

Introduction to R

Bobbur Abhilash Chowdary

IIM Calcutta

August 2018

Why R?

- It's open source!
- Need not run behind IT support to get the license renewed (SAS users?)
- It's one of the fastest growing languages along with Python for Machine Learning and AI
- Easy integration with GitHub and Latex
- Reproducible research(?)
- Imagine with one click your code analyzes your data and generates all the required tables & graphs, and finally generates a nice pdf/word document which you can send it to a journal. This can be done in R.

Installing R

- Download and install R from <https://www.r-project.org/>
- User interface of just R is very bad. So we need to install R Studio. There are alternative to R-Studio like Emacs etc. You can also try them.
- Download and install R-Studio from <https://www.rstudio.com/>. Free version is good enough.
- R-studio runs on top of R. R-studio relies on R to execute all the commands. R-studio cannot function without R.
- Don't change the order of installation. First R and then R-Studio. Otherwise sometime it can create problems.
- Try this in console and check output to see if RStudio working

```
1+2
```

```
[1] 3
```

Optional - Git -1

Have you ever been frustated seeing files named Final_Version, New_Final_Version? Have you ever wondered what is the difference between Version_1 and Version_2?

Then what you need is a version control system. Git is a popular version control system. Luckily R-Studio has very good interface with Git and GitHub.

Once again Git is open source and free. One can download Git from <https://git-scm.com/>.

Git Clients : Git ~ RStudio : R. There several Git clients out there. You may use any of them (Sourcetree, GitUp, GitHub etc).

Rstudio does most of the basic stuff a Git clients does. Did I forget to say RStudio has good interface with Git?

Optional - Git -2

Also create GitHub account so you can store your code online. Private repository facility is free if you register with IIM Cal id. GO through the steps after installing Git.

Step 1 - Open Git Bash app and type the following commands

```
git config --global user.name 'abcd efgh'
git config --global user.email 'abcdexy@email.iimcal.ac.in'
git config --global --list
```

Step 2 - Go to GitHub.com and create a new repository. Choose default options. Open your repository to find readme.md file. Press “clone or download” button on top right corner and copy the url.

Step 3 - Open Rstudio >File >New Project >Version Control >Git . Paste the above url and choose the folder you want to install the project in. A new folder with repository name will be created.

Optional - Git -3

Step 4 - Open the readme.md file. Make some changes and save it. In your Git pane in Rstudio (beside Environment pane) select Readme.md and press commit. Type a commit message in new window and then press commit. Then press push (arrow up) to send the commit to GitHub. Open GitHub.com to find the changes reflected in Readme.md file there.

Use ‘diff’ in Git pane to see changes you have made since last commit. To revert to previous commits use the addin below the menu section in RStudio. Git is most useful when you are collaborating with your colleagues. You can copy the code from GitHub to your desktop

RStudio Basics

‘Source’ pane shows the scripts. For a new script : RStudio >File >New File >R Script

Script has .R extension. It is the place where you will write all your code. Select the code and press ‘Run’ option on top right corner to execute it. Shortcut: CTRL + ENTER

The ‘Console’ pane is where you will find the result of executing the code you have written in script. Always read the output in console especially if it is in **RED**

‘Environment’ pane is where you will find the datasets you have loaded into the memory.

‘Files’ pane contains the list of files in the current working directory

Intro to R

- Display current working directory and set current working directory

```
getwd()
setwd("G:/My Drive/Thesis/BGs")
```

- Installing Packages and Loading Packages. Need to install a library only once per computer but have to load it everytime you restart.

```
install.packages("data.table")
install.packages("tidyverse")
install.packages("nycflights13")
library(nycflights13)
```

```
library(data.table)
library(tidyverse)
```

Variable names

- R is case sensitive. 'Data' is not same as 'data'.
- Names can have ' . ' (dot), ' _ ', letters and numbers.
- A good coding convention is to always start with a letter.
- Press 'ALT + SHIFT + K' for list of Keyboard shortcuts
- whenever you find '+' in the console that implies R is waiting for you to complete your command. Try typing 1+ in console.
- Always use '<-' for assigning. '=' also works but sometimes it won't.
- Missing values in R are called 'NA'. The only way to do conditional tests is "is.na".

```
x =3; y=4; x==y
x =NA; y=NA; x==y
x <- c(1,2,3,4,5,6,NA); x>3; x<4
```

mtcars data set

mtcars is a default data set in R. We will use this extensively.

