr, colors = 'Paired',*
  marker = list(opacity = 0.5, sizemode = 'diameter'))
- *This creates a Plotly plot object (fig) using the plot_ly() function.*
- *It specifies sodium as the x-axis variable (x), potass as the y-axis variable (y), and name as the text to display when hovering over each point (text).*
- *The plot type is set to 'scatter', and the mode is set to 'markers', indicating that the plot will consist of individual points.*
- *The size of each marker is determined by the calculated gap variable.*
- *Points are colored by the mfr variable, with colors defined by the 'Paired' color palette.*
- *Additional marker properties like opacity and sizemode are set using the marker argument.*
- *fig <- fig %>% layout(title = "Sodium & Potassium Gap in different items per manufacturer",*
  xaxis = list(showgrid = FALSE),
  yaxis = list(showgrid = FALSE),
  showlegend = TRUE) This customizes the layout of the plot.
- *It sets the title of the plot to "Sodium & Potassium Gap in different items per manufacturer".*
- *The x-axis and y-axis gridlines are turned off (showgrid = FALSE) to improve visual clarity.*
- *The legend is displayed (showlegend = TRUE).*
- *Fig This displays the final Plotly plot object (fig), showing the scatter plot with the specified layout and data.*

- *This bubble chart shows that Nabisco and Quaker Oats have very less difference between sodium & Potassium, on contrast kelloggs, raiston purina, post have high gap between sodium & potassium*

 # perform linear regression analysis between the variables `protein` and `fiber`.

```
#Linear model
model111 <- lm(protein ~ fiber)
# Scatter plot and linear regression line
plot(protein,fiber , pch = 16, main="Linear regression model of protein & fiber")
abline(model111, col = 6, lwd = 3)
# Text
coef <- round(coef(model111), 2)
text(2, 12,  paste("Y = ", coef[1], "+", coef[2], "x"))
```

