Data manipulation in R

Surya Lamichhane

10 February 2025

Contents

Learning Objectives 1
Data Wrangling 1
1. Data Exploration in R 2
2. Subsetting Dataframes 5
3. Sorting data in R 9
4. Merging of Data Tables 11
5. dplyr Package for Data Manipulation 14
6. Data Reshaping using tydyr 19

Learning Objectives

By the end of this lesson, you will be able to:

- Understand basic data wrangling techniques
- Read and manipulate data using functions in R
- Summarize data
- Use dplyr for data manipulation
- Merge data tables to prepare data for analysis

Data Wrangling

- A data wrangling is a process of transforming and mapping data from "raw" form into more usable format
- It is used for a variety of proposes including analytics and decision making.

Data Wrangling Steps

- Discovery: It contains process of exploring the data and finding the way to solve the data analysis objective
- Structuring: The process of taking raw data and converting it into a more usable structured data that fits to the relevant study

- Cleaning: The process of removing errors that must be cleaned before it can be used. Data cleaning includes correcting outliers, deleting unreliable data to increase quality and consistency.It finds duplicate values, eliminates structural problems, and verifies data to make it easier to use.
- Enriching: This a data augmentation process dding more information to the data from other datasets that might improved the present analysis
- Validating: This is a process of ensuring that the processed data is accurate and consistent. Through this step you will ensure that the your data is ready for analysis.
- Publishing: This is a step where you finalize the data and make it available to other stakeholders.

1. Data Exploration in R

Data exploration is a critical step in understanding the structure, quality, and patterns in your dataset before performing any analysis. In this course, we are performing basic and intermediate data exploration in R.

1.1. Reading data in R

A. Working directory: The working directory is a default location or path of any files to be read into R, or saved out of R.

- Get the current working directory: *getwd( )*
- Set the new working directory: *setwd("/Useres/admin/. . . .")* in mac
- Set the new working directory: *setwd("C:/Useres/admin/. . . .")* in windows
- Make sure the working directory changed: *getwd( )*

B. Reading csv file:

```
file_need_to_read <- read.csv("C:/path/to/your/file.csv", header = TRUE)# in windows
file_need_to_read <- read.csv("/path/to/your/file.csv", header = TRUE)# in mac
```

```
cars_data <- read.csv("~/Desktop/STAT4101L_all_files/4101L-Fall-2023/cars.csv", header = TRUE)
head(cars_data, 2)
```

```
##     Make                   Model  Type Origin DriveTrain  MSRP Invoice
## 1 Subaru              Forester X Wagon               All 21445   19646
## 2 Toyota  Camry Solara SE V6 2dr Sedan   Asia      Front 21965   19819
##   EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight Wheelbase Length
## 1        2.5         4        165       21          28   3090        99    175
## 2        3.3         6        225       20          29   3417       107    193
```

C. Reading xlsx file:

- Base R cannot read an excel sheet; however,
- it can be read using read_excel function from the "readxl" package.

Here we are going to read two excel datasets:

- **First install and load the library**

```r
#install.packages("readxl") # install "readxl" package first if you have not installed yet
library(readxl) # load the library
```

```
# Syntax to read excel file
file_need_to_read <- read_excel("/path/to/your/file.xlsx")
```

Read Income data.

- This data can also be found in the Blackboard's dataset folder

```r
library(readxl)
Income_data <- read_excel("~/Desktop/STAT4101L_all_files/4101L-Fall-2023/Course Materials/Section-3 Data
print(Income_data)
```

```
## # A tibble: 10 x 4
##    storeID Sale_in_Thous Rent_in_Thous OtherIncome
##    <chr>           <dbl> <chr>         <chr>
##  1 12AR              165 7             NA
##  2 20AR              132 8             5
##  3 17AR              177 NA            4
##  4 11AR              128 NA            NA
##  5 26AR              137 5             3
##  6 18AR              199 NA            2
##  7 27AR              178 NA            6
##  8 25AR              104 6             NA
##  9 10AR              185 9             7
## 10 13AR              109 NA            1
```

D. Reading text file:

We use the read.table() function to import text data. It is important to determine how the data is separated and whether it includes a header.

