Batch: A   Roll No.: 16010123012

Experiment No. 4

Title: Apply descriptive statistics techniques in R to summarize and interpret data.

Aim: To apply various descriptive statistics techniques, such as measures of central tendency, variability, and distribution, to analyze and summarize the key features of a dataset.

Course Outcomes:

CO1, CO2, CO3

Books/ Journals/ Websites referred:

1. The Comprehensive R Archive Network
2. Posit

Resources used:
https://www.rdocumentation.org/
https://www.w3schools.com/r/
https://www.geeksforgeeks.org/r-programming-language-introduction/

Select a built-in R dataset

You can see a list of all the built-in datasets using the data() function.

```
> data()
```

```
R data sets

Data sets in package 'datasets':

AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
BJsales.lead (BJsales)
                        Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide Uptake in Grass Plants
ChickWeight             Weight versus age of chicks on different diets
DNase                   Elisa assay of DNase
EuStockMarkets          Daily Closing Prices of Major European Stock
                        Indices, 1991-1998
Formaldehyde            Determination of Formaldehyde
HairEyeColor            Hair and Eye Color of Statistics Students
```

Here, we'll use the built-in R data set named *iris*. Every student in the batch has to choose a unique dataset.

```
> # Store the data in the variable my_data
> my_data <- iris
```

Check your data

You can inspect your data using the functions **head**() and **tails**(), which will display the first and the last part of the data, respectively.

```
> # Print the first 6 rows
> head(my_data, 6)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

R functions for computing descriptive statistics

Description	R function
Mean	**mean**()
Standard deviation	**sd**()
Variance	**var**()

Minimum	**min()**
Maximum	**maximum()**
Median	**median()**
Range of values (minimum and maximum)	**range()**
Sample quantiles	**quantile()**
Generic function	**summary()**
Interquartile range	**IQR()**

Descriptive statistics for a single group

Measure of central tendency: mean, median, mode

```
> # Compute the mean value
> mean(my_data$Sepal.Length)
[1] 5.843333
>
> # Compute the median value
> median(my_data$Sepal.Length)
[1] 5.8
```

The function mfv() [in the modeest R package] can be used to compute the mode of a variable.

```
> # Compute the mode
> install.packages("modeest")
```

```
> require(modeest)
Loading required package: modeest
> mfv(my_data$Sepal.Length)
[1] 5
```

Measure of variability

Range: minimum & maximum

```
> # Compute the minimum value
> min(my_data$Sepal.Length)
[1] 4.3
> # Compute the maximum value
> max(my_data$Sepal.Length)
[1] 7.9
> # Range
> range(my_data$Sepal.Length)
[1] 4.3 7.9
```

Quantiles

```
> quantile(my_data$Sepal.Length)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

By default, the function returns the minimum, the maximum and three **quartiles** (the 0.25, 0.50 and 0.75 quantiles).

```
> quantile(my_data$Sepal.Length, seq(0, 1, 0.25))
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

To compute deciles (0.1, 0.2, 0.3, …., 0.9), use this:

```
> quantile(my_data$Sepal.Length, seq(0, 1, 0.1))
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
4.30 4.80 5.00 5.27 5.60 5.80 6.10 6.30 6.52 6.90 7.90
```

Interquartile range

```
> IQR(my_data$Sepal.Length)
[1] 1.3
```

Variance and standard deviation

```
> # Compute the variance
> var(my_data$Sepal.Length)
[1] 0.6856935
> # Compute the standard deviation =
> # square root of th variance
> sd(my_data$Sepal.Length)
[1] 0.8280661
```

Median absolute deviation

```
> # Compute the median absolute deviation
> mad(my_data$Sepal.Length)
[1] 1.03782
```

Computing an overall summary of a variable and an entire data frame

summary() function

Summary of a single variable. Five values are returned: the mean, median, 25th and 75th quartiles, min and max in one single line call:

```
> summary(my_data$Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

Summary of a data frame. In this case, the function **summary**() is automatically applied to each column. The format of the result depends on the type of the data contained in the column. For example:

- If the column is a numeric variable, mean, median, min, max and quartiles are returned.
- If the column is a factor variable, the number of observations in each group is returned.

```
> summary(my_data, digits = 2)
  Sepal.Length  Sepal.Width   Petal.Length  Petal.Width        Species
 Min.   :4.3   Min.   :2.0   Min.   :1.0   Min.   :0.1   setosa    :50
 1st Qu.:5.1   1st Qu.:2.8   1st Qu.:1.6   1st Qu.:0.3   versicolor:50
 Median :5.8   Median :3.0   Median :4.3   Median :1.3   virginica :50
 Mean   :5.8   Mean   :3.1   Mean   :3.8   Mean   :1.2
 3rd Qu.:6.4   3rd Qu.:3.3   3rd Qu.:5.1   3rd Qu.:1.8
 Max.   :7.9   Max.   :4.4   Max.   :6.9   Max.   :2.5
```

sapply() function

It's also possible to use the function **sapply**() to apply a particular function over a list or vector. For instance, we can use it to compute for each column in a data frame, the mean, sd, var, min, quantile, …

```
> # Compute the mean of each column
> sapply(my_data[, -5], mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

```
> # Compute quartiles
> sapply(my_data[, -5], quantile)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
0%            4.3         2.0         1.00         0.1
25%           5.1         2.8         1.60         0.3
50%           5.8         3.0         4.35         1.3
75%           6.4         3.3         5.10         1.8
100%          7.9         4.4         6.90         2.5
```

Graphical display of distributions

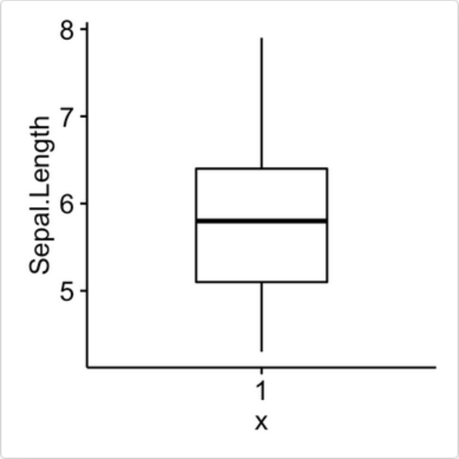
The R package **ggpubr** will be used to create graphs.