mtcars dataset details

```
[, 1] mpg Miles/(US) gallon
[, 2] cyl Number of cylinders
[, 3] disp Displacement (cu.in.)
[, 4] hp Gross horsepower
[, 5] drat Rear axle ratio
[, 6] wt Weight (1000 lbs)
[, 7] qsec 1/4 mile time
[, 8] vs Engine (0 = V-shaped, 1 = straight)
[, 9] am Transmission (0 = automatic, 1 = manual)
[,10] gear Number of forward gears
[,11] carb Number of carburetors
```

Checking out the data set

head() and tail() functions provide the first and last 'n' lines of a given dataset respectively . summary() function summarises and str() gives an idea of structure of data set.

```
var1 <- head(mtcars,10)
var2 <- tail(ggplot2::mpg,10)
View(var1) # capital V in View
var1 <- summary(mtcars)
View(var1)
str(mtcars)
```

Common types of variables.

int - integer; dbl - real number

chr - character vector; date - date
fctr - factor; lgl - logical

Column names

Finding Column names, renaming all column names and changing just one column name

```
colnames(var1)
colnames(var1) <- c("mpg", "cyl", "disp", "hp", "drat",
                   "wt", "qsec", "vs", "am", "gear1", "carb")
setnames(var1, "gear1", "gear")
```

Cleaning Environment and Console

- Removing a specific data set

```
rm(var1, var2)
```

- Removing all data sets

```
rm(list = ls())
```

- Clear Console

```
cat("\014")
```

Reading and writing files

```
var0 <- mtcars
write.csv(var0, file = "Output/data.csv")
write.table(var0, file = "Output/data1.txt", sep = "|")
data <- read.csv(file = "Output/data.csv", row.names = 1)
data1 <- read.table("Output/data1.txt",
                   header = TRUE, sep = "|")
data2 <- read.csv(file = "Output/data.csv")
download.file("https://www.nseindia.com/content/historical/EQUITIES/2018/JUL/cm30JUL2018bhav.csv.zip",
              "NSE BhavCopy/cm30JUL2018bhav.csv.zip")
```

For better import functions refer 'readr' package. Use 'readxl' package for reading Excel data sheets. Use 'haven' & 'foreign' packages for reading SAS, STATA and SPSS files.

aggregate functions in base R

```
x = 1:100
```

```
min(x)
max(x)
mean(x)
sum(x)
sd(x)
median(x)
IQR(x)
```

```
[1] 1
[1] 100
[1] 50.5
[1] 5050
[1] 29.01149
[1] 50.5
[1] 49.5
```

Basic data operations

Most operations in data cleaning process fall into these categories

filter - selecting observations/rows

arrange - ordering the observations

select - selecting columns

mutate - creating new variables from existing ones

summarize - summary

The above five are functions in 'dplyr' package. 'dplyr' and a couple of other packages together are called 'tidyverse' which we already loaded.

dplyr - filter

Use '==' when its a test/question. Use '=' for assigning/informing. We use '==' below because we want to ask "if gear = 4 then select" which is a test.

```
mtcars1 <- filter(mtcars, gear == 4, cyl == 6)
mtcars2 <- filter(mtcars, gear == 4 & cyl == 6) # =mtcars1
mtcars3 <- filter(mtcars, gear == 4 | cyl == 6) #OR operator
mtcars4 <- filter(mtcars, gear == 4 | gear == 6)
mtcars5 <- filter(mtcars, gear %in% c(4,6)) # =mtcars4
```

All 'dplyr' operations dont affect the original dataset. mtcars still has 32 observations. Try 'mtcars' on the left side of the above equation. Common operators recognized in R - ">, <, >=, <=, != (not equal) and =="

dplyr - Arrange and select

Similar to filter(), first input to arrange() is dataset name followed by column names for sorting. use desc() for descing order.

```
arrange(mtcars, mpg, desc(cyl))
```

Selecting columns uses select() function which is similar to arrange() and filter() functions

```
select(mtcars, mpg, cyl)
select(mtcars, -mpg)
select(mtcars, vs, everything()) #rearranging
```

dplyr - creating variables

Similarly use 'mutate' for creating new variables

```
mutate(mtcars, mpg_per_hp = mpg/hp, mpg_per_wt = mpg/wt,
       mpg_per_wt1 = mpg_per_hp*hp/wt)
transmute(mtcars, mpg_per_hp = mpg/hp, mpg_per_wt = mpg/wt,
          mpg_per_wt1 = mpg_per_hp*hp/wt)
```