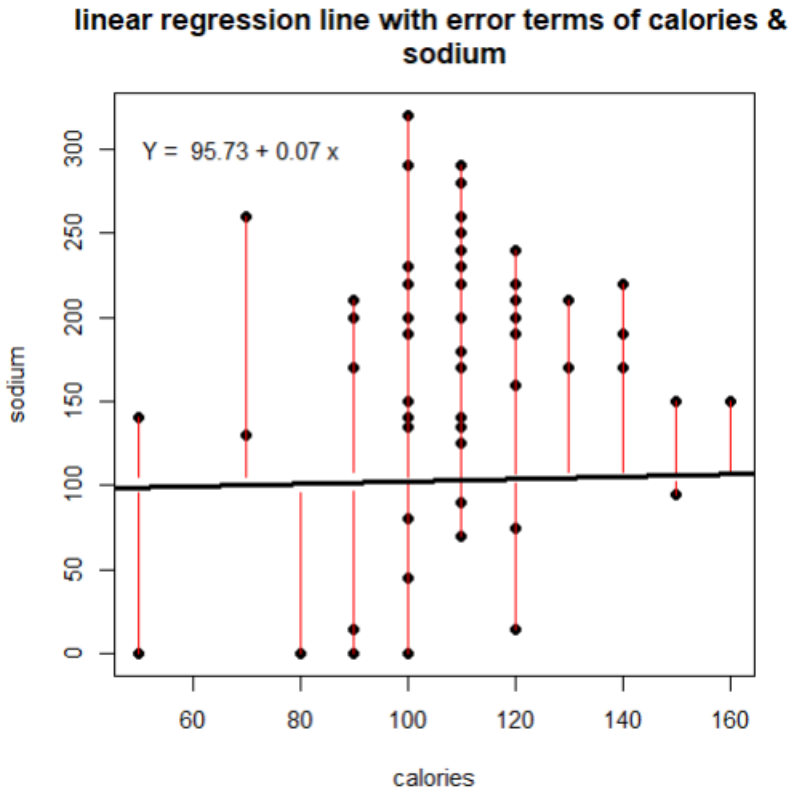


- *model111 <- lm(protein ~ fiber) This line creates a linear regression model (lm) where protein is the dependent variable and fiber is the independent variable.*
- *The model model111 represents the linear relationship between protein and fiber*
- *plot(protein,fiber , pch = 16, main="Linear regression model of protein & fiber")*
  abline(model111, col = 6, lwd = 3)
- *This code creates a scatter plot of the data points with protein on the y-axis and fiber on the x-axis.*
- *pch = 16 sets the point character to solid circles.*
- *The main argument sets the main title of the plot.*
- *abline(model111, col = 6, lwd = 3) adds a regression line to the scatter plot. The abline function plots the regression line based on the linear model model111. The col = 6 argument sets the color of the line to be blue (color index 6), and lwd = 3 sets the line width to 3.*
- *coef <- round(coef(model111), 2)*
  text(2, 12, paste("Y = ", coef[1], "+", coef[2], "x")) This code adds text annotation to the plot to display the equation of the regression line.
- *coef(model111) extracts the coefficients of the linear model model111.*
- *round(coef(model111), 2) rounds the coefficients to two decimal places.*

- *text(2, 12, paste("Y = ", coef[1], "+", coef[2], "x")) places the text "Y = [intercept] + [slope]x" at the coordinates (2, 12) on the plot. Here, [intercept] and [slope] represent the intercept and slope coefficients of the linear model, respectively.*
- *This linear regression shows that as fiber increase by one unit protein increase by 0.23 units and as fiber is zero protein is 2.05.*

 # perform linear regression analysis between the variables `calories` and `sodium`, and then creates a scatter plot with a linear regression line and error terms.

```
# Linear model
modelo <- lm(calories ~ sodium)

# Scatter plot and linear regression line
plot(calories,sodium, pch = 16,main="linear regression line with error terms of calories &
    sodium")

# Segments with error terms
segments(x0 = calories, x1 =calories, y0 = sodium, y1 = predict(modelo),
       lwd = 1, col = "red")

# Regression line
abline(modelo, col =1, lwd = 3)

# Paint the points again over the segments
points(carbo, fat, pch = 16)
coef <- round(coef(modelo), 2)
text(69, 300,  paste("Y = ", coef[1], "+", coef[2], "x"))
```



- *modelo <- lm(calories ~ sodium) This line creates a linear regression model (lm) where calories is the dependent variable and sodium is the independent variable.*
- *The model modelo represents the linear relationship between calories and sodium.*

- *plot(calories,sodium, pch = 16,main="linear regression line with error terms of calories &*
- *sodium")*
- *This code creates a scatter plot of the data points with calories on the y-axis and sodium on the x-axis.*
- *pch = 16 sets the point character to solid circles.*
- *The main argument sets the main title of the plot.*
- *abline(modelo, col = 1, lwd = 3) adds a regression line to the scatter plot based on the linear model modelo. The col = 1 argument sets the color of the line to black, and lwd = 3 sets the line width to 3.*
- *segments(x0 = calories, x1 =calories, y0 = sodium, y1 = predict(modelo), lwd = 1, col = "red")*
- *This code adds error terms to the plot. It draws vertical lines (segments) from each data point to the corresponding predicted values on the regression line.*
- *x0 and x1 are both set to calories, which means the segments are vertical.*
- *y0 corresponds to the actual sodium values, and y1 corresponds to the predicted sodium values based on the linear model modelo.*
- *lwd = 1 sets the width of the segments to 1, and col = "red" sets their color to red.*
- *coef <- round(coef(modelo), 2)*
- *text(69, 300, paste("Y = ", coef[1], "+", coef[2], "x"))*
- *This code adds text annotation to the plot to display the equation of the regression line.*
- *coef(modelo) extracts the coefficients of the linear model modelo.*
- *round(coef(modelo), 2) rounds the coefficients to two decimal places.*
- *text(69, 300, paste("Y = ", coef[1], "+", coef[2], "x")) places the text "Y = [intercept] + [slope]x" at the coordinates (69, 300) on the plot. Here, [intercept] and [slope] represent the intercept and slope coefficients of the linear model, respectively.*
- *So this linear regression shows that as sodium increases by one unit calories increase by 0.07 units and as sodium is zero calories increase by 95.73 units*

 # group data by manufacturer and summarize by average sugar

```
install.packages("waffle")
library(waffle)
library(dplyr)
x_1<-df_filtered%>%
 group_by(mfr)%>%
 summarize(mean_sugars=mean(sugars))
x_1
```