```
install.packages("ggpubr")
```

```
library(ggpubr)
```

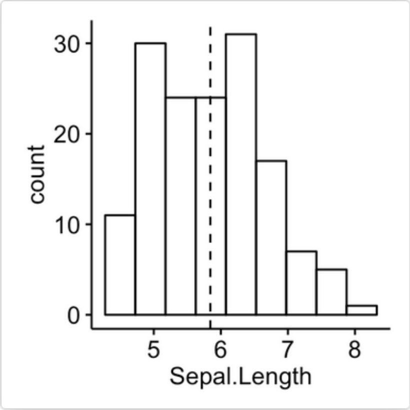
Box Plot

```
ggboxplot(my_data, y = "Sepal.Length", width = 0.5)
```



Histogram with mean line

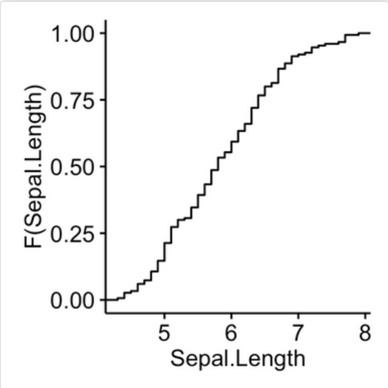
```
gghistogram(my_data, x = "Sepal.Length", bins = 9,
            add = "mean")
```



Empirical cumulative distribution function (ECDF)

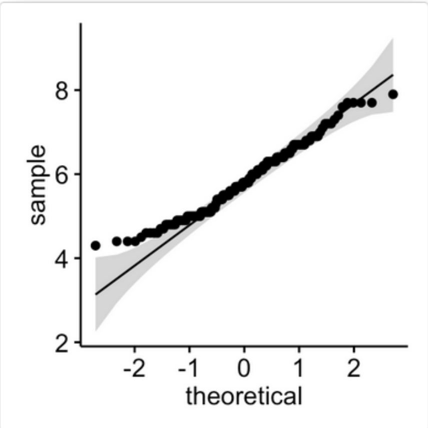
ECDF is the fraction of data smaller than or equal to x.

```
ggecdf(my_data, x = "Sepal.Length")
```



QQ plots are used to check whether the data is normally distributed.

```
ggqqplot(my_data, x = "Sepal.Length")
```



Descriptive statistics by groups

To compute summary statistics by groups, the functions **group_by**() and **summarise**() [in **dplyr** package] can be used.

- We want to group the data by *Species* and then:
    - compute the number of element in each group. R function: **n**()
    - compute the mean. R function **mean**()
    - and the standard deviation. R function **sd**()

Install **ddplyr** as follow:

```
install.packages("dplyr")
```

Descriptive statistics by groups:

To compute summary statistics by groups, the functions **group_by**() and **summarise**() [in **dplyr** package] can be used.

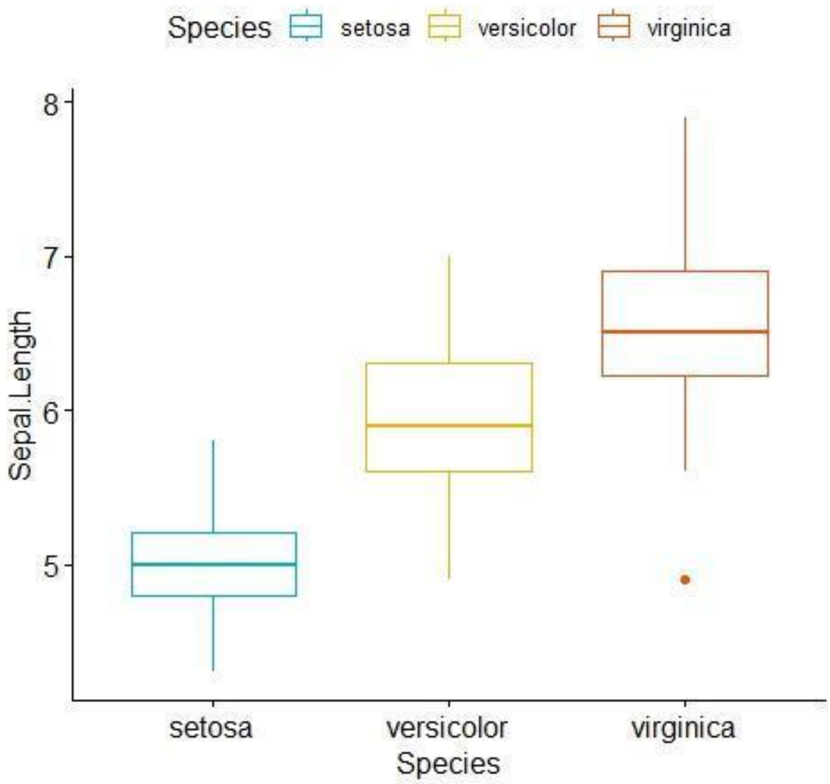
- We want to group the data by *Species* and then:
    - compute the number of element in each group. R function: **n**()
    - compute the mean. R function **mean**()
    - and the standard deviation. R function **sd**()

%>% is used to chain the operations.

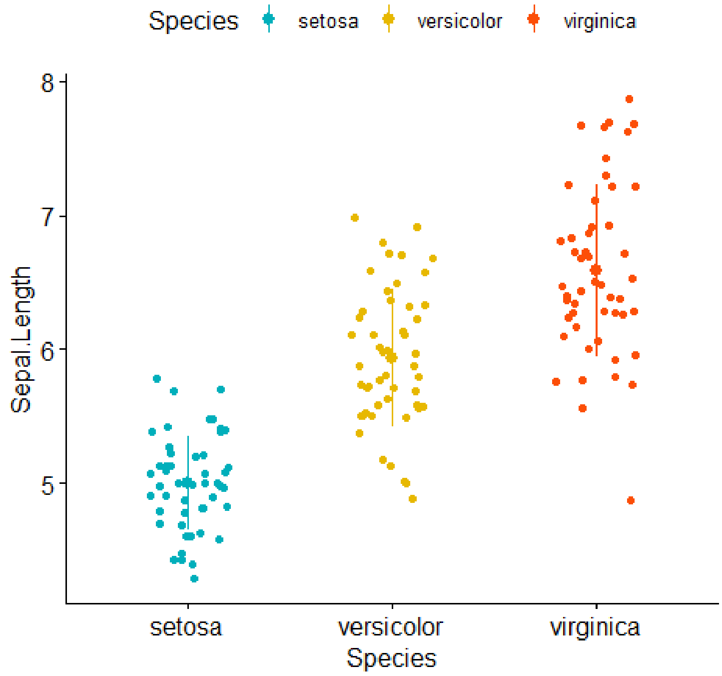
```
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> group_by(my_data, Species) %>%
+     summarise(
+         count = n(),
+         mean = mean(Sepal.Length, na.rm = TRUE),
+         sd = sd(Sepal.Length, na.rm = TRUE)
+     )
# A tibble: 3 x 4
  Species    count  mean    sd
  <fct>      <int> <dbl> <dbl>
1 setosa        50  5.01 0.352
2 versicolor    50  5.94 0.516
3 virginica     50  6.59 0.636
> 
```

Graphics for grouped data:

```
> # Box plot colored by groups: Species
> ggboxplot(my_data, x = "Species", y = "Sepal.Length",
+           color = "Species",
+           palette = c("#00AFBB", "#E7B800", "#FC4E07"))
```




```
> # Stripchart colored by groups: Species
> ggstripchart(my_data, x = "Species", y = "Sepal.Length",
+              color = "Species",
+              palette = c("#00AFBB", "#E7B800", "#FC4E07"),
+              add = "mean_sd")
```



Frequency tables

A frequency table (or contingency table) is used to describe categorical variables. It contains the counts at each combination of factor levels.

R function to generate tables: **table**()

For this section we will use the built-in R dataset that contains the distribution of hair and eye color by sex of 592 students:

```
> # Hair/eye color data
> df <- as.data.frame(HairEyeColor)
> hair_eye_col <- df[rep(row.names(df), df$Freq), 1:3]
> rownames(hair_eye_col) <- 1:nrow(hair_eye_col)
> head(hair_eye_col)
   Hair   Eye  Sex
1 Black Brown Male
2 Black Brown Male
3 Black Brown Male
4 Black Brown Male
5 Black Brown Male
6 Black Brown Male
```

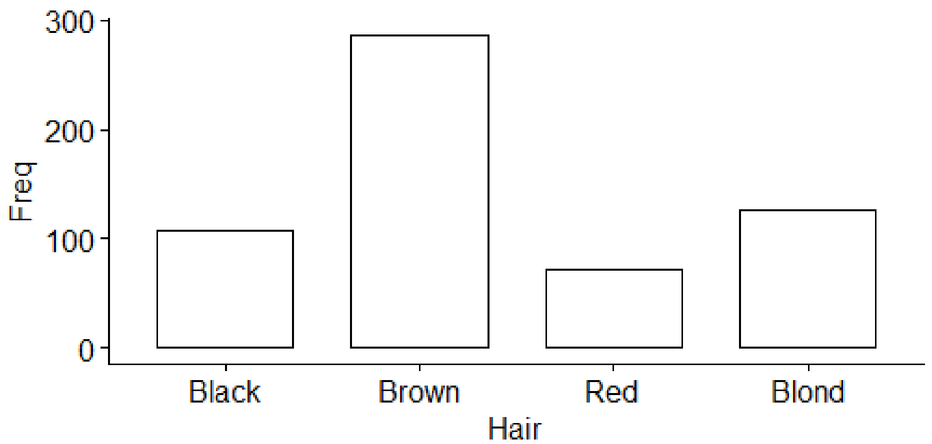
Simple frequency distribution: one categorical variable

Table of counts

```
> # hair/eye variables
> Hair <- hair_eye_col$Hair
> Eye <- hair_eye_col$Eye
> # Frequency distribution of hair color
> table(Hair)
Hair
Black Brown   Red Blond
  108   286    71   127
> # Frequency distribution of eye color
> table(Eye)
Eye
Brown  Blue Hazel Green
  220   215    93    64
```

Visualization:

```
> # Visualize using bar plot
> library(ggpubr)
> ggbarplot(df, x = "Hair", y = "Freq")
```



Two-way contingency table: Two categorical variables

```
> hair_eye <- table(Hair , Eye)
> hair_eye
        Eye
Hair     Brown Blue Hazel Green
  Black     68   20    15     5
  Brown    119   84    54    29
  Red       26   17    14    14
  Blond      7   94    10    16
```

Multiway tables: More than two categorical variables

- Hair and Eye color distributions by sex using **xtabs**():

```
> xtabs(~Hair + Eye + Sex, data = hair_eye_col)
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

You can also use the function **ftable**() [for flat contingency tables]. It returns a cleaner looking output compared to xtabs() when you have more than two variables:

```
> ftable(Sex + Hair ~ Eye, data = hair_eye_col)
      Sex   Male                    Female
      Hair Black Brown Red Blond  Black Brown Red Blond
Eye
Brown         32    53  10     3     36    66  16     4
Blue          11    50  10    30      9    34   7    64
Hazel         10    25   7     5      5    29   7     5
Green          3    15   7     8      2    14   7     8
```

Compute table margins and relative frequency

Table margins correspond to the sums of counts along rows or columns of the table.

```
> # Margin of rows
> margin.table(hair_eye, 1)
Hair
Black Brown   Red Blond
  108   286    71   127

> # Margin of columns
> margin.table(hair_eye, 2)
Eye
Brown  Blue Hazel Green
  220   215    93    64
```

Relative frequencies express table entries as proportions of table margins (i.e., row or column totals).

```
> # Frequencies relative to row total
> prop.table(hair_eye, 1)
       Eye
Hair         Brown       Blue      Hazel      Green
  Black 0.62962963 0.18518519 0.13888889 0.04629630
  Brown 0.41608392 0.29370629 0.18881119 0.10139860
  Red   0.36619718 0.23943662 0.19718310 0.19718310
  Blond 0.05511811 0.74015748 0.07874016 0.12598425
> # Table of percentages
> round(prop.table(hair_eye, 1), 2)*100
       Eye
Hair    Brown Blue Hazel Green
  Black    63   19    14     5
  Brown    42   29    19    10
  Red      37   24    20    20
  Blond     6   74     8    13
> # Table of percentages
> round(prop.table(hair_eye, 1), 4)*100
       Eye
Hair    Brown  Blue Hazel Green
  Black 62.96 18.52 13.89  4.63
  Brown 41.61 29.37 18.88 10.14
  Red   36.62 23.94 19.72 19.72
  Blond  5.51 74.02  7.87 12.60
```

Students have to perform **ALL** the operations shown above on a different dataset of their choice (unique within the lab-batch) and add their screenshots here. Students can use datasets inbuilt within R, or students can download and use freely available datasets from Kaggle, UCI repository, etc.