- **Tabular data**

```r
Class_marks <- read.table("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/
head(Class_marks, 3) # print first 3 rows
```

```
##   Enrol.No. Maths Science English
## 1      A101    16      15      12
## 2      A102    16      17      11
## 3      A103    12      18      17
```

- **Reading Text from a File** (Uses: Processing text data)

```r
Data_wrangling_text = readLines("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/dat
print(Data_wrangling_text) #This text file was created by copying and pasting content from Google.
```

```
## [1] "Data wrangling, sometimes referred to as data munging, is the process of transforming and mappin
```

- We can read text file into string using **readr** package

```r
# Using readr package
library(readr)
Data_wrangling_text <- read_file("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/da
Data_wrangling_text
```

```
## [1] "Data wrangling, sometimes referred to as data munging, is the process of transforming and mappi
```

```r
# find number of characters
cat("Number of characters in our text documents: ", nchar(Data_wrangling_text), "\n")
```

```
## Number of characters in our text documents:  462
```

1.2 Previewing and Understanding Data

Previewing and understanding data are crucial steps in any data analysis process. In R, there are several functions and techniques we can use to get a sense of our data after the loading. Below are some common functions:

```
head(x, n = ..)
tail(x, n = ..)
str(x, n = ..)
View(x)
nrow(x)
ncol(x)
summary(x)
```

```r
View(Income_data) #Invokes a spreadsheet style data viewer for the object
```

```r
data(iris)
head(iris, n = 5) # top 5 observation of Iris flower data from R
```

```
##   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
```

```r
tail(iris, n = 5) # bottom 5 observation of Iris flower data from R
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

1.3. Missing Values

Handling missing values effectively is essential for maintaining the integrity of your dataset. Missing values should be handled appropritely. Here are the most common practices for dealing with missing values:

- Identify Missing Values: Before handling missing data, you must identify where and how many missing values exist.
- Remove Missing Values: Sometimes it is appropriate to remove rows or columns with missing data especially when data presents excessive missing values in a row or a column.
- Impute Missing Values: Replace missing values with plausible estimates.
  - Mean or Median Imputation (for numerical data)
  - Mode Imputation (for categorical data)
- **Other practices are:**
  - Interpolation: interpolate missing values based on surrounding data.
  - K-Nearest Neighbors (KNN) Imputation: Impute missing values based on the nearest neighbors in the dataset.
  - For categorical data, you can add a new category indicating missingness.
  - Use advanced statistical or machine learning models to predict and fill missing values.

1.4. Statistical Summary

```
# Summary statistics for a single column
    summary(data$numeric_column)
# Compute specific metrics
    mean(data$numeric_column, na.rm = TRUE)
    median(data$numeric_column, na.rm = TRUE)
    sd(data$numeric_column, na.rm = TRUE)  # Standard deviation
    var(data$numeric_column, na.rm = TRUE) # Variance
```

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

2. Subsetting Dataframes

- A subset can be created by using square brackets with specifying the row index (indices) and column index (indices)
- A dollar operator can be used to extract a column if we have column names
- Conditional filtering can be used to extract subset

2.1. Conditional filtering

a. Based on single condition

- Filter the dataset that corresponds to 'setosa' Species

```
setosa_data <- iris[iris$Species == 'setosa', ]
head(setosa_data)
```

```
##   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
```

b. Based on multiple conditions

- Filter the dataset that corresponds to 'setosa' Species and Sepal.Length is between 5 and 6 cms.

```
filtered_setosa_data <- iris[iris$Species == 'setosa' &
                              5 < iris$Sepal.Length &  iris$Sepal.Length < 6,  ]
head(filtered_setosa_data)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
```

2.2. The subset() Function

The subset() function can also be used to extract subsets of data if given conditions are met.

a. single condition

- Filter the dataset that corresponds to 'setosa' Species

```
setosa_data <- subset(iris,  Species = 'setosa')
head(setosa_data)
```

```
##   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
```

b. multiple conditions

- Filter the dataset that corresponds to 'setosa' Species and Sepal.Length is between 5 and 6 inches.