You can use the following operators in mutate - '+, -, /, *, ^',
 %% (reminder e.g. - 5 %% 2 = 1), %/% (e.g. - 5 %/% 2 = 2), sum(), cumsum(), prod()

dplyr - summarize()

summarize can use all aggregate function in base R and the following

n(): the number of observations in the current group n_distinct(x):the number of unique values in x. first(x), last(x) and nth(x, n) - these work similarly to x[1], x[length(x)], and x[n] but give you more control over the result if the value is missing.

```
by_gear <- group_by(mtcars, gear, carb)
summarise(by_gear, mean_milage = mean(mpg))
```

Pipes

Pipes are very useful to write good looking code. The most used Pipe is %>%. Pipes also reduce the need to save intermediaries.

x %>% f(y) is equivalent to f(x,y). Above example can be rewritten as...

```
mtcars %>% group_by(gear, carb) %>%
  summarise(mean_milage = mean(mpg))
```

```
ungroup(mtcars)
```

dplyr - joins() introduction

By default it uses common variables. here by= "carrier" is default but we will specify it. airlines data has "name" of carrier

```
flights2 <- flights %>% select(year, hour, origin, dest, tailnum, carrier)
flights2 %>% left_join(planes[,c("tailnum", "manufacturer", "model")], by = "tailnum") %>% head(5) %>% knitr::kable()
```

year	hour	origin	dest	tailnum	carrier	manufacturer	model
2013	5	EWR	IAH	N14228	UA	BOEING	737-824
2013	5	LGA	IAH	N24211	UA	BOEING	737-824
2013	5	JFK	MIA	N619AA	AA	BOEING	757-223
2013	5	JFK	BQN	N804JB	B6	AIRBUS	A320-232
2013	6	LGA	ATL	N668DN	DL	BOEING	757-232

If the by variables are not named same in both the data sets

```
flights2 %>% left_join(airports[,c("faa", "name", "lat")], c("dest" = "faa")) %>% head(5) %>% knitr::kable()
```

year	hour	origin	dest	tailnum	carrier	name	lat
2013	5	EWR	IAH	N14228	UA	George Bush Intercontinental	29.98443

year	hour	origin	dest	tailnum	carrier	name	lat
2013	5	LGA	IAH	N24211	UA	George Bush Intercontinental	29.98443
2013	5	JFK	MIA	N619AA	AA	Miami Intl	25.79325
2013	5	JFK	BQN	N804JB	B6	NA	NA
2013	6	LGA	ATL	N668DN	DL	Hartsfield Jackson Atlanta Intl	33.63672

dplyr - left_join()

```
df1 <- data_frame(x = c(1, 2), y = 2:1)
df2 <- data_frame(x = c(1, 3), a = 10, b = "a")
```

left join includes observations of 'x' dataset irrespective of there is amatching column in 'Y' dataset

```
df1 %>% left_join(df2) %>% knitr::kable()
```

Joining, by = "x"

x	y	a	b
1	2	10	a
2	1	NA	NA

dplyr - right_join()

right join - is same as left_join(y,x). but here columns of y are still after x in right_join(x,y)

```
df1 %>% right_join(df2) %>% knitr::kable()
```

Joining, by = "x"

x	y	a	b
1	2	10	a
3	NA	10	a

dplyr - inner_join()

inner join includes observation common to both datasets

```
df1 %>% inner_join(df2) %>% knitr::kable()
```

Joining, by = "x"

x	y	a	b
1	2	10	a

dplyr - full_join()

full_join inlcudes all rows of x and y

```
df1 %>% full_join(df2) %>% knitr::kable()
```

Joining, by = "x"

x	y	a	b
1	2	10	a
2	1	NA	NA
3	NA	10	a

dplyr - join problems

beware of cartesian products arising out of your dataset

```
df1 <- data_frame(x = c(1, 1, 2), y = 1:3)
df2 <- data_frame(x = c(1, 1, 2), z = c("a", "b", "a"))
df1 %>% left_join(df2) %>% knitr::kable()
```

Joining, by = "x"

x	y	z
1	1	a
1	1	b
1	2	a
1	2	b
2	3	a

dplyr - semi_join and anti_join

If you're worried about what observations your joins(left, right, inner, and full joins) will match, start with a semi_join() or anti_join(). semi_join() and anti_join() never duplicate they only ever remove observations.