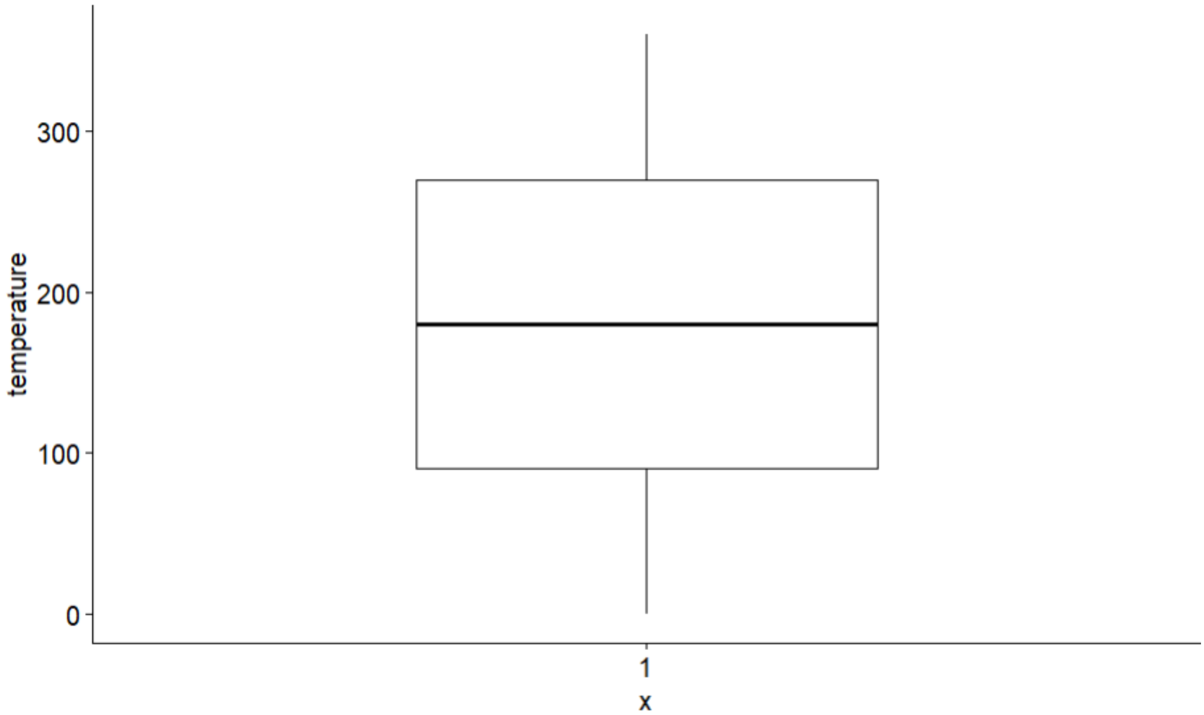
Lab work (Temperature):

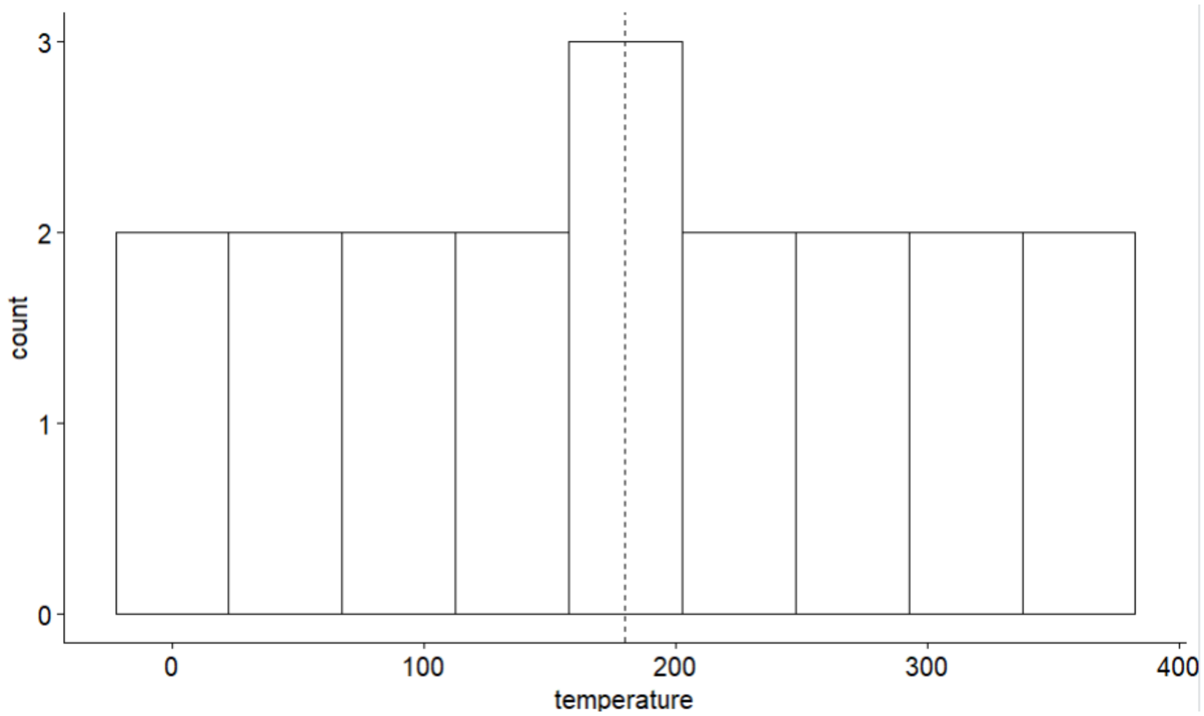
```
> data()
> my_data <- pressure
> head(my_data, 9)
  temperature pressure
1           0   0.0002
2          20   0.0012
3          40   0.0060
4          60   0.0300
5          80   0.0900
6         100   0.2700
7         120   0.7500
8         140   1.8500
9         160   4.2000
> tail(my_data, 9)
   temperature pressure
11         200     17.3
12         220     32.1
13         240     57.0
14         260     96.0
15         280    157.0
16         300    247.0
17         320    376.0
18         340    558.0
19         360    806.0
> mean(my_data$temperature)
[1] 180
> median(my_data$temperature)
[1] 180
> install.packages("modeest")
package 'timeDate' successfully unpacked and MD5 sums checked
package 'timeSeries' successfully unpacked and MD5 sums checked
package 'gss' successfully unpacked and MD5 sums checked
package 'rmutil' successfully unpacked and MD5 sums checked
package 'clue' successfully unpacked and MD5 sums checked
package 'fBasics' successfully unpacked and MD5 sums checked
package 'stable' successfully unpacked and MD5 sums checked
package 'stabledist' successfully unpacked and MD5 sums checked
package 'statip' successfully unpacked and MD5 sums checked
package 'modeest' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Aaryan\AppData\Local\Temp\RtmpG26gGz\downloaded_packages
```