```
filtered_setosa_data <- subset(iris, Species == 'setosa' &
                                5 < Sepal.Length &  Sepal.Length < 6)
head(filtered_setosa_data)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
```

2.3. Select, drop or add varaibles in Dataframes

a. Select variables using column index

- Select Sepal.Width and Petal.Width from iris data

```
# select using the columns indices
colnames <- c("Sepal.Length","Sepal.Width", "Petal.Length","Petal.Width","Species")
iris_new1 <- iris[,  c(2, 4)]
head(iris_new1)
```

```
##   Sepal.Width Petal.Width
## 1         3.5         0.2
## 2         3.0         0.2
## 3         3.2         0.2
## 4         3.1         0.2
## 5         3.6         0.2
## 6         3.9         0.4
```

b. Select variables using names

- Select Sepal.Width and Petal.Width from iris data

```
# select using the variable names
iris_new2 <- iris[,  c("Sepal.Width","Petal.Width")]
head(iris_new2)
```

```
##   Sepal.Width Petal.Width
## 1         3.5         0.2
## 2         3.0         0.2
## 3         3.2         0.2
## 4         3.1         0.2
## 5         3.6         0.2
## 6         3.9         0.4
```

c. **Select variables using subset**

- Select Sepal.Width and Petal.Width from iris data

```
# select using the subset
iris_new3 <- subset(iris,  select = c("Sepal.Width","Petal.Width"))
head(iris_new3)
```

```
##   Sepal.Width Petal.Width
## 1         3.5         0.2
## 2         3.0         0.2
## 3         3.2         0.2
## 4         3.1         0.2
## 5         3.6         0.2
## 6         3.9         0.4
```

d. **Dropping variables using negative column index**

- Drop Sepal.Width and Petal.Width from iris data

```
# can be dropped using -ve sign of the columns index
iris_new4 <- iris[, - c(2, 4)]
head(iris_new4)
```

```
##   Sepal.Length Petal.Length Species
## 1          5.1          1.4  setosa
## 2          4.9          1.4  setosa
## 3          4.7          1.3  setosa
## 4          4.6          1.5  setosa
## 5          5.0          1.4  setosa
## 6          5.4          1.7  setosa
```

e. **Dropping variables using subset**

- Drop Sepal.Width and Petal.Width from iris data

```
iris_new5 <- subset(iris, select = - c(Sepal.Width, Petal.Width))
head(iris_new5, 2)
```

```
##   Sepal.Length Petal.Length Species
## 1          5.1          1.4  setosa
## 2          4.9          1.4  setosa
```

f. **Add new variables in data frame** <div class="alert alert-block alert-info", style="margin-top: 20px"> Create a categorical variable 'Sepal_Len_cat' based on following rule and add it to the iris data. :

```
Sepal.Length <= 5, catgory="low"
5 < Sepal.Length <= 6, catgory="low_mid"
6 < Sepal.Length <= 7, catgory="high_mid"
7 < Sepal.Length <= 8, catgory="high",
```

- Let's create a variable using loop and conditions

- We use *cut( )* function

```
var1 <- iris$Sepal.Length
cut_off <- c(0, 5, 6, 7 , 8)
catgory <- c("low", "low_mid", "high_mid", "high")
Sepal_Len_cat <- cut(var1, breaks = cut_off, labels = catgory)
iris_new <- cbind(iris, Sepal_Len_cat)
head(iris_new)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Len_cat
## 1          5.1         3.5          1.4         0.2  setosa       low_mid
## 2          4.9         3.0          1.4         0.2  setosa           low
## 3          4.7         3.2          1.3         0.2  setosa           low
## 4          4.6         3.1          1.5         0.2  setosa           low
## 5          5.0         3.6          1.4         0.2  setosa           low
## 6          5.4         3.9          1.7         0.4  setosa       low_mid
```

3. Sorting data in R

a. Sorting based on single column

- use order() function
- order() function returns the indices of the entries in desired order
- Syntax : order(x, decreasing = FALSE)

```
sx <- c(3, 4, 2, 4)
order(sx)
```

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

In the above example shows smallest entry is 2 which is in 3rd position, 2nd smallest entry is 3 which is in first position and so on. We now order the data based on their positions that we found above.

