Session 1: Data Exploration and Visualization

Tingfang Wang

June 2024

Contents

Data Exploration and Visualization with tidyverse 1
- Introduction . . . 1
  - Install tidyverse package . . . 2
  - Data sets . . . 3
- Topics in this session . . . 6
  - Data Types . . . 7
  - Data Sturcture . . . 7
  - Descriptive Measures . . . 10
  - Data Grouping . . . 11
  - Histograms and Bar Charts . . . 14
  - Boxplots . . . 16
  - Data Simulation . . . 18
  - Sampling Distribution . . . 19
- Exercise . . . 23
- Conclusion . . . 23
- Further Reading and Resources . . . 23

Data Exploration and Visualization with tidyverse

Introduction

R is a powerful programming language widely used for data manipulation, analysis, and visualization. The tidyverse is a collection of R packages that provide a unified and efficient framework for working with data. This tutoring guide will introduce you to the basics of R programming and the core tidyverse packages, demonstrating their usage through code examples.

Install tidyverse package

To get started with the `tidyverse`, you need to install the necessary packages. Open R or RStudio and run the following command:

```
install.packages("tidyverse")
```

This will install the `tidyverse` package along with its dependencies. Once installed, you can load the `tidyverse` library using the following command:

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.3

## Warning: package 'ggplot2' was built under R version 4.3.3

## Warning: package 'tibble' was built under R version 4.3.3

## Warning: package 'tidyr' was built under R version 4.3.3

## Warning: package 'readr' was built under R version 4.3.3

## Warning: package 'purrr' was built under R version 4.3.3

## Warning: package 'dplyr' was built under R version 4.3.3

## Warning: package 'stringr' was built under R version 4.3.3

## Warning: package 'forcats' was built under R version 4.3.3

## Warning: package 'lubridate' was built under R version 4.3.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

This one line of code loads the core tidyverse packages that you will use in almost every data analysis. It also tells you which functions from the tidyverse conflict with functions in base R or from other packages you might have loaded.

If we need to explicit about where a function or dataset comes form, we will use the special form `package::function()` or `package::dataset_name`. For example, `ggplot2::ggplot()` indicates that we are using the `ggplot()` function from the `ggplot2` package, and `ggplot2::mpg` indicates we are using the `mpg` dataset from the `gglot2` package.

Data sets

To explore the basic data manipulation, analysis, and visualization, we will use three build-in datasets in R.

1. mpg dataset:

   The `mpg` dataset is a built-in dataset in R that contains information about fuel economy (miles per gallon) for various car models. It is part of the `ggplot2` package and provides a useful dataset for practicing data analysis and visualization tasks. The dataset includes the following variables:

   - manufacturer: Manufacturer of the car.
   - model: Model name of the car.
   - displ: Engine displacement in liters.
   - year: Year the car was manufactured.
   - cyl: Number of cylinders.
   - trans: Transmission type.
   - drv: Drive train type (f = front-wheel drive, r = rear-wheel drive, 4 = 4-wheel/all-wheel drive).
   - cty: City miles per gallon.
   - hwy: Highway miles per gallon.
   - fl: Fuel type.
   - class: Vehicle class.

   You can access the mpg dataset by loading the `tidyverse/ggplot2` package in R and using the `mpg` object. Here's an example:

```r
# Load tidyverse package
library(tidyverse)

# Access the mpg dataset
data(mpg)

# Display the structure of the dataset
str(mpg)
```

```
## tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
##  $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
##  $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
##  $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr [1:234] "f" "f" "f" "f" ...
##  $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr [1:234] "p" "p" "p" "p" ...
##  $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

```r
# View the first few rows of the dataset
head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
```

```
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

```
# To download the data (change the file path)
# write.csv(mpg,
#           file = "C:/Users/tingf/Box/TA/BootCamp/mpg.csv",
#           row.names = FALSE)
```

2. flights dataset:

   The `flights` dataset is another built-in dataset in R that contains information about flights departing from three major airports in the United States in the year 2013. It is part of the `nycflights13` package and provides a useful dataset for practicing data analysis and modeling tasks related to flights. The dataset includes the following variables:

   - year: Year of the flight.
   - month: Month of the flight.
   - day: Day of the flight.
   - dep_time: Departure time (local, hhmm).
   - sched_dep_time: Scheduled departure time (local, hhmm).
   - dep_delay: Departure delay (minutes).
   - arr_time: Arrival time (local, hhmm).
   - sched_arr_time: Scheduled arrival time (local, hhmm).
   - arr_delay: Arrival delay (minutes).
   - carrier: Airline carrier code.
   - flight: Flight number.
   - tailnum: Aircraft tail number.
   - origin: Origin airport code.
   - dest: Destination airport code.
   - air_time: Flight time (minutes).
   - distance: Distance traveled (miles).
   - hour: Departure hour.
   - minute: Departure minute.
   - time_hour: Date and time of departure.