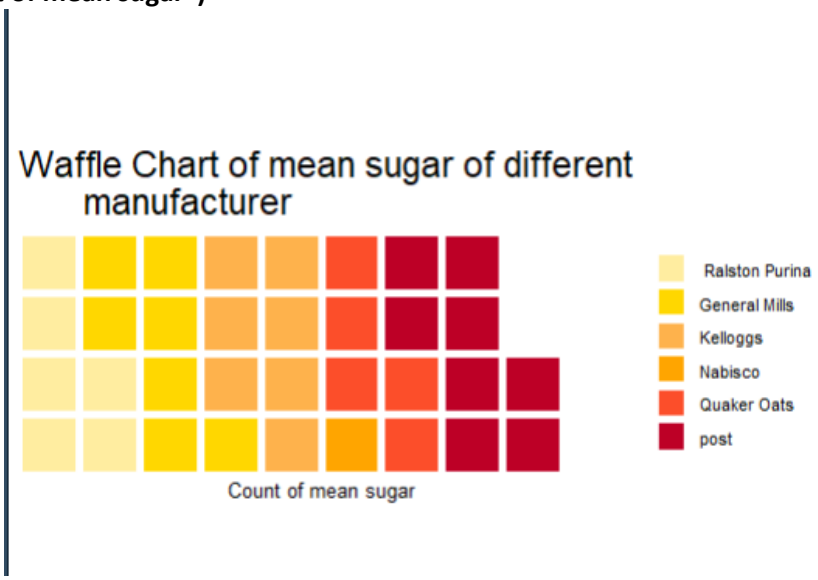
```
# A tibble: 6 × 2
  mfr                mean_sugars
  <chr>                    <dbl>
1 " Ralston Purina"         6.12
2 "General Mills"           7.95
3 "Kelloggs"                7.57
4 "Nabisco"                 1.83
5 "Quaker Oats"             5.25
6 "post"                    8.78
```

- *install.packages("waffle") installs the "waffle" package. This command is typically run once to install the package if it's not already installed in your R environment.*
- *library(waffle) and library(dplyr) load the "waffle" and "dplyr" packages into the R session. This makes functions and datasets from these packages available for use in your R script.*

- *x_1 <- df_filtered %>% ...: This line of code assumes there's a dataframe named df_filtered. It pipes (%>%) this dataframe into a series of operations using the dplyr package.*
- *group_by(mfr): Groups the data by the unique values in the "mfr" column. This is a common operation in data analysis where you want to perform calculations separately within each group.*
- *summarize(mean_sugars = mean(sugars)): Calculates the mean of the "sugars" column within each group defined by the "mfr" column. The result is a new dataframe (x_1) with one row for each unique value in the "mfr" column, and a column "mean_sugars" containing the calculated mean sugar value for each group.*
- *Print Result: x_1 is printed, showing the calculated mean sugar content for each group of manufacturers.*
- *This result shows that post manufacturer has the highest average sugar content in its product while Nabisco has the lowest*

 # generate a waffle chart using the "waffle" package, visualizing the mean sugar content of different manufacturers.

```
waffle(x_1, rows = 4,
    colors = c("#FFEDA0","gold", "#FEB24C","orange", "#FC4E2A", "#BD0026"),title ="Waffle Chart of mean sugar of different
    manufacturer",
    xlab = "Count of mean sugar")
```



- *x_1: This is the data frame containing the summary statistics of mean sugar content for different manufacturers.*
- *rows = 4: Specifies the number of rows in the waffle chart grid. In this case, the waffle chart will have 4 rows.*
- *colors = c("#FFEDA0","gold", "#FEB24C","orange", "#FC4E2A", "#BD0026"):*
- *Defines the color palette to be used in the waffle chart. The waffle segments will be filled with colors specified in this vector.*
- *title = "Waffle Chart of mean sugar of different manufacturer":*
- *Sets the title of the waffle chart.*
- *xlab = "Count of mean sugar":*
- *Labels the x-axis of the waffle chart.*