```
sx <- c(3, 4, 2, 4)
order(sx, decreasing = TRUE)
```

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

More Practice of order function

- rearrange the iris_new data in ascending order of Sepal.Length as in above

```
sorted_iris <- iris_new[order(iris_new$Sepal.Length, decreasing = FALSE), ]
head(sorted_iris)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Len_cat
## 14          4.3         3.0          1.1         0.1  setosa           low
## 9           4.4         2.9          1.4         0.2  setosa           low
## 39          4.4         3.0          1.3         0.2  setosa           low
## 43          4.4         3.2          1.3         0.2  setosa           low
## 42          4.5         2.3          1.3         0.3  setosa           low
## 4           4.6         3.1          1.5         0.2  setosa           low
```

- rearrange the iris_new data in descending order of Sepal.Length

```
sorted_iris <- iris_new[order(iris_new$Sepal.Length, decreasing = TRUE), ]
head(sorted_iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species Sepal_Len_cat
## 132          7.9         3.8          6.4         2.0 virginica          high
## 118          7.7         3.8          6.7         2.2 virginica          high
## 119          7.7         2.6          6.9         2.3 virginica          high
## 123          7.7         2.8          6.7         2.0 virginica          high
## 136          7.7         3.0          6.1         2.3 virginica          high
## 106          7.6         3.0          6.6         2.1 virginica          high
```

b. Sorting based on multiple column

- rearrange the iris_new data in ascending order of Sepal.Length, Sepal.Width and Species type

```
sorted_iris <- iris_new[order(iris_new$Sepal.Length,iris_new$Sepal.Width,iris_new$Species,
                              decreasing = FALSE), ]
head(sorted_iris)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Len_cat
## 14          4.3         3.0          1.1         0.1  setosa           low
## 9           4.4         2.9          1.4         0.2  setosa           low
## 39          4.4         3.0          1.3         0.2  setosa           low
## 43          4.4         3.2          1.3         0.2  setosa           low
## 42          4.5         2.3          1.3         0.3  setosa           low
## 4           4.6         3.1          1.5         0.2  setosa           low
```

c. Decoding a variable

Let's decode the variable "Sepal_Len_cat" as "low" = 1, "low_mid" = 2, "high_mid" = 3, "high" = 4.

```
# make sure levels are in order
iris_new$Sepal_Len_cat <- factor(iris_new$Sepal_Len_cat, levels = c("low", "low_mid", "high_mid", "high

# Decoding the levels as 1, 2, 3, 4
iris_new$Sepal_Len_decoded <- as.numeric(iris_new$Sepal_Len_cat)

# View the result

head(iris_new)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Len_cat
## 1          5.1         3.5          1.4         0.2  setosa       low_mid
## 2          4.9         3.0          1.4         0.2  setosa           low
## 3          4.7         3.2          1.3         0.2  setosa           low
## 4          4.6         3.1          1.5         0.2  setosa           low
## 5          5.0         3.6          1.4         0.2  setosa           low
## 6          5.4         3.9          1.7         0.4  setosa       low_mid
##   Sepal_Len_decoded
## 1                 2
## 2                 1
## 3                 1
## 4                 1
## 5                 1
## 6                 2
```

4. Merging of Data Tables

Data merging is essential in data analysis, data science, and database management because it allows combining information from multiple sources to create a comprehensive dataset.

- Data is usually merged based on the primary key,
- Primary key is the set of data columns that are common in dataframes.
- "merge( )" function in R combines two data tables at a tim.

Syntax( )

merge(x,y, . . .), where x, y are data frames to be coerced to one.

Examples (Merging datasets)

- Table-1: Income data of 10 variuos store
- Table-2: Expense data of 10 variuos store

Now we want to find the details of from these two dataset.