semi_join(x, y) keeps all observations in x that have a match in y. anti_join(x, y) drops all observations in x that have a match in y.

```
df1 <- data_frame(x = c(1, 1, 3, 4), y = 1:4)
df2 <- data_frame(x = c(1, 1, 2), z = c("a", "b", "a"))
df1 %>% nrow()
df1 %>% inner_join(df2, by = "x") %>% nrow()
df1 %>% semi_join(df2, by = "x") %>% knitr::kable()
```

[1] 4

[1] 4

x	y
1	1
1	2

dplyr - set operations

```
df1 <- data_frame(x = 1:2, y = c(1L, 1L))
df2 <- data_frame(x = 1:2, y = 1:2)
```

like in sets we have four operations - Union, intersection, A-B, B-A

```
intersect(df1, df2) %>% knitr::kable()
union(df1, df2) %>% knitr::kable()
setdiff(df1, df2) %>% knitr::kable()
setdiff(df2, df1) %>% knitr::kable()
```

x	y
1	1

x	y
1	1
2	1
2	2

x	y
2	1

x	y
2	2

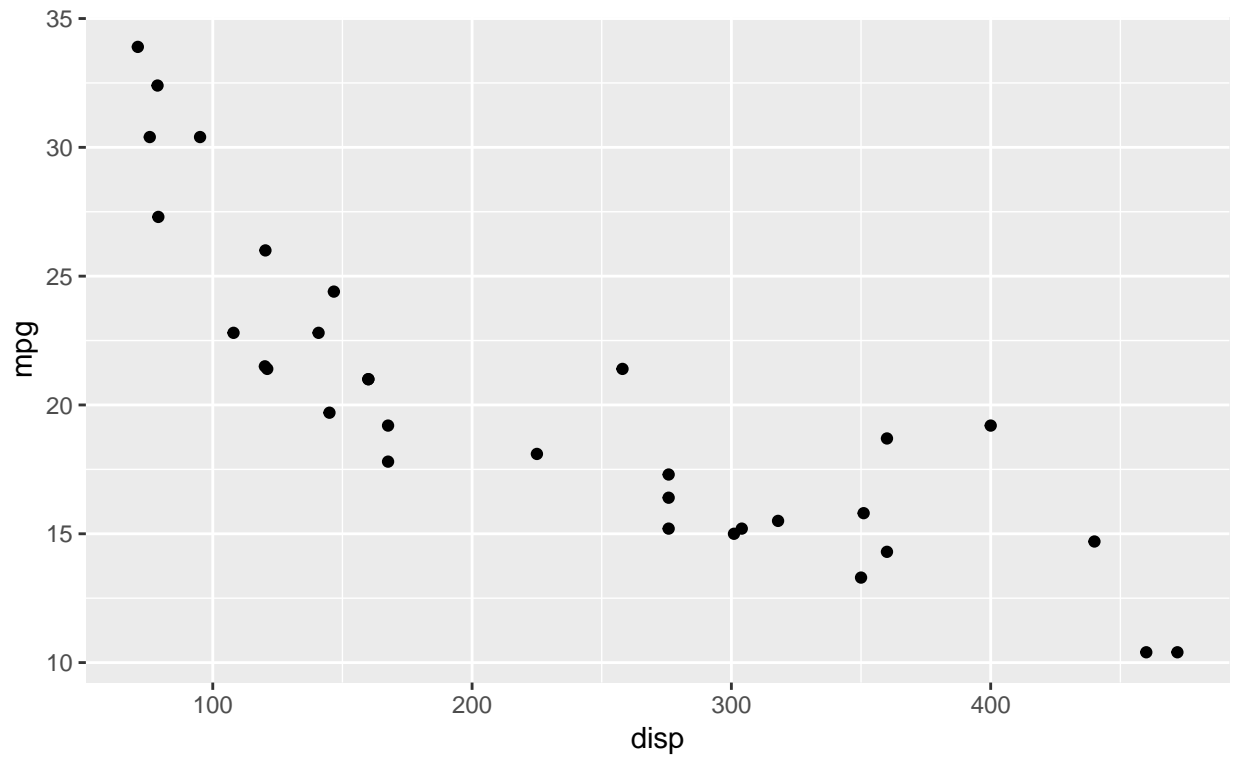
dplyr- rename

```
rename(df2, newnameofy = y) %>% knitr::kable()
```