   You can access the `flights` dataset by loading the `nycflights13` package in R and using the `flights` object. Here's an example:

```
#install nycflights13 package
#install.packages("nycflights13")

# Load nycflights13 package
library(nycflights13)

# Access the flights dataset
data(flights)

# Display the structure of the dataset
str(flights)
```

```
## tibble [336,776 x 19] (S3: tbl_df/tbl/data.frame)
```

```
## $ year          : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
## $ month         : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ day           : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ dep_time      : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
## $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
## $ dep_delay     : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
## $ arr_time      : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
## $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
## $ arr_delay     : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
## $ carrier       : chr [1:336776] "UA" "UA" "AA" "B6" ...
## $ flight        : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
## $ tailnum       : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
## $ origin        : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
## $ dest          : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
## $ air_time      : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
## $ distance      : num [1:336776] 1400 1416 1089 1576 762 ...
## $ hour          : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
## $ minute        : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
## $ time_hour     : POSIXct[1:336776], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

```
# View the first few rows of the dataset
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# To download the data (change the file path)
# write.csv(flights,
#           file = "C:/Users/tingf/Box/TA/BootCamp/flights.csv",
#           row.names = FALSE)
```

3. Iris dataset:

   The `datasets` package in R provides a collection of built-in datasets that are commonly used for learning and practicing data analysis techniques. For example, the Iris dataset, a popular dataset in the field of machine learning and data analysis is in this package. It consists of measurements of four features of iris flowers (sepal length, sepal width, petal length, and petal width) and the corresponding species (setosa, versicolor, and virginica).

   You can access the `iris` dataset by loading the `datasets` package in R and using the `iris` object. Here's an example:

```
#install datasets package if not yet
#install.packages("datasets")
```

```r
# Load datasets package
library(datasets)

# Access the iris dataset
data(iris)

# Display the structure of the dataset
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
# View the first few rows of the dataset
head(iris)
```

```
##   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
# To download the data (change the file path)
# write.csv(iris,
#           file = "C:/Users/tingf/Box/TA/BootCamp/iris.csv",
#           row.names = FALSE)
```

Topics in this session

This document provides a tutorial on using the tidyverse package in R for data manipulation, descriptive measures, and visualization. We will cover the following topics:

- Data Types
- Descriptive Measures
- Data Grouping
- Histograms and Bar Charts
- Boxplots
- Sampling Distribution
- Data Simulation

Data Types

To begin, let's explore different data types commonly used in R. The `tidyverse` provides functions to handle various data types, such as numeric, character, and logical. Here's an example:

```
# Numeric
x <- 5

# Character
name <- "John Doe"

# Logical
is_true <- TRUE

# Print the data types
typeof(x)
```

```
## [1] "double"
```

```
typeof(name)
```

```
## [1] "character"
```

```
typeof(is_true)
```

```
## [1] "logical"
```

Data Sturcture

1. Vectors

   Vectors are one-dimensional arrays that can hold elements of the same data type. Here's an example of creating a vector:

   ```
   # Numeric vector
   numeric_vector <- c(1, 2, 3, 4, 5)
   # Print the numeric vector
   print(numeric_vector)
   ```

   ```
   ## [1] 1 2 3 4 5
   ```

   ```
   # Character vector
   character_vector <- c("apple", "banana", "orange")
   # Print the character vector
   print(character_vector)
   ```

   ```
   ## [1] "apple"  "banana" "orange"
   ```

   ```
   # Logical vector
   logical_vector <- c(TRUE, FALSE, TRUE)
   # Print the logical vector
   print(logical_vector)
   ```

```
## [1]  TRUE FALSE  TRUE
```

2. Matrices

   Matrices are two-dimensional arrays with rows and columns. Elements in a matrix must be of the same data type. Here's an example of creating a matrix:

```r
# Numeric matrix
numeric_matrix <- matrix(1:9, nrow = 3, ncol = 3)
numeric_matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
# Character matrix
character_matrix <- matrix(c("a", "b", "c", "d", "e", "f"), nrow = 2, ncol = 3)
character_matrix
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "e" 
## [2,] "b"  "d"  "f" 
```

3. Arrays

   Arrays are multi-dimensional extensions of matrices. They can have more than two dimensions. Here's an example of creating an array:

```r
# Numeric array
numeric_array <- array(1:24, dim = c(2, 3, 4))
numeric_array
```

```
## , , 1
## 
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## 
## , , 2
## 
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
## 
## , , 3
## 
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
## 
## , , 4
## 
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

4. Lists

   Lists are a collection of different data types or objects. Each element of a list can be of a different length or data type. Here's an example of creating a list:

```r
# List
my_list <- list(numeric_vector, character_vector, numeric_matrix)
my_list
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "apple"  "banana" "orange"
##
## [[3]]
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

5. Data Frames

   Data frames are two-dimensional structures that can store different types of data. They are similar to matrices, but columns can have different data types. Here's an example of creating a data frame:

```r
# Data frame
data_frame <- data.frame(
  name = c("John", "Alice", "Bob"),
  age = c(25, 30, 35),
  stringsAsFactors = FALSE
)
data_frame
```

```
##    name age
## 1  John  25
## 2 Alice  30
## 3   Bob  35
```

6. Factors

   Factors are used to represent categorical variables with predefined levels. They are often used for storing and analyzing categorical data. Here's an example of creating a factor:

```r
# Factor
gender <- factor(c("Male", "Female", "Male", "Female"))
gender
```

```
## [1] Male   Female Male   Female
## Levels: Female Male
```

7. Tibble

   In R, a `tibble` is a modern and enhanced version of a data frame. It is part of the tidyverse collection of packages and provides a more user-friendly and intuitive way to work with tabular data. Tibbles have become the preferred data structure for data manipulation and analysis in the tidyverse ecosystem. There are some reasons that data analysts prefer tibble over data frame, such as

- when printing a `tibble`, it only displays the first few rows and columns, providing a concise summary of the data. This feature is particularly useful when dealing with large datasets, as it helps avoid flooding the console with excessive output.
- A `tibble` preserves the data types of its columns, preventing unintended coercion that can happen in regular data frames. This behavior is crucial for maintaining data integrity and accuracy during data manipulations.

To create a tibble, you can use the `tibble()` function from the tibble package or other functions from the tidyverse packages that return tibbles. Here's an example:

```r
# Create a tibble
my_tibble <- tibble(
  name = c("John", "Alice", "Bob"),
  age = c(25, 30, 35),
  city = c("New York", "London", "Paris")
)