```r
library(readxl)
Income_data <- read_excel("~/Desktop/STAT4101L_all_files/4101L-Fall-2023/Course Materials/Section-3 Data
print(Income_data)
```

```
## # A tibble: 10 x 4
##    storeID Sale_in_Thous Rent_in_Thous OtherIncome
##    <chr>           <dbl> <chr>         <chr>
##  1 12AR              165 7             NA
##  2 20AR              132 8             5
##  3 17AR              177 NA            4
##  4 11AR              128 NA            NA
##  5 26AR              137 5             3
##  6 18AR              199 NA            2
##  7 27AR              178 NA            6
##  8 25AR              104 6             NA
##  9 10AR              185 9             7
## 10 13AR              109 NA            1
```

```r
Expense_data <- read_excel("~/Desktop/STAT4101L_all_files/4101L-Fall-2023/Course Materials/Section-3 Da
print(Expense_data)
```

```
## # A tibble: 10 x 4
##    storeID Purchase_in_Thous EmployeeCost OtherCost
##    <chr>               <dbl>        <dbl>     <dbl>
##  1 19AR                  120           10        15
##  2 29AR                  107           10        11
##  3 27AR                   98           11        12
##  4 18AR                   86           15        12
##  5 13AR                   81           14        14
##  6 20AR                  103           15        15
##  7 30AR                  138           11        11
##  8 14AR                  128           14        11
##  9 25AR                  127           14        12
## 10 26AR                  135           15        10
```

```r
merge(Income_data, Expense_data, by = 'storeID')
```

```
##   storeID Sale_in_Thous Rent_in_Thous OtherIncome Purchase_in_Thous
## 1    13AR           109            NA           1                81
## 2    18AR           199            NA           2                86
## 3    20AR           132             8           5               103
## 4    25AR           104             6          NA               127
## 5    26AR           137             5           3               135
## 6    27AR           178            NA           6                98
##   EmployeeCost OtherCost
## 1           14        14
## 2           15        12
## 3           15        15
## 4           14        12
## 5           15        10
## 6           11        12
```

In the above two tables: Both tables have same common id, when the id names are diffrent we use by.x, by.y

```r
class_info <- data.frame(ID = c(001, 002, 005),
                         Name = c("Alex", "Mia", "Sam"))

test_score <- data.frame(stID = c(001, 001, 002, 003),
                         Course = c("Math", "Hist", "Math", "Math"),
                         Score = c(73, 82, 88, 80))

merge(class_info, test_score, by.x = 'ID', by.y = 'stID')
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
```

Types of Merge

1. Inner merge: An inner merge combines two dataframes to include only the rows with matching primary keys in both dataframes. The resulting dataframe will have the following characteristics:

a. Only matching rows are included: Rows are included in the resulting dataframe only if the primary key exists in both dataframes.
b. No unmatched rows: Any row from either dataframe that does not have a match in the other dataframe is excluded from the result.
c. Columns from both dataframes are combined for matching rows: For rows where the primary key matches, the columns from both dataframes are merged into a single row.

This ensures that the resulting dataframe contains only data that is common to both dataframes.

```r
merge(class_info, test_score, by.x = 'ID', by.y = 'stID', all = FALSE)
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
```

2. Left merge: A left merge combines two dataframes to include all rows from the first dataframe ("x") and only the matching rows from the second dataframe ("y"). The resulting dataframe will have the following characteristics:

- a. All rows with common primary keys are included: Rows that have matching primary key values in both dataframes are combined into the resulting dataframe.

- b. All rows from the first dataframe are included, even if there is no match in the second dataframe: If a primary key exists in the first dataframe but not in the second, that row from the first dataframe will still appear in the result.

- c. Missing values (NA) will appear for unmatched columns from the second dataframe: For rows where a primary key is present only in the first dataframe and not in the second, the columns from the second dataframe will be filled with NA values.

```r
merge(class_info, test_score, by.x = 'ID', by.y = 'stID', all.x = TRUE)
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
## 4  5  Sam   <NA>    NA
```

3. Right merge: A right merge combines two dataframes to include all rows from the second dataframe ("y") and only the matching rows from the first dataframe ("x").