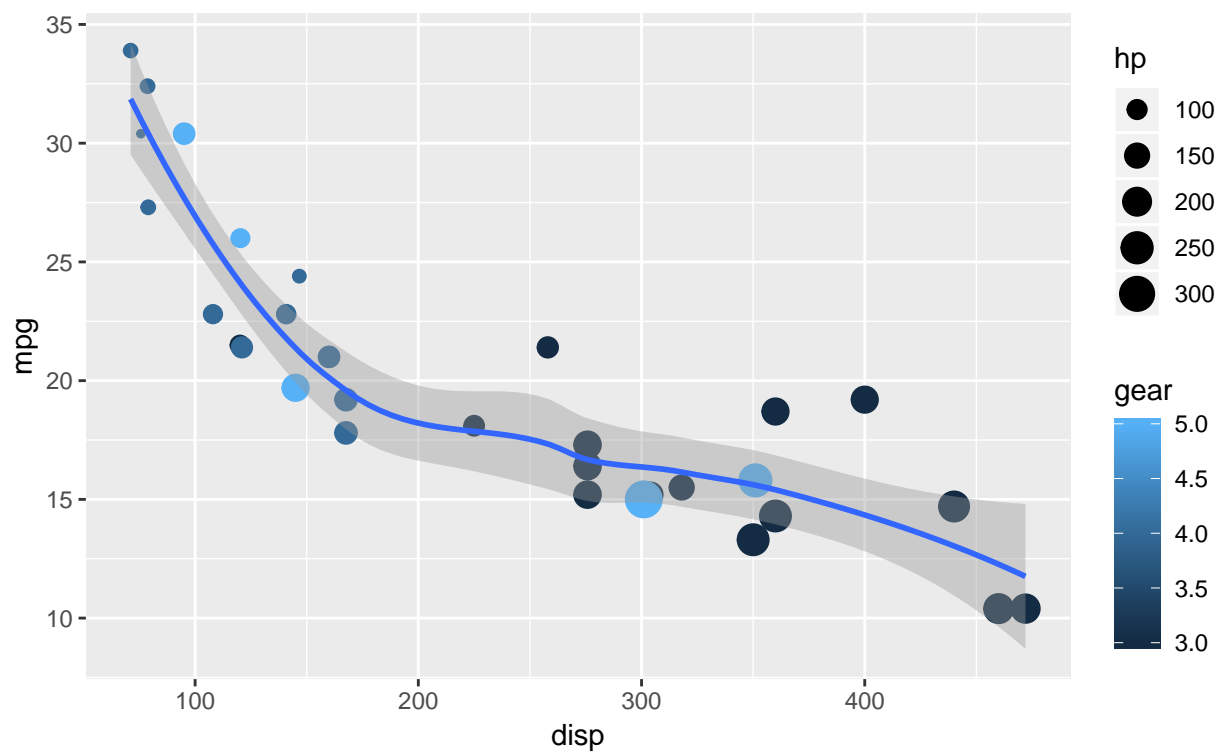
x	newnameofy
1	1
2	2

ggplot

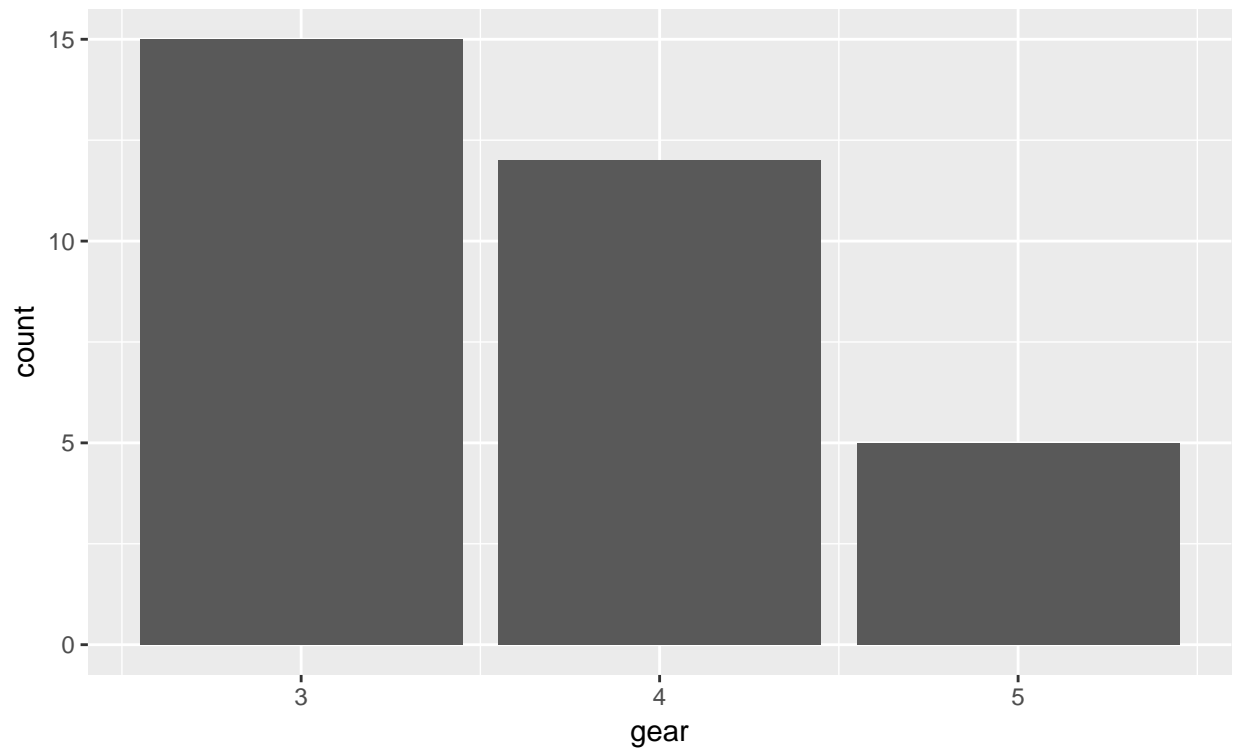
```
ggplot(mtcars ) + geom_point(aes(x= disp, y = mpg))
```



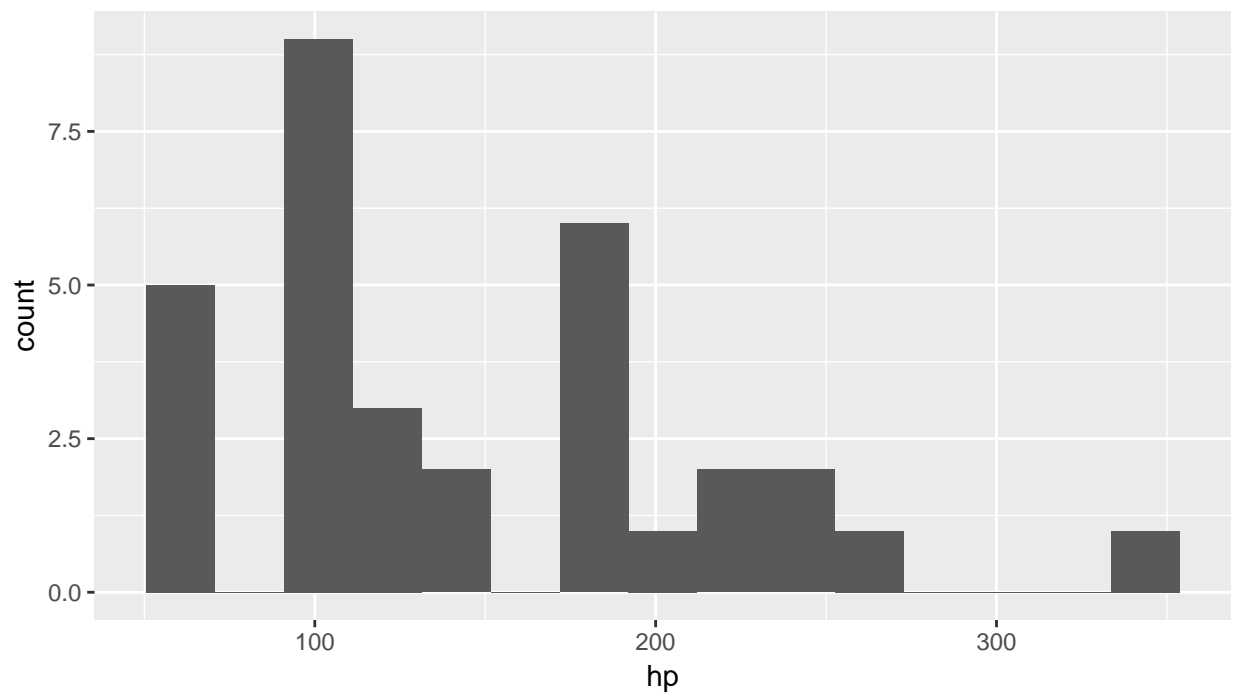
```
ggplot(mtcars, aes(x= disp, y = mpg))+geom_point(aes(
  size = hp,color = gear)) + geom_smooth()
```



```
ggplot(mtcars) + geom_bar(mapping = aes(x = gear))
```



```
ggplot(mtcars) + geom_histogram(mapping=aes(x=hp), bins=15)
```