 # create a new dataframe named `df_123` using the `data.frame` function

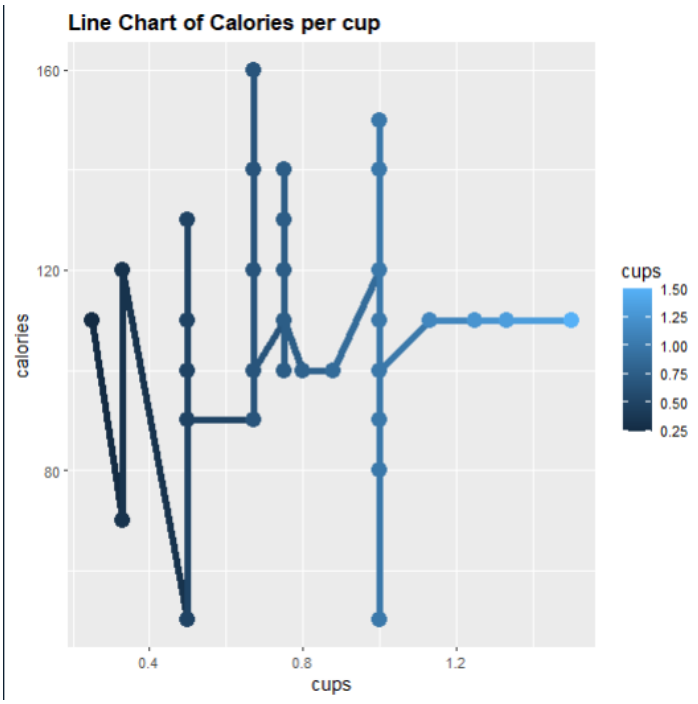
```
df_123<-data.frame(cups=cereal_data$cups,calories=cereal_data$calories)
head(df_123)
```

	cups	calories
1	0.33	70
2	1.00	120
3	0.33	70
4	0.50	50
5	0.75	110
6	0.75	110
7	1.00	110
8	0.75	130
9	0.67	90
10	0.67	90
11	0.75	120
12	1.25	110
13	0.75	120
14	0.50	110

- *data.frame(cups = cereal_data$cups, calories = cereal_data$calories): This part of the code creates a dataframe with two columns: "cups" and "calories". The data for the "cups" column is taken from the "cups" column of a dataframe or dataset named cereal_data, and the data for the "calories" column is taken from the "calories" column of the same dataframe.*
- *head(df_123): This function is used to display the first few rows of the newly created dataframe df_123. It's a convenient way to inspect the structure and contents of a dataframe without printing the entire dataset. By default, head() displays the first 11 rows of the dataframe.*

 # use the ggplot2 package to create a line chart with points representing the relationship between the "cups" and "calories" variables in the dataframe df_123.

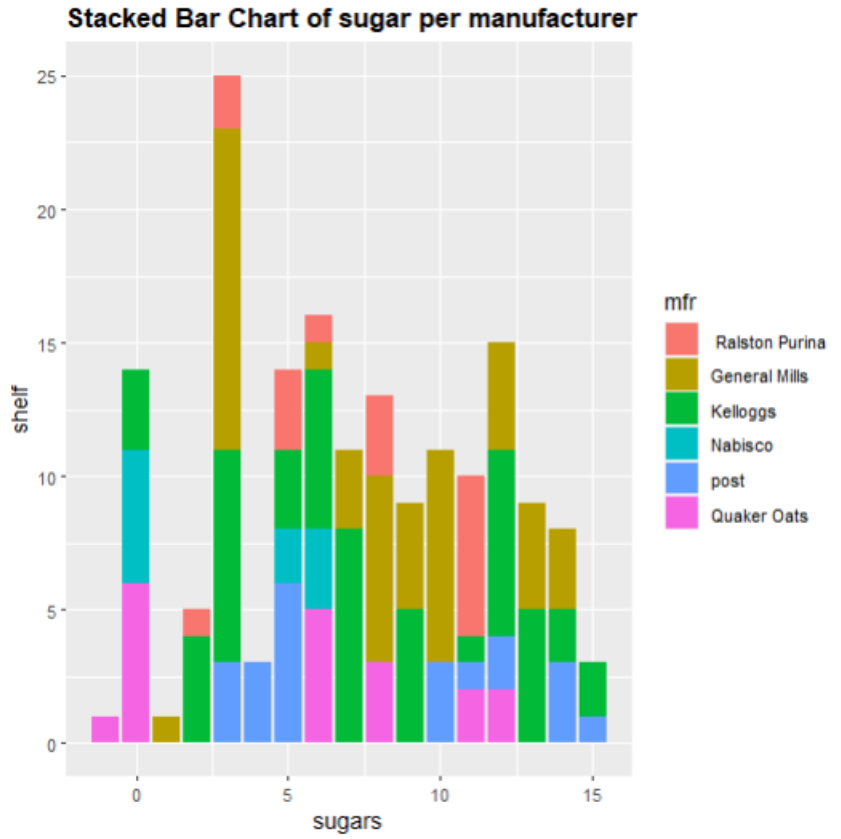
```
ggplot(df_123, aes(x = cups, y = calories, color = cups)) +
  geom_line(size=2)+
  geom_point(size=4)+
ggtitle("Line Chart of Calories per cup")+
theme(plot.title=element_text(face = "bold"))
```