```r
merge(class_info, test_score, by.x = 'ID', by.y = 'stID', all.y = TRUE)
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
## 4  3 <NA>   Math    80
```

4. Outer merge: An outer merge combines two dataframes to include all rows from both dataframes ("x" and "y"), regardless of whether there is a match in their primary keys. The resulting dataframe will have the following characteristics:

- a. All rows from both dataframes are included: Every row from both dataframes is part of the resulting dataframe, even if there is no match in the primary keys.
- b. Matching rows are merged: If a primary key exists in both dataframes, the rows are combined, and data from both dataframes is included.
- c. Missing values (NA) appear for unmatched rows:
  - For rows present only in the first dataframe (x) and not in the second (y), the columns from the second dataframe will be filled with NA.
  - Similarly, for rows present only in the second dataframe (y) and not in the first (x), the columns from the first dataframe will be filled with NA.

This ensures that no data is lost from either dataframe, and all unique primary keys are represented in the resulting dataframe.

```r
Merged_data <- merge(class_info, test_score, by.x = 'ID', by.y = 'stID', all = TRUE)
Merged_data
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
## 4  3 <NA>   Math    80
## 5  5  Sam   <NA>    NA
```

5. dplyr Package for Data Manipulation

- Install package: *install.packages('dplyr')*
- Most popular function in *dplyr Package*

select: select varaibles and find subset filter: find subset based on conditional filtering mutate: create a new varaibles or modify existing varaibles arrange: sorting and ordering groupby: aggregating varaibles summarize: aggregating varaibles and finding aggregated values

```r
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
```

5.1. select

```
select(Merged_data, ID, Name, Score)
```

```
##   ID Name Score
## 1  1 Alex    73
## 2  1 Alex    82
## 3  2  Mia    88
## 4  3 <NA>    80
## 5  5  Sam    NA
```

- **Using Pyping operator: %>%**

```
Merged_data %>%
    select(ID, Name, Score) %>%
    head(n = 3) # print top 3
```

```
##   ID Name Score
## 1  1 Alex    73
## 2  1 Alex    82
## 3  2  Mia    88
```

```
# Drop the columns of the dataframe and print top 3
select(Merged_data,-c("Name", "Score")) %>% head(n =3)
```

```
##   ID Course
## 1  1   Math
## 2  1   Hist
## 3  2   Math
```

5.2. filter

```
filter(Merged_data, Course == "Math")
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  2  Mia   Math    88
## 3  3 <NA>   Math    80
```

- Filter the Courses Math and Hist and student is Alex

```r
filter(Merged_data,  Name == "Alex", Course %in% c("Math", "Hist")) %>% head()
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
```

```r
Merged_data %>% filter(Name == "Alex", Course %in% c("Math", "Hist")) %>% head()
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
```

5.3. mutate (Uses: Create a new varaible)

Example: Create a new varaible individual_average

```r
cars_data <- read.csv("~/Desktop/STAT4101L_all_files/4101L-Fall-2023/cars.csv", header = TRUE)
head(cars_data)
```

```
##       Make                   Model  Type Origin DriveTrain  MSRP Invoice
## 1   Subaru              Forester X Wagon              All 21445   19646
## 2   Toyota  Camry Solara SE V6 2dr Sedan   Asia      Front 21965   19819
## 3   Suzuki             Aerio LX 4dr Sedan   Asia      Front 14500   14317
## 4    Dodge          Dakota Club Cab Truck    USA       Rear 20300   18670
## 5    Mazda             Mazda3 s 4dr Sedan   Asia      Front    NA   15922
## 6 Infiniti      G35 Sport Coupe 2dr Sedan   Asia       Rear 29795   27536
##   EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight Wheelbase Length
## 1        2.5         4        165       21          28   3090        99    175
## 2        3.3         6        225       20          29   3417       107    193
## 3        2.3         4        155       25          31   2676        98    171
## 4        3.7         6        210       16          22   3829       131    219
## 5        2.3         4        160       25          31   2762       104    179
## 6        3.5         6        280       18          26   3416       112    182
```

c. mutate

- Create a new varaible avg_mpg = (MPG_City + MPG_Highway)/2

