Data manipulation in R

Surya Lamichhane

17 January 2024

Contents

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.

Reading data in R

1. 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/. . . .")*
- Make sure the working directory changed: *getwd( )*

2. Reading data:

a. Reading csv file: /Users/suryalamichhane/Desktop/STAT4101L_all_files/

```
cars_data = read.csv("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/Assi
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
```

b. Reading Excel file: Base R cannot read an excel sheet; however, it can be read using read_excel function from the "readxl" package.

```
# install "readxl" package first if required
library(readxl)
Claim_insurance = read_excel("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataS
Beneficiary_insurance = read_excel("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles
names(Claim_insurance)
```

```
##  [1] "BenefUserID"    "ClmAmount"      "Clmdoa"         "Clmdod"
##  [5] "ClmApprovedamt" "StatusName"     "AilmentName"    "AilmentCode"
##  [9] "TreatmentName"  "CityName"       "HospId"
```

```r
names(Beneficiary_insurance)
```

```
## [1] "BenefUserID"     "BenefPriID"      "BenefRelToPriID" "BenefPolID"
## [5] "BenefDOB"        "BenefSex"        "BenefPremium"    "BenefActive"
```

b. Text data: We use function *read.table( )* to read text data. Check how the data are separated and check if the data has header or not.

- Tablular data

```r
Class_marks = read.table("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/
head(Class_marks, 2)
```

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

- Text string

```r
Data_wrangling_text = paste(readLines("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfi
Data_wrangling_text
```

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

- 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
nchar(Data_wrangling_text)
```

```
## [1] 462
```

3. Data exploration in R:

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

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

```r
data(iris)
head(iris, 2) # 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
```

```r
cat("first 4 entris of Sepal.Length", '\n')
```

```
## first 4 entris of Sepal.Length
```

```r
iris$Sepal.Length[1:4] # first 4 entris of Sepal.Length
```

```
## [1] 5.1 4.9 4.7 4.6
```

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

Conditional filtering

a. Based on single condition

- Filter the dataset that corresponds to 'setosa' Species

```r
iris$Species == 'setosa'
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
#returns the true if iris$Species is setosa otherwise false,
#which can be used to find a subset where Species == 'setosa'
```

```r
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
```

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
```

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
```

Add new variables in data frame

f. Create a categorical variable 'Sepal_Len_cat' based on following rule:

```
 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",
```

and add it to the iris data.

- Let's create a variable using loop and conditions

```
var1 = iris$Sepal.Length
Sepal_Len_cat = c() # empty vector
for ( value in var1){
  if (value <= 5){
    catgory = "low"
  }else if(value <= 6){
    catgory = "low_mid"
  }else if(value <= 7){
    catgory = "high_mid"
  }else{
```

```
    catgory = "high"
  }
  Sepal_Len_cat = c(Sepal_Len_cat, catgory)
}
table(Sepal_Len_cat)
```

```
## Sepal_Len_cat
##     high high_mid      low  low_mid
##       12       49       32       57
```

- We can add the created variable using dollor sign or cbind() function

```
# iris = cbind(iris, Sepal_Len_cat) #new data defined
iris$Sepal_Len_cat = Sepal_Len_cat
#iris$index = 1: length( Sepal_Len_cat)
head(iris)
```

```
##   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
```

- we can create the same variable using *cut( )* function

```
var1 = iris$Sepal.Length
cut_off = c(0, 5, 6, 7 , 8)
catgory = c("low", "low_mid", "high_mid", "high")
Sepal_Len_cat1 = cut(var1, breaks = cut_off, labels = catgory)
iris_new = cbind(iris, Sepal_Len_cat1)
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_cat1
## 1        low_mid
## 2            low
## 3            low
## 4            low
## 5            low
## 6        low_mid
```

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

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

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

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

```
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
##    Sepal_Len_cat1
## 14            low
## 9             low
## 39            low
## 43            low
## 42            low
## 4             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
##     Sepal_Len_cat1
## 132           high
## 118           high
## 119           high
## 123           high
## 136           high
## 106           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
##    Sepal_Len_cat1
## 14            low
## 9             low
## 39            low
## 43            low
## 42            low
## 4             low
```

c. Decoding a varaible

Let's drop created variable 'Sepal_Len_cat1' from the *iris_new* data, and decode the variable "Sepal_Len_cat" as "low" = 0, "low_mid" = 1, "high_mid" = 2, "high" = 3.