# Print the tibble
my_tibble
```

```
## # A tibble: 3 x 3
##   name    age city
##   <chr> <dbl> <chr>
## 1 John     25 New York
## 2 Alice    30 London
## 3 Bob      35 Paris
```

Descriptive Measures

we can use the base R function `summary()`, this function can be used with various types of objects, including vectors, data frames, matrices, and more. Here are a few examples of how the `summary()` function can be used:

1. Summarizing a numeric vector:

```r
# Create a numeric vector
my_vector <- c(1, 2, 3, 4, 5)

# Obtain the summary,
summary(my_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       2       3       3       4       5
```

The output will include the minimum, first quartile, median, mean, third quartile, and maximum values of the vector.

2. Summarizing a data frame:

```r
# Create a data frame
my_df <- data.frame(
  name = as.factor(c("John", "Alice", "Bob", "Alice", "John")),
```

```r
  age = c(25, 30, 35, 20, 40),
  city = as.factor(c("New York", "London", "Paris", "Dallas", "London"))
)

# Obtain the summary
summary(my_df)
```

```
##     name        age           city  
##  Alice:2   Min.   :20   Dallas  :1  
##  Bob  :1   1st Qu.:25   London  :2  
##  John :2   Median :30   New York:1  
##            Mean   :30   Paris   :1  
##            3rd Qu.:35               
##            Max.   :40               
```

The output will provide a summary of each column in the data frame, including the minimum, first quartile, median, mean, third quartile, and maximum values for numerical columns. For categorical variables, it will display the frequency count of each unique value.

Data Grouping

Data grouping allows us to split data into subsets based on one or more variables. The `group_by()` function from the dplyr package is commonly used for this purpose.

1. Grouping by one variable

   For the mpg dataset, suppose we want to know which manufacture has the highest average "hwy". We can do the following:

```r
# Load data
data(mpg)

# Print a few rows of data
head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

```r
# Group the data by the 'manufacturer' variable
grouped_data <- group_by(mpg, manufacturer)

# Summarize the grouped data to get average "hwy"
summarized_data <- summarise(grouped_data, mean_hwy = mean(hwy))

# View the summarized data
summarized_data
```

```
## # A tibble: 15 x 2
##    manufacturer mean_hwy
##    <chr>           <dbl>
##  1 audi             26.4
##  2 chevrolet        21.9
##  3 dodge            17.9
##  4 ford             19.4
##  5 honda            32.6
##  6 hyundai          26.9
##  7 jeep             17.6
##  8 land rover       16.5
##  9 lincoln          17
## 10 mercury          18
## 11 nissan           24.6
## 12 pontiac          26.4
## 13 subaru           25.6
## 14 toyota           24.9
## 15 volkswagen       29.2
```

we can further sort the summarized data to find the highest value more easily:

```r
# Group the data by the 'manufacturer' variable
grouped_data <- group_by(mpg, manufacturer)

# Summarize the grouped data to get average "hwy"
summarized_data <- summarise(grouped_data, mean_hwy = mean(hwy))

# Reorder the summarized data by 'mean_hwy'
arrange(summarized_data, desc(mean_hwy))
```

```
## # A tibble: 15 x 2
##    manufacturer mean_hwy
##    <chr>           <dbl>
##  1 honda            32.6
##  2 volkswagen       29.2
##  3 hyundai          26.9
##  4 audi             26.4
##  5 pontiac          26.4
##  6 subaru           25.6
##  7 toyota           24.9
##  8 nissan           24.6
##  9 chevrolet        21.9
## 10 ford             19.4
## 11 mercury          18
## 12 dodge            17.9
## 13 jeep             17.6
## 14 lincoln          17
## 15 land rover       16.5
```

2. Grouping by multiple variables.

   Suppose for the flights dataset, we want to count the number of flights in each day/month/year. We can do:

```r
# Load data
data(flights)

# Print a few rows of data
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# Group the data by 'year', 'month', and 'day' variables
daily <- group_by(flights, year, month, day)

# Count flights in each day
# .groups = "drop_last" to drop the 'day' grouping but keep 'year' and 'month'
per_day <- summarise(daily, flights = n(),
                  .groups = "drop_last")
# Each summary call removes one grouping level (since that group
# is now just a single row)
per_day
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day flights
##    <int> <int> <int>   <int>
##  1  2013     1     1     842
##  2  2013     1     2     943
##  3  2013     1     3     914
##  4  2013     1     4     915
##  5  2013     1     5     720
##  6  2013     1     6     832
##  7  2013     1     7     933
##  8  2013     1     8     899
##  9  2013     1     9     902
## 10  2013     1    10     932
## # i 355 more rows
```

```r
# Count flights in each month
# .groups = "drop_last" to drop the 'month' grouping but keep 'year'
per_month <- summarise(per_day, flights = sum(flights),
                   .groups = "drop_last")
per_month
```

```
## # A tibble: 12 x 3
```

```
## # Groups:   year [1]
##     year month flights
##    <int> <int>   <int>
##  1  2013     1   27004
##  2  2013     2   24951
##  3  2013     3   28834
##  4  2013     4   28330
##  5  2013     5   28796
##  6  2013     6   28243
##  7  2013     7   29425
##  8  2013     8   29327
##  9  2013     9   27574
## 10  2013    10   28889
## 11  2013    11   27268
## 12  2013    12   28135
```

```r
# Count flights in each year
# .groups = "drop" to remove all grouping
per_year <- summarise(per_month, flights = sum(flights),
                  .groups = "drop")
per_year
```