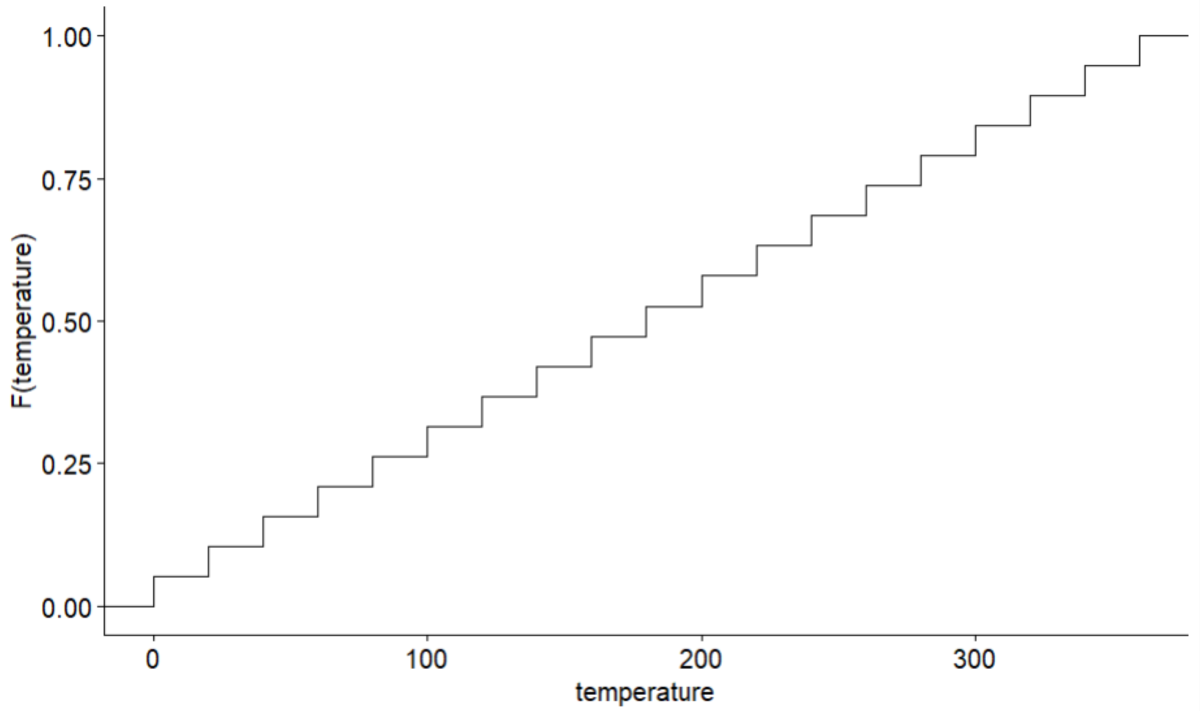
```
> mfv(my_data$temperature)
 [1]   0  20  40  60  80 100 120 140 160 180 200 220 240 260 280 300 320 340 360
> min(my_data$temperature)
[1] 0
> max(my_data$temperature)
[1] 360
> range(my_data$temperature)
[1]   0 360
> quantile(my_data$temperature)
  0%  25%  50%  75% 100% 
   0   90  180  270  360 
> quantile(my_data$temperature, seq(0, 1, 0.25))
  0%  25%  50%  75% 100% 
   0   90  180  270  360 
> quantile(my_data$temperature, seq(0, 1, 0.1))
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100% 
   0   36   72  108  144  180  216  252  288  324  360 
> IQR(my_data$temperature)
[1] 180
> var(my_data$temperature)
[1] 12666.67
> sd(my_data$temperature)
[1] 112.5463
> mad(my_data$temperature)
[1] 148.26
> summary(my_data$temperature)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      0      90     180     180     270     360 
> summary(my_data, digits = 2)
  temperature     pressure     
 Min.   :  0   Min.   :2.0e-04  
 1st Qu.: 90   1st Qu.:1.8e-01  
 Median :180   Median :8.8e+00  
 Mean   :180   Mean   :1.2e+02  
 3rd Qu.:270   3rd Qu.:1.3e+02  
 Max.   :360   Max.   :8.1e+02  
> sapply(my_data[, -5], mean)
temperature    pressure 
   180.0000    124.3367 
> sapply(my_data[, -5], quantile)
     temperature pressure
0%             0   0.0002
25%           90   0.1800
50%          180   8.8000
75%          270 126.5000
100%         360 806.0000
> install.packages("ggpubr")
```