```
iris_new = subset(iris_new, select = - Sepal_Len_cat1)
iris_new$Sepal_Len_cat[iris_new$Sepal_Len_cat == "low"] = 0
iris_new$Sepal_Len_cat[iris_new$Sepal_Len_cat == "low_mid"] = 1
iris_new$Sepal_Len_cat[iris_new$Sepal_Len_cat == "high_mid"] = 2
iris_new$Sepal_Len_cat[iris_new$Sepal_Len_cat == "high"] = 3
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             1
## 2          4.9         3.0          1.4         0.2  setosa             0
## 3          4.7         3.2          1.3         0.2  setosa             0
## 4          4.6         3.1          1.5         0.2  setosa             0
## 5          5.0         3.6          1.4         0.2  setosa             0
## 6          5.4         3.9          1.7         0.4  setosa             1
```

Summarizing Data

a. Descriptive statistics

summary() function can be used to compute the statistical summary of data.

```
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   Sepal_Len_cat
##  setosa    :50   Length:150
##  versicolor:50   Class :character
##  virginica :50   Mode  :character
##
##
##
```

Important Statistical Functions for Data Analysis

```
mean(x): Returns mean for a column of a dataframe
median(x):  Returns median for a column of a dataframe
sd(x): Returns standard deviation for a column of a dataframe
var(x): Returns variance for a column of a dataframe
table(x): Returns count for each unique value in a data column
quantile(x): Returns the 0th, 25th, 50th, 75th, and 100th percentiles of the data column
IQR(x): Returns inter quartile range for a data column
range(x): Returns minimum and maximum values for the data column
```

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( )*.

a. Use of *aggregate( )*

- Let's find aggregated mean of 'Sepal.Length'

** Syntax**

```
aggregate(x, by, FUN, ...)
```

```
#mean of Sepal.Length by Species
mean_by_Species = aggregate(x = iris_new$Sepal.Length,
                            by = list(iris_new$Species),  FUN = mean )
mean_by_Species
```

```
##      Group.1     x
## 1     setosa 5.006
## 2 versicolor 5.936
## 3  virginica 6.588
```

- Using formula

```
## find count based on  Sepal_Len_cat
mean_by_Species1 <- aggregate(Sepal.Length ~ Sepal_Len_cat, data = iris_new, length)
mean_by_Species1
```

```
##   Sepal_Len_cat Sepal.Length
## 1             0           32
## 2             1           57
## 3             2           49
## 4             3           12
```

b. Use of *tapply( )*

```
#mean of Sepal.Length by Species
mean_by_Species = tapply(iris_new$Sepal.Length, iris_new$Species, FUN = mean )
mean_by_Species
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

Merging of Data Tables

- When data for analysis might not be available as a single source, we may need to combine data from two or more sources for our analysis.
- 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 time.

Syntax( )

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

```
load('/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/Assisted Practice da
class_info
```