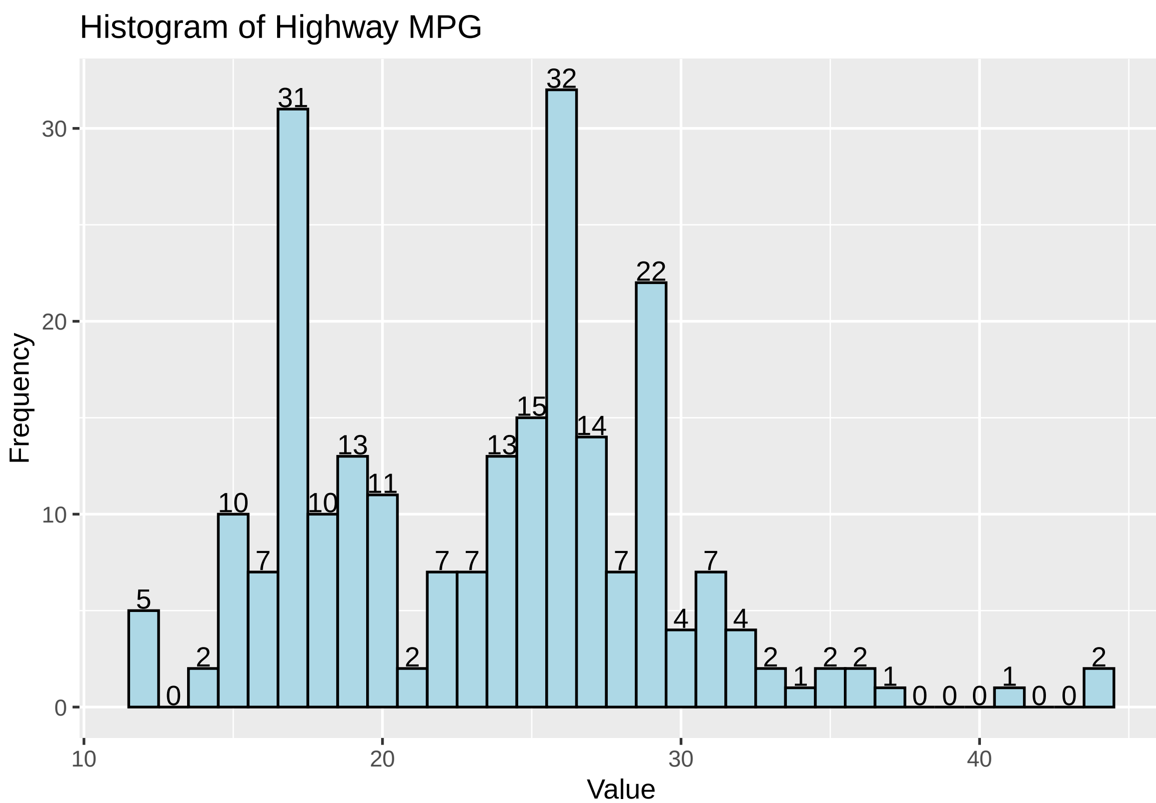
```
## # A tibble: 1 x 2
##    year flights
##   <int>   <int>
## 1  2013  336776
```

Histograms and Bar Charts

1. Histogram

   Histograms are useful for visualizing the distribution of numeric variables. The ggplot2 package provides functions to create histograms. Here's an example:

```r
# Use the mpg data from ggplot2
data(mpg)

# Create a histogram for "hwy": highway miles per gallon
histogram <- ggplot(mpg, aes(x = hwy)) +
  geom_histogram(binwidth = 1, fill = "lightblue", color = "black") +
  labs(x = "Value", y = "Frequency", title = "Histogram of Highway MPG") +
  # Add text
  stat_bin(geom = "text",
       aes(label = after_stat(count)),
       vjust = -0.1,
       binwidth = 1)

# Display the histogram
print(histogram)
```