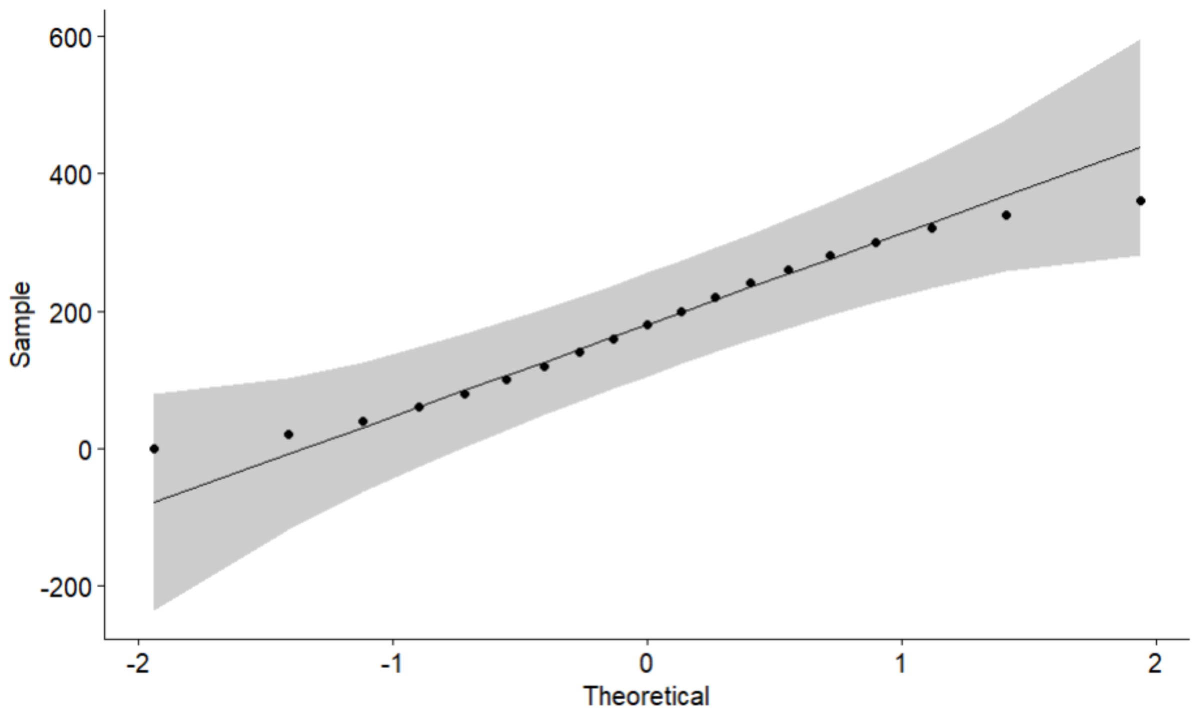
```
package ‘rbibutils’ successfully unpacked and MD5 sums checked
package ‘Deriv’ successfully unpacked and MD5 sums checked
package ‘modelr’ successfully unpacked and MD5 sums checked
package ‘microbenchmark’ successfully unpacked and MD5 sums checked
package ‘Rdpack’ successfully unpacked and MD5 sums checked
package ‘numDeriv’ successfully unpacked and MD5 sums checked
package ‘doBy’ successfully unpacked and MD5 sums checked
package ‘SparseM’ successfully unpacked and MD5 sums checked
package ‘MatrixModels’ successfully unpacked and MD5 sums checked
package ‘minqa’ successfully unpacked and MD5 sums checked
package ‘nloptr’ successfully unpacked and MD5 sums checked
package ‘reformulas’ successfully unpacked and MD5 sums checked
package ‘RcppEigen’ successfully unpacked and MD5 sums checked
package ‘stringi’ successfully unpacked and MD5 sums checked
package ‘utf8’ successfully unpacked and MD5 sums checked
package ‘colorspace’ successfully unpacked and MD5 sums checked
package ‘backports’ successfully unpacked and MD5 sums checked
package ‘carData’ successfully unpacked and MD5 sums checked
package ‘abind’ successfully unpacked and MD5 sums checked
package ‘Formula’ successfully unpacked and MD5 sums checked
package ‘pbkrtest’ successfully unpacked and MD5 sums checked
package ‘quantreg’ successfully unpacked and MD5 sums checked
package ‘lme4’ successfully unpacked and MD5 sums checked
package ‘gtable’ successfully unpacked and MD5 sums checked
package ‘isoband’ successfully unpacked and MD5 sums checked
package ‘vctrs’ successfully unpacked and MD5 sums checked
package ‘withr’ successfully unpacked and MD5 sums checked
package ‘Rcpp’ successfully unpacked and MD5 sums checked
package ‘stringr’ successfully unpacked and MD5 sums checked
package ‘tidyselect’ successfully unpacked and MD5 sums checked
package ‘cpp11’ successfully unpacked and MD5 sums checked
package ‘generics’ successfully unpacked and MD5 sums checked
package ‘pillar’ successfully unpacked and MD5 sums checked
package ‘farver’ successfully unpacked and MD5 sums checked
package ‘labeling’ successfully unpacked and MD5 sums checked
package ‘munsell’ successfully unpacked and MD5 sums checked
package ‘RColorBrewer’ successfully unpacked and MD5 sums checked
package ‘viridisLite’ successfully unpacked and MD5 sums checked
package ‘broom’ successfully unpacked and MD5 sums checked
package ‘corrplot’ successfully unpacked and MD5 sums checked
package ‘car’ successfully unpacked and MD5 sums checked
package ‘fansi’ successfully unpacked and MD5 sums checked
package ‘pkgconfig’ successfully unpacked and MD5 sums checked
package ‘ggplot2’ successfully unpacked and MD5 sums checked
package ‘ggrepel’ successfully unpacked and MD5 sums checked
package ‘ggsci’ successfully unpacked and MD5 sums checked
package ‘tidyr’ successfully unpacked and MD5 sums checked
package ‘purrr’ successfully unpacked and MD5 sums checked
package ‘dplyr’ successfully unpacked and MD5 sums checked
package ‘cowplot’ successfully unpacked and MD5 sums checked
package ‘ggsignif’ successfully unpacked and MD5 sums checked
package ‘scales’ successfully unpacked and MD5 sums checked
package ‘gridExtra’ successfully unpacked and MD5 sums checked
package ‘polynom’ successfully unpacked and MD5 sums checked
package ‘rstatix’ successfully unpacked and MD5 sums checked
package ‘tibble’ successfully unpacked and MD5 sums checked
package ‘ggpubr’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Aaryan\AppData\Local\Temp\RtmpG26gGz\downloaded_packages
```

```
> library(ggpubr)
Loading required package: ggplot2
> ggboxplot(my_data, y = "Temperature", width = 0.5)
Error:
! Problem while computing aesthetics.
i Error occurred in the 1st layer.
Caused by error:
! object 'Temperature' not found
Run `rlang::last_trace()` to see where the error occurred.
> ggboxplot(my_data, y = "temperature", width = 0.5)
> gghistogram(my_data, x = "temperature", bins = 9, add = "mean")
Warning messages:
1: `geom_vline()`: Ignoring `mapping` because `xintercept` was provided.
2: `geom_vline()`: Ignoring `data` because `xintercept` was provided.
> ggecdf(my_data, x = "temperature")
> ggqqplot(my_data, x = "temperature")
```