```r
new_cars_data <- cars_data %>%
  mutate(avg_mpg = (MPG_City + MPG_Highway)/2)

new_cars_data  %>%  head(3)
```

```
##     Make                   Model  Type Origin DriveTrain  MSRP Invoice
## 1 Subaru              Forester X Wagon              All 21445   19646
## 2 Toyota  Camry Solara SE V6 2dr Sedan   Asia      Front 21965   19819
## 3 Suzuki             Aerio LX 4dr Sedan   Asia      Front 14500   14317
##   EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight Wheelbase Length
## 1        2.5         4        165       21          28   3090        99    175
## 2        3.3         6        225       20          29   3417       107    193
```

```
## 3        2.3         4        155        25           31   2676         98    171
##   avg_mpg
## 1    24.5
## 2    24.5
## 3    28.0
```

5.4. arrange (Uses: to arrange the dataset in some specific order)

- Sort the dataset in increasing order of Make and decreasing order of MSRP.

```
cars_data_new2 = cars_data %>%
            arrange(Make, desc(MSRP))

head(cars_data_new2)
```

```
##    Make                  Model   Type Origin DriveTrain  MSRP Invoice
## 1 Acura   NSX coupe 2dr manual S Sports   Asia       Rear 89765   79978
## 2 Acura  3.5 RL w/Navigation 4dr  Sedan   Asia      Front 46100   41100
## 3 Acura              3.5 RL 4dr  Sedan   Asia      Front 43755   39014
## 4 Acura                  TL 4dr  Sedan   Asia      Front 33195   30299
## 5 Acura         RSX Type S 2dr  Sedan   Asia      Front 23820   21761
## 6 Acura                 TSX 4dr  Sedan   Asia      Front    NA   24647
##   EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight Wheelbase Length
## 1        3.2         6        290       17          24   3153       100    174
## 2        3.5         6        225       18          24   3893       115    197
## 3        3.5         6        225       18          24   3880       115    197
## 4        3.2         6        270       20          28   3575       108    186
## 5        2.0         4        200       24          31   2778       101    172
## 6        2.4         4        200       22          29   3230       105    183
```

5.5. Data Aggregation

- Aggregation includes process of summarizing group wise the data based on levels of a factor column.
- Aggregation in R can be done using functions *aggregate( )* and *tapply( )*.
- summarise() and group_by()

```
summarise(group_by(cars_data, Type),
          mean_price = mean(MSRP, na.rm = TRUE), count = n() )
```

```
## # A tibble: 6 x 3
##   Type   mean_price count
##   <chr>       <dbl> <int>
## 1 Hybrid     20325      3
## 2 SUV        34447.    60
## 3 Sedan      29716.   262
## 4 Sports     53793.    49
## 5 Truck      22967.    24
## 6 Wagon      29188.    30
```

5.6 Decoding data using *dplyr* package

```
# Remove the 'Sepal_Len_cat1' column and recode 'Sepal_Len_cat'
iris_new <- iris_new %>%
  mutate(Sepal_Len_cat = case_when(
    Sepal_Len_cat == "low" ~ 1,
    Sepal_Len_cat == "low_mid" ~ 2,
    Sepal_Len_cat == "high_mid" ~ 3,
    Sepal_Len_cat == "high" ~ 4
  ))

# Display the first few rows of the updated dataset
head(iris_new)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Len_cat
## 1          5.1         3.5          1.4         0.2  setosa             2
## 2          4.9         3.0          1.4         0.2  setosa             1
## 3          4.7         3.2          1.3         0.2  setosa             1
## 4          4.6         3.1          1.5         0.2  setosa             1
## 5          5.0         3.6          1.4         0.2  setosa             1
## 6          5.4         3.9          1.7         0.4  setosa             2
##   Sepal_Len_decoded
## 1                 2
## 2                 1
## 3                 1
## 4                 1
## 5                 1
## 6                 2
```

5.7 Merging data using dplyr