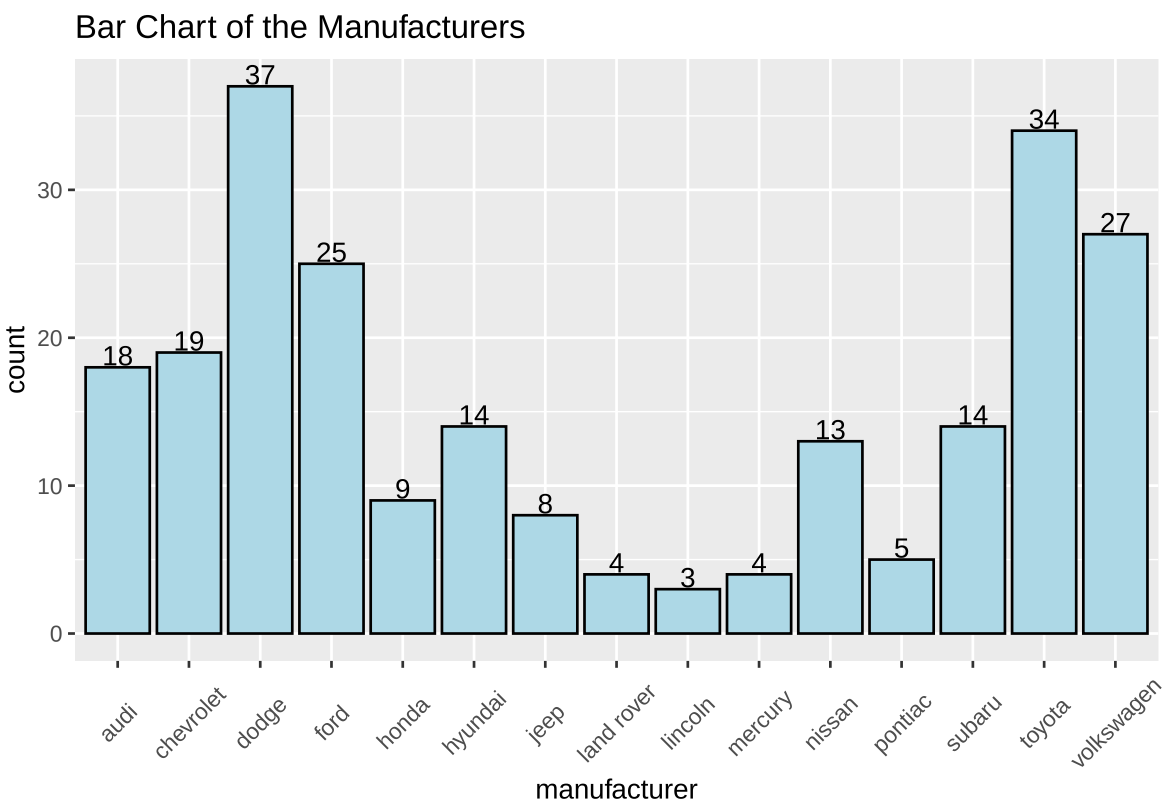


A histogram divides the x_axis into equally spaced bibs and then uses the height of each bar to display the number of observations that fall in each bin.

2. Bar Chart

   Bar charts are useful for visualizing the distribution of categorical variables. The ggplot2 package provides functions to create bar charts. Here's an example:

```r
# Use the mpg data from ggplot2
data(mpg)

# Create a histogram for "hwy": highway miles per gallon
bar_chart <- ggplot(mpg, aes(x = manufacturer)) +
  geom_bar(fill = "lightblue", color = "black") +
  labs(title = "Bar Chart of the Manufacturers") +
  # Add text
  geom_text(stat = "count", aes(label = after_stat(count)), vjust = -0.1) +
  theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 0.5))

# Print the histogram
print(bar_chart)
```

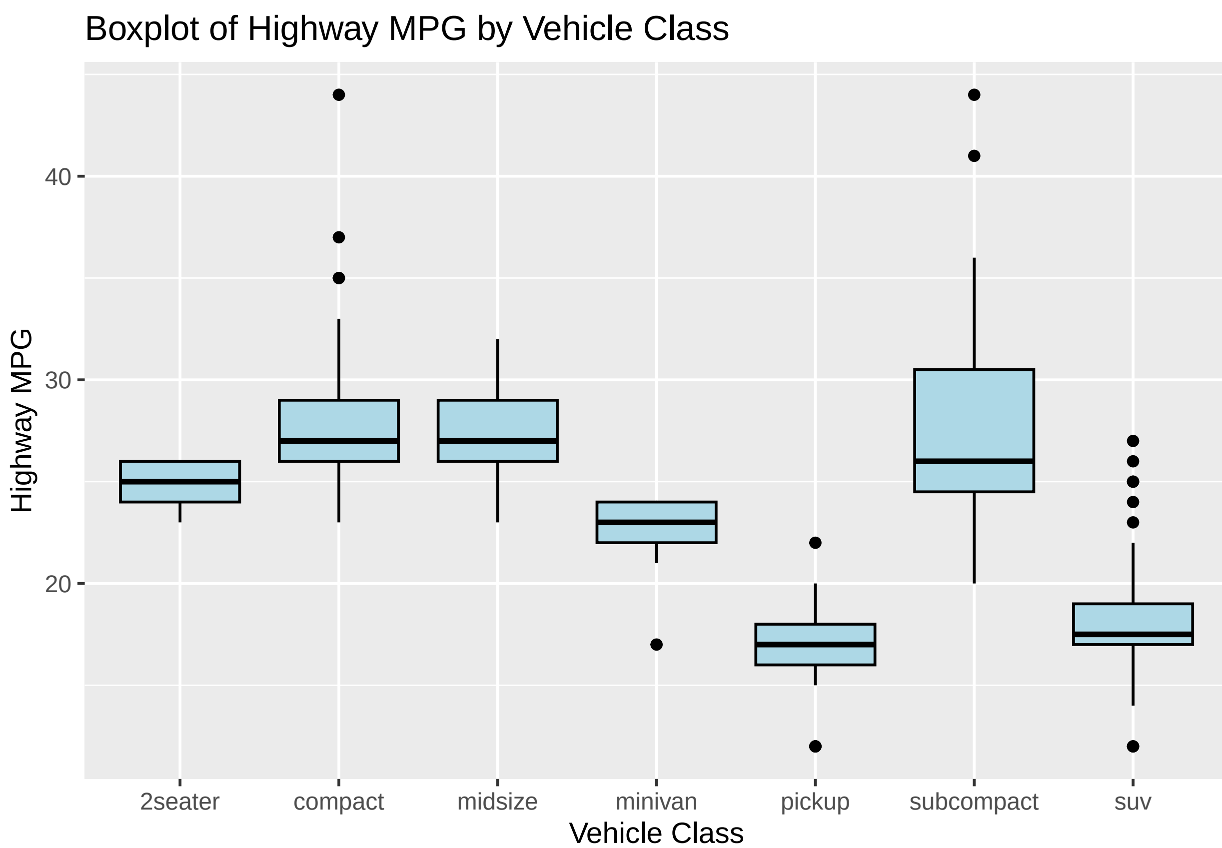


The height of the bar displays how many observations occurred with each "manufacturer".