- *ggplot(df_123, aes(x = cups, y = calories, color = cups)): This initializes a ggplot object using the dataframe df_123 as the data source. It specifies that the "cups" column should be mapped to the x-axis, the "calories" column to the y-axis, and the "cups" column should also determine the color of the lines and points.*
- *geom_line(size=2): This adds a line layer to the plot, representing the relationship between cups and calories. The size parameter sets the thickness of the lines to 2.*
- *geom_point(size=4): This adds a point layer to the plot, showing individual data points for each combination of cups and calories. The size parameter sets the size of the points to 4.*
- *ggtitle("Line Chart of Calories per cup"): This sets the title of the plot to "Line Chart of Calories per cup".*
- *theme(plot.title=element_text(face = "bold")): This modifies the theme of the plot. Specifically, it sets the title text to be in bold font.*

 # utilize ggplot2 to create a stacked bar chart to visualize the distribution of sugar content per manufacturer, using the dataframe `df_filtered`.

```
ggplot(df_filtered, aes(fill=mfr, y=shelf, x=sugars)) +
  geom_bar(position='stack', stat='identity')+
  ggtitle("Stacked Bar Chart of sugar per manufacturer")+
  theme(plot.title=element_text(face = "bold"))
```



- *ggplot(df_filtered, aes(fill = mfr, y = shelf, x = sugars)): Initializes a ggplot object using the dataframe df_filtered as the data source. It maps the "sugars" variable to the x-axis, the "shelf" variable to the y-axis, and the "mfr" variable to the fill color aesthetic. This means that each bar will be divided into segments based on the manufacturer, and the height of each segment will represent the shelf value.*
- *geom_bar(position = 'stack', stat = 'identity'): Adds a bar layer to the plot. The position = 'stack' parameter stacks the bars on top of each other, and stat = 'identity' indicates that the height of the bars should be directly proportional to the values in the data.*
- *ggtitle("Stacked Bar Chart of sugar per manufacturer"): Sets the title of the plot to "Stacked Bar Chart of sugar per manufacturer".*

- *theme(plot.title = element_text(face = "bold")): Modifies the theme of the plot, specifically setting the title text to be in bold font.*
- *Overall, this code generates a stacked bar chart where each bar represents the total sugar content per manufacturer, and each segment within the bar represents a different shelf value. The color of each segment indicates the manufacturer.*

 # group by manufacturer and create a new column of avg ratio of protein/carbohydrate and summarize average of this ratio

```
df <- df_filtered %>%
  group_by(mfr)%>%
  mutate(ratio =protein/carbo)%>%
  summarise(avg_ratio=mean(ratio))
df
```