```
head(class_info)
```

```
##   ID Name
## 1  1 Alex
## 2  2  Mia
## 3  5  Sam
```

```
test_score
```

```
##   stID Course Score
## 1    1   Math    73
## 2    1   Hist    82
## 3    2   Math    88
## 4    3   Math    80
```

```
Left_join_data <- left_join(class_info, test_score, by = c("ID"="stID"))
Left_join_data
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
## 4  5  Sam   <NA>    NA
```

```r
Inner_join_data <- inner_join(class_info, test_score, by = c("ID"="stID"))
Inner_join_data
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
```

```r
full_join_data <- full_join(class_info, test_score, by = c("ID"="stID"))
full_join_data
```

```
##   ID Name Course Score
## 1  1 Alex   Math    73
## 2  1 Alex   Hist    82
## 3  2  Mia   Math    88
## 4  5  Sam   <NA>    NA
## 5  3 <NA>   Math    80
```

6. Data Reshaping using tydyr

The tidyr package in R is powerful for data wrangling and analysis. It is commonly used in conjunction with dplyr for tidying, transforming, and analyzing data before applying statistical models or visualizations. Below are key data analysis workflows using tidyr.

6.1. Load Required Library

```r
library(tidyr)
```

Example Data: Student Scores Dataset

```r
# Sample data
df <- data.frame(
  ID = c(1, 1, 2, 2, 3, 3, 4),
  Name = c("Alex", "Alex", "Mia", "Mia", "Sam", "Sam", NA),
  Course = c("Math", "Hist", "Math", "Hist", "Math", "Hist", "Math"),
  Score = c(90, 85, 88, 92, 78, NA, 95)
)

print(df)
```

```
##   ID Name Course Score
## 1  1 Alex   Math    90
## 2  1 Alex   Hist    85
```

```
## 3  2  Mia   Math   88
## 4  2  Mia   Hist   92
## 5  3  Sam   Math   78
## 6  3  Sam   Hist   NA
## 7  4 <NA>   Math   95
```

6.2. Data Cleaning and Tidying with tidyr

Handling Missing Values: replace_na() & drop_na() We have missing values in the Name and Score columns. We can either remove them or replace them.

```
df_clean <- df %>%
  replace_na(list(Name = "Unknown", Score = 0))  # Replace missing names with "Unknown" and scores with

print(df_clean)
```

```
##   ID    Name Course Score
## 1  1    Alex   Math    90
## 2  1    Alex   Hist    85
## 3  2     Mia   Math    88
## 4  2     Mia   Hist    92
## 5  3     Sam   Math    78
## 6  3     Sam   Hist     0
## 7  4 Unknown   Math    95
```

```
df_no_na <- df %>%
  drop_na() # remove rows with missing value

print(df_no_na)
```

```
##   ID Name Course Score
## 1  1 Alex   Math    90
## 2  1 Alex   Hist    85
## 3  2  Mia   Math    88
## 4  2  Mia   Hist    92
## 5  3  Sam   Math    78
```

6.3. Reshaping Data: pivot_wider() & pivot_longer()

After tidying, we may need to reshape the data.

```
df_wide <- df_clean %>%
  pivot_wider(names_from = Course, values_from = Score) # Convert Long to Wide Format

print(df_wide)
```

```
## # A tibble: 4 x 4
##      ID Name     Math  Hist
##   <dbl> <chr>   <dbl> <dbl>
## 1     1 Alex       90    85
## 2     2 Mia        88    92
## 3     3 Sam        78     0
## 4     4 Unknown    95    NA
```

```r
df_long <- df_wide %>%
  pivot_longer(cols = c(Math, Hist), names_to = "Course", values_to = "Score") #Convert Wide to Long Fo

print(df_long)
```

```
## # A tibble: 8 x 4
##      ID Name    Course Score
##   <dbl> <chr>   <chr>  <dbl>
## 1     1 Alex    Math      90
## 2     1 Alex    Hist      85
## 3     2 Mia     Math      88
## 4     2 Mia     Hist      92
## 5     3 Sam     Math      78
## 6     3 Sam     Hist       0
## 7     4 Unknown Math      95
## 8     4 Unknown Hist      NA
```