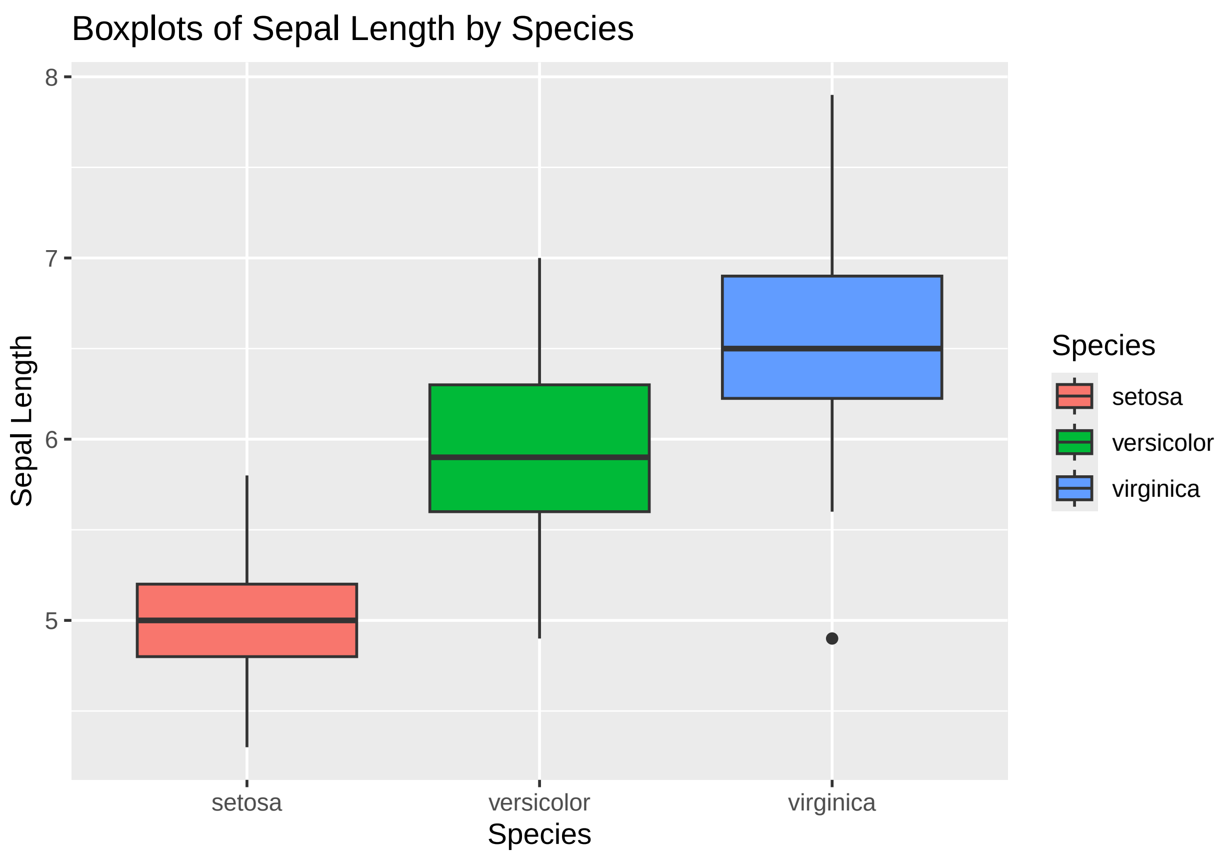
Boxplots

Boxplots display the distribution of a numeric variable and provide insights into its central tendency and spread. The ggplot2 package can be used to create boxplots. Here's an example:

```
# Create a boxplot that shows the distribution of highway miles per gallon (hwy)
# by vehicle class (class).
boxplot1 <- ggplot(mpg, aes(x = class, y = hwy)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  labs(x = "Vehicle Class", y = "Highway MPG",
       title = "Boxplot of Highway MPG by Vehicle Class")

# Print the boxplot
print(boxplot1)
```




```r
# Create boxplots that shows the distribution of sepal length in different species.
boxplot2 <- ggplot(iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_boxplot() +
  labs(x = "Species", y = "Sepal Length",
       title = "Boxplots of Sepal Length by Species")

# Print the boxplot
print(boxplot2)
```



- A box that stretches from the 25th percentile of the distribution to the 75th percentile, a distance known as the interquartile range (IQR).
- In the middle of the box is a line that displays the median, i.e., the 50th percentile of the distribution.
- Visual points that displays observations that fall more than 1.5 times the IQR from either edge of the box.
- A line (or whisker) that extends from each end of the box and goes to the farthest non-outlier point in the distribution.

Data Simulation

Simulations allow us to generate random data based on specified distributions. R provides several functions to generate random numbers from various probability distributions. Here are a few commonly used functions:

- `rnorm()`: Generates random numbers from a normal distribution.
- `runif()`: Generates random numbers from a uniform distribution.
- `rbinom()`: Generates random numbers from a binomial distribution.

To generate a random sample of 1000 numbers from a standard normal distribution:

```r
# Generate random numbers from a standard normal distribution
x = rnorm(n = 1000, mean = 0, sd = 1)
# Print the first few random numbers
head(x)
```

```
## [1]  0.1213085 -0.8795850  1.9516936  2.3570560 -0.2688141 -2.8161014
```

To generate a random sample of 1000 numbers from a uniform distribution $U[-1, 1]$:

```
# Generate random numbers from a uniform distribution 'U[-1,1]'
x = runif(n = 1000, min = -1, max = 1)
# Print the first few random numbers
head(x)
```

```
## [1] -0.71974495 -0.39407065 -0.35125158 -0.84292385  0.59348244 -0.08759785
```

To generate a random sample of 1000 numbers from a binomial distribution $B(10, 0.2)$:

```
# Generate random numbers from a uniform distribution 'U[-1,1]'
x = rbinom(n = 1000, size = 10, prob = 0.2)
# Print the first few random numbers
head(x)
```

```
## [1] 1 3 1 2 1 3
```