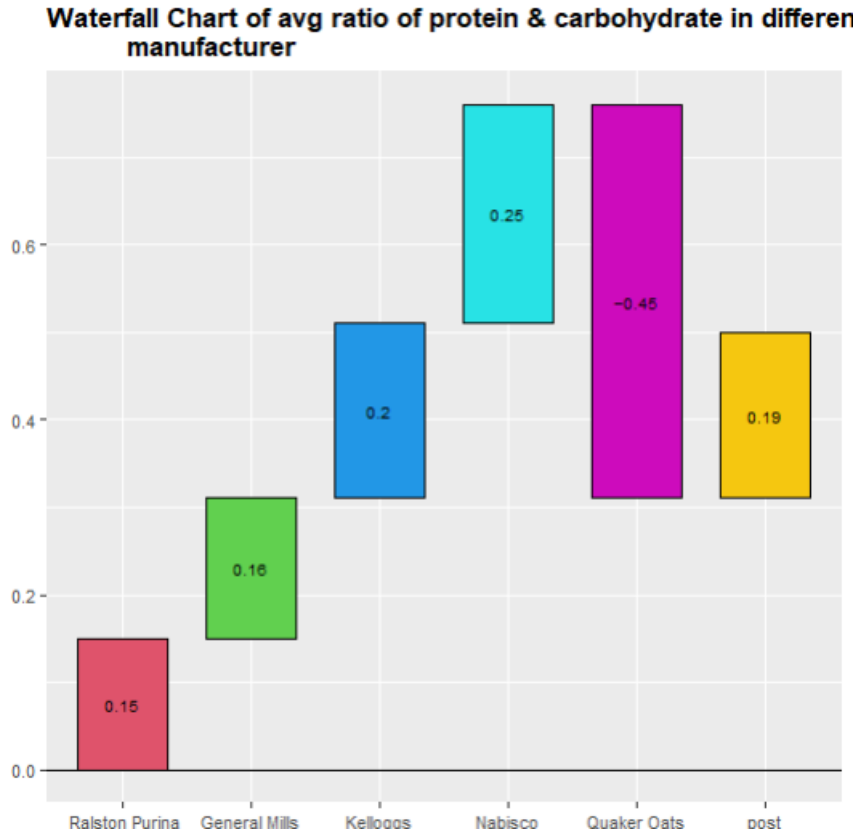
	mfr	avg_ratio
	<chr>	<dbl>
1	" Ralston Purina"	0.150
2	"General Mills"	0.162
3	"Kelloggs"	0.199
4	"Nabisco"	0.252
5	"Quaker Oats"	-0.445
6	"post"	0.188

>

- *df <- df_filtered %>% ...: This code assumes there's a dataframe named df_filtered. It pipes (%>%) this dataframe into a series of operations using the dplyr package.*
- *group_by(mfr): Groups the data by the unique values in the "mfr" column. This means that subsequent operations will be applied separately within each group of manufacturers.*
- *mutate(ratio = protein/carbo): Calculates a new column called "ratio" by dividing the "protein" column by the "carbo" column for each row within each group. This calculates the ratio of protein to carbohydrates for each cereal.*
- *summarise(avg_ratio = mean(ratio)): Calculates the mean of the "ratio" column within each group of manufacturers. This calculates the average ratio of protein to carbohydrates for all cereals within each manufacturer group.*
- *df: Displays the resulting dataframe df, which contains the average ratio of protein to carbohydrates for each manufacturer group.*
- *In summary, this code calculates the average ratio of protein to carbohydrates for cereals within each manufacturer group and stores the results in a dataframe called df.*
- *Here Nabisco has the highest protein/carbs ratios whereas Quaker Oats has a negative ratio.*

 # utilize the "waterfalls" package to create a waterfall chart visualizing the average ratio of protein to carbohydrates for different manufacturers.

```
install.packages("waterfalls")
library(waterfalls)
waterfall(values = round(df$avg_ratio,2),labels=df$mfr, draw_lines = FALSE, fill_by_sign = FALSE,
      fill_colours = 2:7)+
  ggtitle("Waterfall Chart of avg ratio of protein & carbohydrate in different
      manufacturer")+
  theme(plot.title=element_text(face = "bold"))
```



- *install.packages("waterfalls"): This command installs the "waterfalls" package. It's typically executed only once to install the package if it's not already installed.*
- *library(waterfalls): This loads the "waterfalls" package into the R session, making its functions available for use.*
- *waterfall(...): This function creates a waterfall chart. The parameters passed to the function are as follows:*
- *values = round(df$avg_ratio, 2): These are the values to be displayed in the waterfall chart, representing the average ratio of protein to carbohydrates for each manufacturer. The values are rounded to two decimal places.*
- *labels = df$mfr: These are the labels for each bar in the waterfall chart, representing the manufacturer names.*
- *draw_lines = FALSE: This parameter specifies whether to draw lines between bars in the waterfall chart. In this case, lines are not drawn.*
- *fill_by_sign = FALSE: This parameter specifies whether to fill bars with different colors based on their direction (positive or negative). Here, bars are not filled by sign.*
- *fill_colours = 2:7: This parameter specifies the colors to fill the bars in the waterfall chart. The colors are specified as a sequence from the second to the seventh color in the default palette.*
- *ggtitle("Waterfall Chart of avg ratio of protein & carbohydrate in different manufacturer"): This sets the title of the plot to "Waterfall Chart of avg ratio of protein & carbohydrate in different manufacturer".*
- *theme(plot.title=element_text(face = "bold")): This modifies the theme of the plot, specifically setting the title text to be in bold font.*
- *Overall, this code generates a waterfall chart visualizing the average ratio of protein to carbohydrates for different manufacturers, with each bar representing a manufacturer and the height of the bar indicating the average ratio.*
- *Top of Form*