Transforming data (tidyr package) - creating dataset

```
data <- data.frame(alpha = rep(LETTERS[1:3], 2),  
name = sort(rep(c("bobbur", "abhi"), 3)), year=2011:2016)
```

	alpha	name	year
1	A	abhi	2011
2	B	abhi	2012
3	C	abhi	2013
4	A	bobbur	2014
5	B	bobbur	2015
6	C	bobbur	2016

Transforming data - spread()

```
spread(data, key = name, value = year)
```

	alpha	abhi	bobbur
1	A	2011	2014
2	B	2012	2015
3	C	2013	2016

Transforming data - gather()

```
gather(data, abhi, bobbur, key = name, value = year)
```

	alpha	name	year
1	A	abhi	2011
2	B	abhi	2012
3	C	abhi	2013
4	A	bobbur	2014
5	B	bobbur	2015
6	C	bobbur	2016

Others - Joining datasets - rbind

```
data1 <- data  
data1$year <- data1$year + 1000  
data2 <- rbind(data, data1)  
head(data2, 8)
```

	alpha	name	year
1	A	abhi	2011
2	B	abhi	2012
3	C	abhi	2013
4	A	bobbur	2014
5	B	bobbur	2015
6	C	bobbur	2016
7	A	abhi	3011
8	B	abhi	3012

Joining datasets - merge

```
merge(data,data1, by = c("alpha","name"))
```

	alpha	name	year.x	year.y
1	A	abhi	2011	3011
2	A	bobbur	2014	3014
3	B	abhi	2012	3012
4	B	bobbur	2015	3015
5	C	abhi	2013	3013
6	C	bobbur	2016	3016

IF ELSE, IFELSE and for loops

```
x=2
if(x==1){print("hello World")} else if(x==2)
  {print("hello!")} else {print("!")}
data1$dummy <- ifelse(data1$year>3012, 1, 0)
for (i in 1:nrow(data1)){data1$dummy1[i] <-
  data1$dummy[i]+1 }
data1$name <- NULL
```

```
[1] "hello!"
```

	alpha	year	dummy	dummy1
1	A	3011	0	1
2	B	3012	0	1
3	C	3013	1	2

Row Selection

```
data4 <- data.frame(alpha = rep(LETTERS[1:3], 2), num = 1:6)
elim <- c(TRUE, FALSE, TRUE, TRUE, FALSE, FALSE)
data4[elim,]
data4[num > 3,]
data4[data4$num >3,]
```

	alpha	num	beta
1	A	1	b
3	C	3	b
4	A	4	a

Row selection

```
data4[data4$num >3,]
data4[c(1,2,2,2),]
```

	alpha	num	beta
4	A	4	a
5	B	5	a
6	C	6	a

	alpha	num	beta
1	A	1	b
2	B	2	b
2.1	B	2	b
2.2	B	2	b

Selection of columns

```
data4[,c(2,3)] %>% head(n=3)
data4[,c('num', 'alpha')] %>% head(n=3)
```

	num	beta
1	1	b
2	2	b
3	3	b

	num	alpha
1	1	A
2	2	B
3	3	C

duplicates

```
data4[duplicated(data4$alpha),]
data4[!duplicated(data4$beta),]
```

	alpha	num	beta
4	A	4	a
5	B	5	a
6	C	6	a

	alpha	num	beta
1	A	1	b
4	A	4	a

order

```
data4[order(data4$alpha),] %>% head(n=3)
data4[order(data4$alpha, data4$beta),] %>% head(n=3)
```

	alpha	num	beta
1	A	1	b
4	A	4	a
2	B	2	b

	alpha	num	beta
4	A	4	a
1	A	1	b
5	B	5	a

Thank you!