To generate a tibble with three columns from the standard normal distribution, the uniform distribution $U[-1, 1]$, and the binomial distribution $B(10, 0.2)$ respectively:

```
# Make this example reproducible
set.seed(1)

# Simulate data from a normal distribution
simulated_data <- tibble(
  x = rnorm(n = 100, mean = 0, sd = 1),
  y = runif(n = 100, min = 0, max = 1),
  z = rbinom(n = 100, size = 10, prob = 0.2)
)
# Print the first few rows of the simulated data
head(simulated_data)
```

```
## # A tibble: 6 x 3
##        x     y     z
##    <dbl> <dbl> <int>
## 1 -0.626 0.268     2
## 2  0.184 0.219     0
## 3 -0.836 0.517     2
## 4  1.60  0.269     2
## 5  0.330 0.181     1
## 6 -0.820 0.519     5
```

Sampling Distribution

Sampling distribution refers to the distribution of sample statistics from repeated sampling. The tidyverse package provides functions for generating sampling distributions. Here's an example of sampling from a normal distribution:

1. Sampling distribution of mean

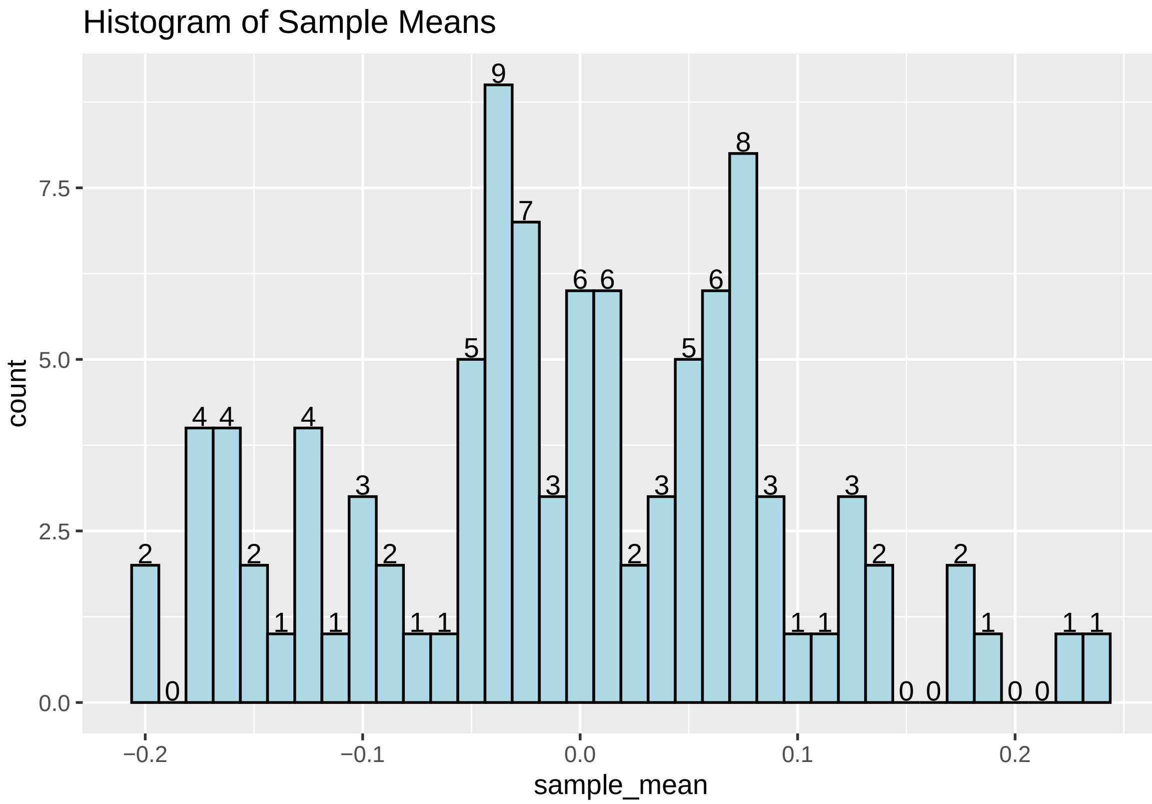
The sampling distribution of the mean is the distribution of sample means obtained from repeated sampling from a population. It provides information about the variability of sample means and approximates the population mean.

To illustrate, let's generate multiple samples and calculate the mean for each sample.

```r
# Make this example reproducible
set.seed(1)

# Generating a 100 sample means of 100 samples from a standard normal distribution
sampling_dist <- tibble(
  sample_mean = replicate(100, mean(rnorm(100, mean = 0, sd = 1)))
)

# View the sample mean of the first 5 samples
head(sampling_dist)
```

```
## # A tibble: 6 x 1
##   sample_mean
##         <dbl>
## 1      0.109
## 2     -0.0378
## 3      0.0297
## 4      0.0516
## 5     -0.0391
## 6     -0.0445
```

```r
# Create a histogram for the sample means
histogram <- ggplot(sampling_dist, aes(x = sample_mean)) +
  geom_histogram(binwidth = 0.0125, fill = "lightblue", color = "black") +
  labs(title = "Histogram of Sample Means") +
  #Add text
  stat_bin(geom = "text",
       aes(label = after_stat(count)),
       vjust = -0.1,
       binwidth = 0.0125)

# Print the histogram
print(histogram)
```