#find z-score for each data value

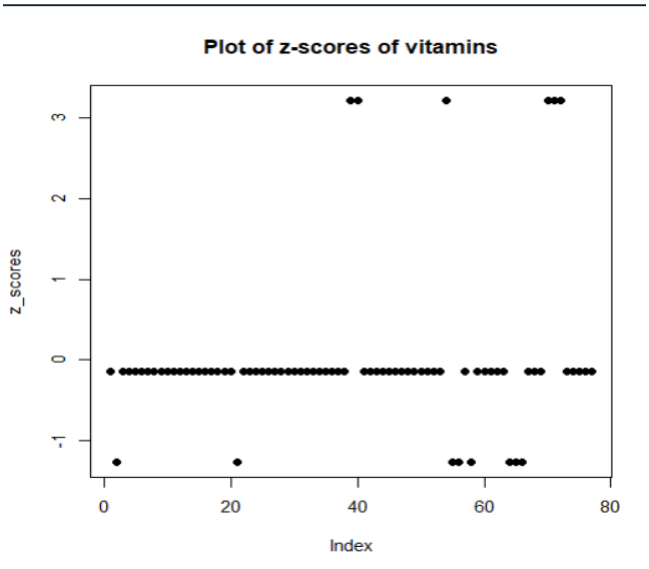
```
z_scores <- (cereal_data$vitamins-mean(cereal_data$vitamins))/sd(cereal_data$vitamins)
#display z-scores
z_scores
```

```
> z_scores
 [1] -0.1453172 -1.2642598 -0.1453172 -0.1453172 -0.1453172 -0.1453172
 [7] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
[13] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
[19] -0.1453172 -0.1453172 -1.2642598 -0.1453172 -0.1453172 -0.1453172
[25] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
[31] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
[37] -0.1453172 -0.1453172  3.2115106  3.2115106 -0.1453172 -0.1453172
[43] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
[49] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172  3.2115106
[55] -1.2642598 -1.2642598 -0.1453172 -1.2642598 -0.1453172 -0.1453172
[61] -0.1453172 -0.1453172 -0.1453172 -1.2642598 -1.2642598 -1.2642598
[67] -0.1453172 -0.1453172 -0.1453172  3.2115106  3.2115106  3.2115106
[73] -0.1453172 -0.1453172 -0.1453172 -0.1453172 -0.1453172
>
```

- *(cereal_data$vitamins - mean(cereal_data$vitamins)): This part calculates the deviation of each value in the "vitamins" column from the mean of the "vitamins" column in the cereal_data dataframe. This is achieved by subtracting the mean of the "vitamins" column (mean(cereal_data$vitamins)) from each individual value in the "vitamins" column (cereal_data$vitamins).*
- */ sd(cereal_data$vitamins): After calculating the deviations, this part of the code further standardizes them by dividing each deviation by the standard deviation of the "vitamins" column (sd(cereal_data$vitamins)). This step ensures that the resulting z-scores have a mean of 0 and a standard deviation of 1.*
- *z_scores: This variable stores the resulting z-scores calculated in the previous steps. It contains the standardized values for the "vitamins" variable in the cereal_data dataframe.*
- *In summary, the code calculates the z-scores for the "vitamins" variable, providing a standardized measure of how each observation deviates from the mean in terms of standard deviations.*

 # plotting the z-score

```
plot(z_scores,color="black",pch=16, main="Plot of z-scores of vitamins")
```



- *z_scores: This is the vector containing the z-scores calculated for the "vitamins" variable. It serves as the data for the plot.*
- *color = "black": This parameter specifies the color of the points in the scatter plot. In this case, all points will be plotted in black.*
- *pch = 16: This parameter sets the type of plotting symbol (point character) to be used in the plot. The value 16 corresponds to solid circles.*

- *main = "Plot of z-scores of vitamins": This parameter sets the main title of the plot to "Plot of z-scores of vitamins".*
- *Overall, this code generates a scatter plot where each point represents an observation's z-score for the "vitamins" variable. The points are plotted as solid black circles, and the plot is titled "Plot of z-scores of vitamins".*
- *This plot shows that majority of the z-scores between 0.145372.*