Examples (Merging datasets)

```
##      ID   Name Sex Age Interest
## 1  A103 Jennie   F  11    dance
## 2  A105 Garret   M  10   sports
```

```
## 3  A106 Warren   M  12     arts
## 4  A107  Janet   F  10   sports
## 5  A108   John   M  11     arts
## 6  A110   Sean   M  10   sports
## 7  A112   Rita   F  11    music
## 8  A114    Leo   M  12     arts
## 9  A115  Keith   M  11   sports
## 10 A116 Nicole   F  11   sports
## 11 A117   Dave   M  11   sports
## 12 A119  Annie   F  12     arts
## 13 A120  Frank   M  11     arts
```

```
Class_marks = read.csv('/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/As
Class_marks
```

```
##    Enrol.No. Maths Science English
## 1       A101    16      15      12
## 2       A102    16      17      11
## 3       A103    12      18      17
## 4       A104    11      13      16
## 5       A107    14      14      11
## 6       A108    19      15      12
## 7       A109    19      11      13
## 8       A110    16      13      18
## 9       A111    15      13      14
## 10      A112    16      17      19
## 11      A113    13      19      10
## 12      A114    12      19      17
## 13      A115    18      19      17
## 14      A118    11      17      10
## 15      A119    15      19      12
```

```
merge(class_info, Class_marks, by.x = 'ID', by.y = 'Enrol.No.')
```

```
##     ID   Name Sex Age Interest Maths Science English
## 1 A103 Jennie   F  11    dance    12      18      17
## 2 A107  Janet   F  10   sports    14      14      11
## 3 A108   John   M  11     arts    19      15      12
## 4 A110   Sean   M  10   sports    16      13      18
## 5 A112   Rita   F  11    music    16      17      19
## 6 A114    Leo   M  12     arts    12      19      17
## 7 A115  Keith   M  11   sports    18      19      17
## 8 A119  Annie   F  12     arts    15      19      12
```

Types of Merge

1. Inner merge: It combines dataframes to keep only the rows that match the primary key.

```
merge(class_info, Class_marks, by.x = 'ID', by.y = 'Enrol.No.', all = FALSE)
```

```
##     ID   Name Sex Age Interest Maths Science English
```

```
## 1 A103 Jennie   F  11    dance    12      18      17
## 2 A107  Janet   F  10   sports    14      14      11
## 3 A108   John   M  11     arts    19      15      12
## 4 A110   Sean   M  10   sports    16      13      18
## 5 A112   Rita   F  11    music    16      17      19
## 6 A114    Leo   M  12     arts    12      19      17
## 7 A115  Keith   M  11   sports    18      19      17
## 8 A119  Annie   F  12     arts    15      19      12
```

2. Left merge: It combines dataframes to include all the rows of the dataframe "x" and only the matching rows from "y"

```
merge(class_info, Class_marks, by.x = 'ID', by.y = 'Enrol.No.', all.x = TRUE)
```

```
##      ID   Name Sex Age Interest Maths Science English
## 1  A103 Jennie   F  11    dance    12      18      17
## 2  A105 Garret   M  10   sports    NA      NA      NA
## 3  A106 Warren   M  12     arts    NA      NA      NA
## 4  A107  Janet   F  10   sports    14      14      11
## 5  A108   John   M  11     arts    19      15      12
## 6  A110   Sean   M  10   sports    16      13      18
## 7  A112   Rita   F  11    music    16      17      19
## 8  A114    Leo   M  12     arts    12      19      17
## 9  A115  Keith   M  11   sports    18      19      17
## 10 A116 Nicole   F  11   sports    NA      NA      NA
## 11 A117   Dave   M  11   sports    NA      NA      NA
## 12 A119  Annie   F  12     arts    15      19      12
## 13 A120  Frank   M  11     arts    NA      NA      NA
```

3. Right merge: It combines dataframes to include all the rows of the dataframe "y" and only the matching rows from "x"

```
merge(class_info, Class_marks, by.x = 'ID', by.y = 'Enrol.No.', all.y = TRUE)
```

```
##      ID   Name  Sex Age Interest Maths Science English
## 1  A101   <NA> <NA>  NA     <NA>    16      15      12
## 2  A102   <NA> <NA>  NA     <NA>    16      17      11
## 3  A103 Jennie    F  11    dance    12      18      17
## 4  A104   <NA> <NA>  NA     <NA>    11      13      16
## 5  A107  Janet    F  10   sports    14      14      11
## 6  A108   John    M  11     arts    19      15      12
## 7  A109   <NA> <NA>  NA     <NA>    19      11      13
## 8  A110   Sean    M  10   sports    16      13      18
## 9  A111   <NA> <NA>  NA     <NA>    15      13      14
## 10 A112   Rita    F  11    music    16      17      19
## 11 A113   <NA> <NA>  NA     <NA>    13      19      10
## 12 A114    Leo    M  12     arts    12      19      17
## 13 A115  Keith    M  11   sports    18      19      17
## 14 A118   <NA> <NA>  NA     <NA>    11      17      10
## 15 A119  Annie    F  12     arts    15      19      12
```