2. Central Limit Theory

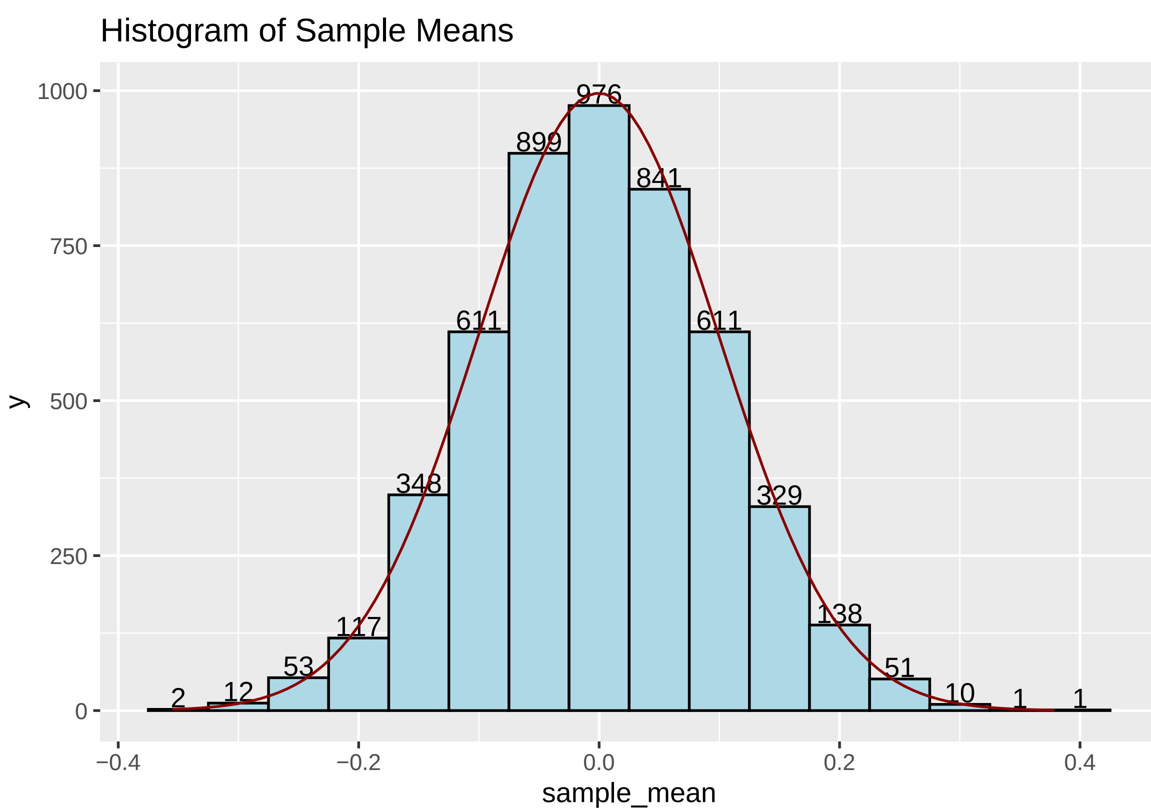
   The central limit theory states that for independent and identically distributed random variables, the sampling distribution of the standardized sample mean tends towards the standard normal distribution even if the original variables themselves are not normally distributed.

   If we increase the sample size to 5000, the histogram of the sample means should be close to a bell shape.

```r
# Make this example reproducible
set.seed(1)

# Generating a 5000 sample means of 5000 samples from a standard normal distribution
sampling_dist <- tibble(
  sample_mean = replicate(5000, mean(rnorm(100, mean = 0, sd = 1)))
)

# View the sample mean of the first 5 samples
head(sampling_dist)
```

```
## # A tibble: 6 x 1
##   sample_mean
##         <dbl>
## 1       0.109
## 2     -0.0378
## 3      0.0297
## 4      0.0516
## 5     -0.0391
## 6     -0.0445
```

```r
# Create a histogram for the sample means
histogram <- ggplot(sampling_dist, aes(x = sample_mean)) +
  geom_histogram(binwidth = 0.05, fill = "lightblue", color = "black") +
  labs(title = "Histogram of Sample Means") +
  # #Add text
  stat_bin(geom = "text",
       aes(label = after_stat(count)),
       vjust = -0.1,
       binwidth = 0.05) +
  #Add normal curve 250 = binwidth * N
  stat_function(fun = function(x) dnorm(x,
                                        mean = mean(sampling_dist$sample_mean),
                                        sd = sd(sampling_dist$sample_mean)) * 250,
           color = "darkred", linewidth = 0.5)

# Print the histogram
print(histogram)
```



Exercise

```r
# Read the survey data
survey_data <- read.csv(file = "Data/Stats_survey_Sp23_shortcourse.csv")

# Remove rows with missing values
survey_data <- na.omit(survey_data)

# Check the columns' name
colnames(survey_data)
```

```
##  [1] "Eyecolor"      "BloodType"     "BornPlace"     "Weight"
##  [5] "Height"        "NumofSiblings" "NumofTVs"      "HighSchoolGPA"
##  [9] "StarbucksRate" "HoursSleep"
```

Download and read the data file "Stats_survey_Sp23_shortcourse.csv" to R. calculate the five statistics (Mean, Median, Range, Variance, Std. Dev.) for "HighSchoolGPA" and create a histogram for "HighSchoolGPA" with the count of each bin at the top. Set the bin width to be 0.25.

Conclusion

In this tutorial, we explored various aspects of data manipulation, visualization, and statistical analysis using the `tidyverse` package and some base R functions. We covered data types, descriptive measures, data grouping, scatterplots, histograms and bar charts, boxplots, data simulation, sampling distributions, and central limit theory. These are just a few examples of what you can do with tidyverse. Feel free to explore more functions and features to further enhance your data analysis workflows.

Further Reading and Resources

To enhance your understanding and proficiency in R, consider exploring the following resources:

- Books:
  - "R for Data Science" by Hadley Wickham and Garrett Grolemund
  - "The Art of R Programming" by Norman Matloff
  - "Advanced R" by Hadley Wickham
- Online Courses and Tutorials:
  - DataCamp: https://www.datacamp.com/
  - Coursera: https://www.coursera.org/
- R Documentation and Package Manuals:
  - R Documentation: https://www.rdocumentation.org/
  - CRAN: https://cran.r-project.org/