300
200
100
0
temperature
1
x


3
2
1
0
count
0
100
200
300
400
temperature

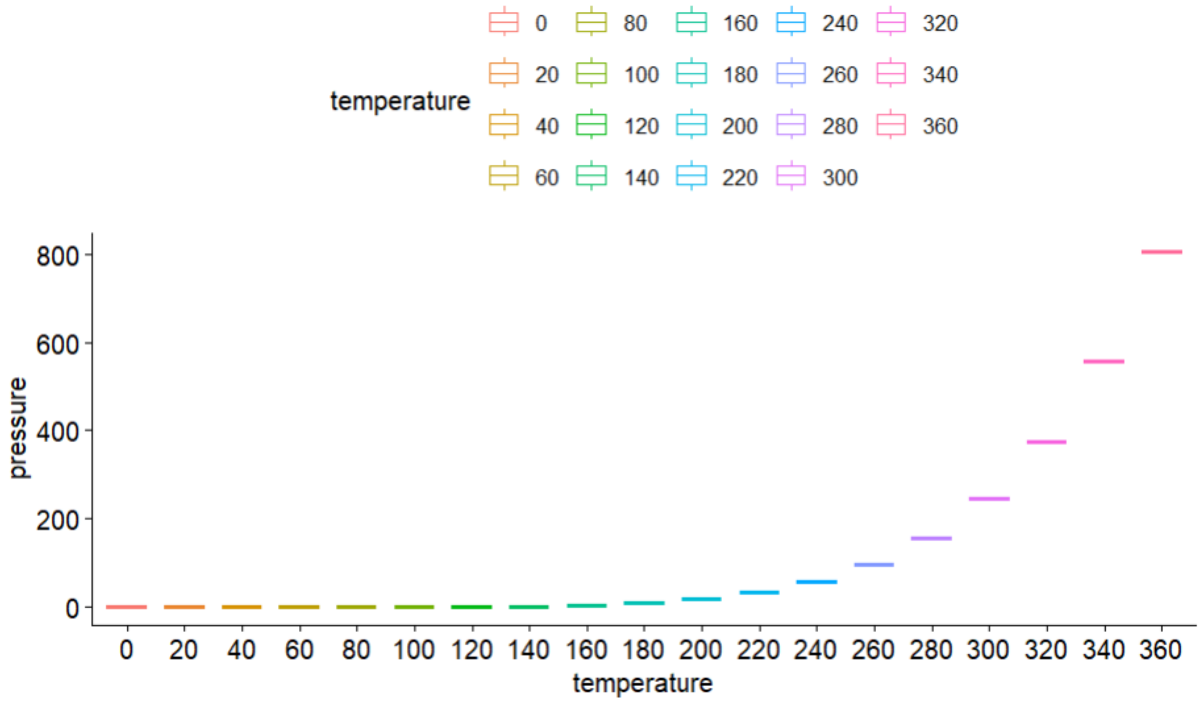


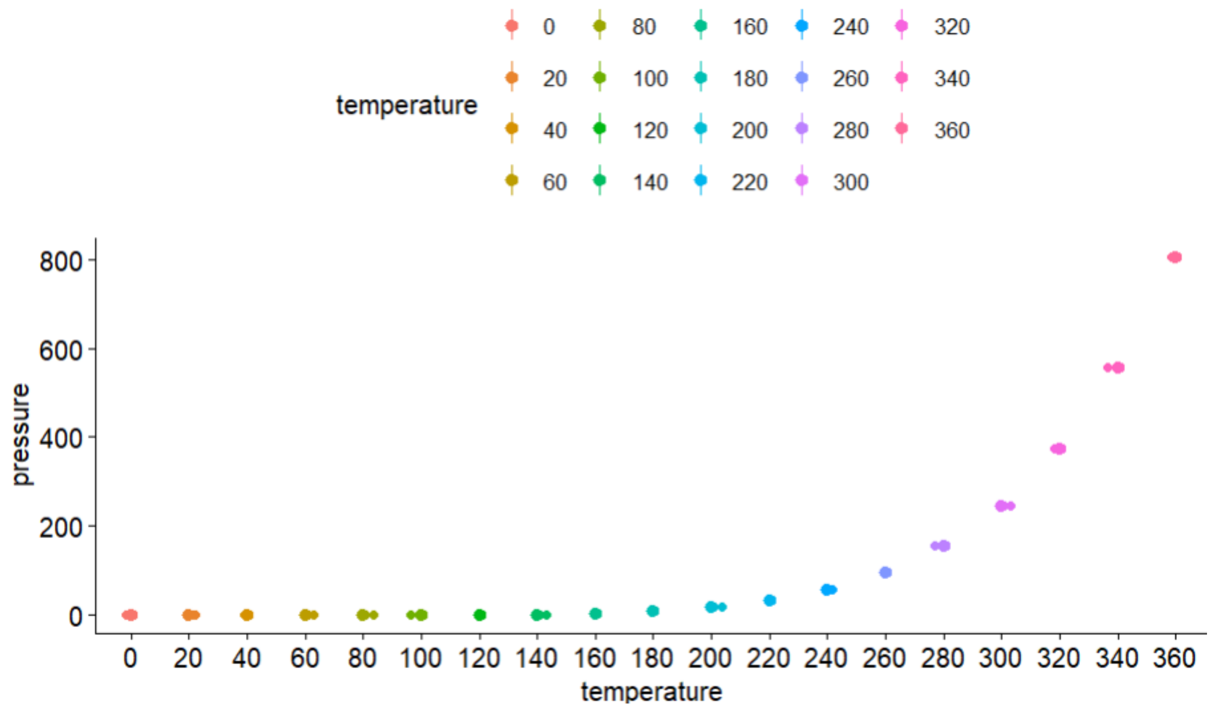



```
> install.packages("dplyr")
```

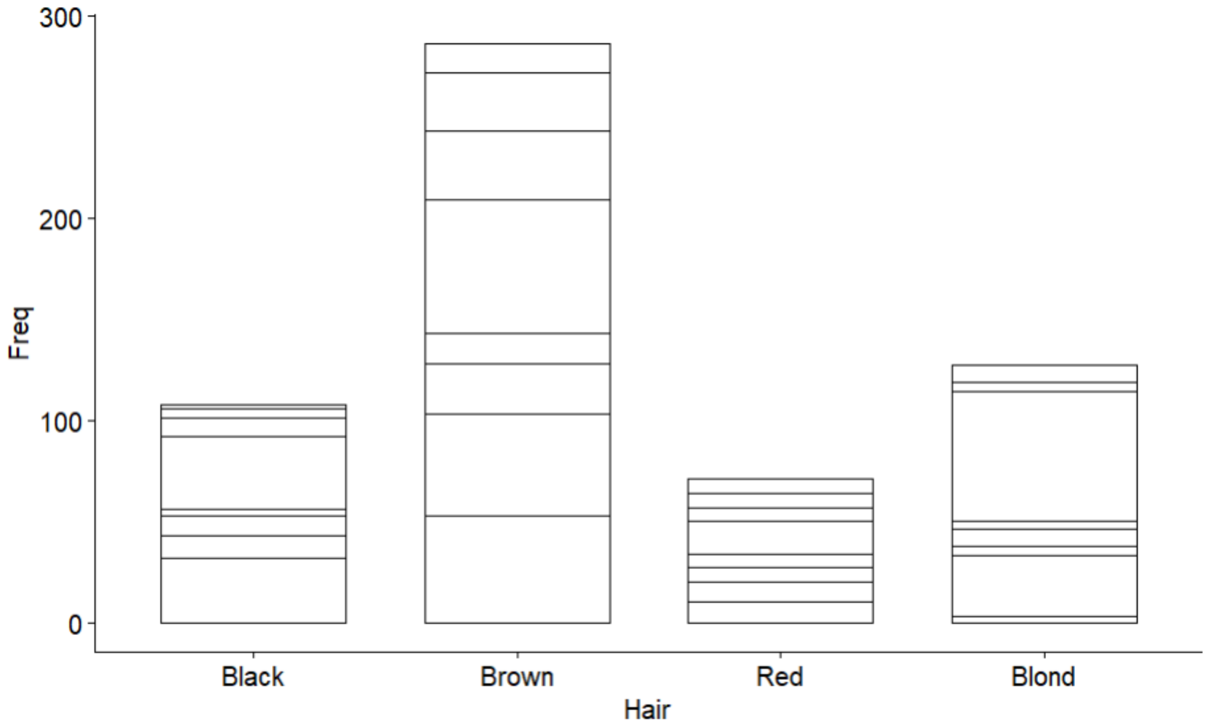
```
The downloaded binary packages are in
        C:\Users\Aaryan\AppData\Local\Temp\RtmpETok8B\downloaded_packages
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> group_by(my_data, temperature)
# A tibble: 19 × 2
# Groups:   temperature [19]
   temperature pressure
         <dbl>    <dbl>
 1           0   0.0002
 2          20   0.0012
 3          40   0.006
 4          60   0.03
 5          80   0.09
 6         100   0.27
 7         120   0.75
 8         140   1.85
 9         160   4.2
10         180   8.8
11         200  17.3
12         220  32.1
13         240  57
14         260  96
15         280 157
16         300 247
17         320 376
18         340 558
19         360 806
```

```
> group_by(my_data, temperature) %>%
+     summarise(
+         count = n(),
+         mean = mean(temperature, na.rm = TRUE),
+         sd = sd(temperature, na.rm = TRUE)
+     )
# A tibble: 19 × 4
   temperature count  mean    sd
         <dbl> <int> <dbl> <dbl>
 1           0     1     0    NA
 2          20     1    20    NA
 3          40     1    40    NA
 4          60     1    60    NA
 5          80     1    80    NA
 6         100     1   100    NA
 7         120     1   120    NA
 8         140     1   140    NA
 9         160     1   160    NA
10         180     1   180    NA
11         200     1   200    NA
12         220     1   220    NA
13         240     1   240    NA
14         260     1   260    NA
15         280     1   280    NA
16         300     1   300    NA
17         320     1   320    NA
18         340     1   340    NA
19         360     1   360    NA
> ggboxplot(my_data, x = "temperature", y = "pressure",
+           color = "temperature",
+           palette = c("#2E8B57", "#FFC300", "#3399FF"))
> ggstripchart(my_data, x = "temperature", y = "pressure",
+              color = "temperature",
+              palette = c("#2E8B57", "#FFC300", "#3399FF"),
+              add = "mean_sd")
```