4. Outer merge: It combines dataframes to keep all the rows from both the dataframes.

```
merge(class_info, Class_marks, by.x = 'ID', by.y = 'Enrol.No.', all = TRUE)
```

```
##      ID   Name  Sex Age Interest Maths Science English
## 1  A101   <NA> <NA>  NA     <NA>    16      15      12
## 2  A102   <NA> <NA>  NA     <NA>    16      17      11
## 3  A103 Jennie    F  11    dance    12      18      17
## 4  A104   <NA> <NA>  NA     <NA>    11      13      16
## 5  A105 Garret    M  10   sports    NA      NA      NA
## 6  A106 Warren    M  12     arts    NA      NA      NA
## 7  A107  Janet    F  10   sports    14      14      11
## 8  A108   John    M  11     arts    19      15      12
## 9  A109   <NA> <NA>  NA     <NA>    19      11      13
## 10 A110   Sean    M  10   sports    16      13      18
## 11 A111   <NA> <NA>  NA     <NA>    15      13      14
## 12 A112   Rita    F  11    music    16      17      19
## 13 A113   <NA> <NA>  NA     <NA>    13      19      10
## 14 A114    Leo    M  12     arts    12      19      17
## 15 A115  Keith    M  11   sports    18      19      17
## 16 A116 Nicole    F  11   sports    NA      NA      NA
## 17 A117   Dave    M  11   sports    NA      NA      NA
## 18 A118   <NA> <NA>  NA     <NA>    11      17      10
## 19 A119  Annie    F  12     arts    15      19      12
## 20 A120  Frank    M  11     arts    NA      NA      NA
```

dplyr Package for Data Manipulation

- Install package: *install.packages('dplyr')*
- Must 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
```

```
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
```

1. select

```
select(class_info, ID, Name, Interest)
```

```
cars_data = read.csv('/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/Assi
dplyr::select(cars_data, starts_with('MPG'), ends_with("Length"))
```

```
# Drop the columns of the dataframe
select(cars_data,-c("Horsepower","MPG_City","MPG_Highway"))
```

2. filter

```
filter(cars_data, MSRP > 100000, Origin == "Europe")
```

- Filter the Type = 'Hybrid' and manufactured by Honda and Toyota.

```
filter(cars_data,  !Make %in% c("Honda", "Toyota") )
```

3. mutate

- Create a new varaible

avg_mpg = (MPG_City + MPG_Highway)/2

```
cars_data_new1 = mutate(cars_data, avg_mpg = (MPG_City + MPG_Highway)/2)
head(cars_data_new1)
```

4. arrange

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

```
cars_data_new2 = arrange(cars_data, Make, desc(MSRP))
head(cars_data_new2)
```

5. summarise() and group_by()

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

Using Pipeline Operator : %>% in dplyr

- Pipeline operator can be used to link functions from any package.
- The above tasks can be done as follows:

a. select

```
class_info %>%
  select(ID, Name, Interest) %>%
  filter(Interest == 'dance')
```

```
cars_data %>% select(starts_with('MPG'), ends_with('Length'))
```

```
# Drop the columns of the dataframe
cars_data %>%
  select(-c("Horsepower","MPG_City","MPG_Highway"))
```

b. filter

```
cars_data %>%
filter(MSRP > 100000, Origin == "Europe")
```

- Filter the Type = 'Hybrid' and manufactured by Honda and Toyota.

```
cars_data %>%
  filter(Make %in% c("Honda", "Toyota"), Type == 'Hybrid' )
```

c. mutate

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

```
cars_data %>%
  mutate(avg_mpg = (MPG_City + MPG_Highway)/2)
```

c. arrange

- 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)
```

d. summarise() and group_by()

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

```
## # A tibble: 6 x 3
##   Type   mean_price     n
##   <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
```