temperature
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
pressure
800
600
400
200
0
temperature


temperature
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
pressure
800
600
400
200
0
temperature

```
> df <- as.data.frame(HairEyeColor)
> hair_eye <- df[rep(row.names(df), df$Freq), 1:4]
> rownames(hair_eye) <- 1:nrow(hair_eye)
> head(hair_eye)
   Hair   Eye  Sex Freq
1 Black Brown Male   32
2 Black Brown Male   32
3 Black Brown Male   32
4 Black Brown Male   32
5 Black Brown Male   32
6 Black Brown Male   32
> Hair <- hair_eye$Hair
> Eye <- hair_eye$Eye
> table(Hair)
Hair
Black Brown   Red Blond
  108   286    71   127
> table(Eye)
Eye
Brown  Blue Hazel Green
  220   215    93    64
> library(ggpubr)
> ggbarplot(df, x = "Hair", y = "Freq")
```



```
> he <- table(Hair, Eye)
> he
       Eye
Hair    Brown Blue Hazel Green
  Black    68   20    15     5
  Brown   119   84    54    29
  Red      26   17    14    14
  Blond     7   94    10    16
> xtabs(~Hair + Eye + Sex, data = hair_eye)
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

```
> ftable(Sex + Hair ~ Eye, data = hair_eye)
      Sex    Male                        Female
      Hair Black Brown Red Blond  Black Brown Red Blond
Eye
Brown         32    53  10     3     36    66  16     4
Blue          11    50  10    30      9    34   7    64
Hazel         10    25   7     5      5    29   7     5
Green          3    15   7     8      2    14   7     8
> margin.table(he, 1)
Hair
Black Brown   Red Blond
  108   286    71   127
> margin.table(he, 2)
Eye
Brown  Blue Hazel Green
  220   215    93    64
> prop.table(he, 1)
       Eye
Hair         Brown       Blue      Hazel      Green
  Black 0.62962963 0.18518519 0.13888889 0.04629630
  Brown 0.41608392 0.29370629 0.18881119 0.10139860
  Red   0.36619718 0.23943662 0.19718310 0.19718310
  Blond 0.05511811 0.74015748 0.07874016 0.12598425
> round(prop.table(he, 1), 2) * 100
       Eye
Hair    Brown Blue Hazel Green
  Black    63   19    14     5
  Brown    42   29    19    10
  Red      37   24    20    20
  Blond     6   74     8    13
> round(prop.table(he, 1), 4) * 100
       Eye
Hair    Brown  Blue Hazel Green
  Black 62.96 18.52 13.89  4.63
  Brown 41.61 29.37 18.88 10.14
  Red   36.62 23.94 19.72 19.72
  Blond  5.51 74.02  7.87 12.60
```

Conclusion:
I have successfully completed this experiment and gained a comprehensive understanding of working with data sets, including their analysis and implementation in R. Through this experiment, I learned how to apply descriptive statistics techniques in R to summarize and interpret data. Using charts, graphs, tables, I was able to represent data in a structured and meaningful way. This analysis allowed me to effectively describe datasets, making them easier to understand and interpret for decision-making and further statistical exploration.

Post Lab questions

1. Critically assess the limitations of using only measures of central tendency in data analysis.
2. Compare and contrast the different measures of variability, with the focus on when one measure might be more informative than the other.
3. Imagine you are presented with a dataset from a research study. Discuss how applying descriptive statistics techniques could aid in understanding the key features and trends in the data. Take any real life examples to aid your analysis.

Aaryan Sharma
16010123012

IDS Exp 4

1) i) Ignores Data Variability: Measures of central tendency summarize the typical value of a dataset but do not indicate how datapts vary.
ii) Cannot detect Outliners: Mean is affected by extremes, median is robust and mode may be unreliable.
iii) Loss of Distribution Information: They fail to capture spread, skewness or outlines.
iv) Limited in Comparing Datasets: Same mean, doesn't mean similar dist.
v) Misleading Data: Mean is meaningless for categorical data.

2) i) Range is used for getting a quick view of the spread, range()
ii) Variance is used for spread in terms of squared derivations, var()
iii) S.D is more interpretable cause of same units as data, sd().
iv) IQR is bet for spread without outliners, IQR().

3) Descriptive statistics help summarise and interpret data, making it easier to identify key patterns and trends. These techniques include central tendency, variability and visual Representation.
A real life example could be a dataset collected on average daily screen time to teenagers.
i) Mean & Median: These could help to understand the average screen time, & median would provide insight into the middle of the dataset, showing if there are any outliners.
ii) S.D & Range: They tell us about how much variation there is in screen time among teenagers
iii) Frequency Distribution: By creating a histogram or frequency table we can see how many teenagers fall in different categories of screen time.