STOR 565 Spring 2019 Homework 1

COLEMAN BREEN

Remark. This homework aims to introduce you to the basics in **R**, which would be the main software we shall work on throughout this course. It might look like a long homework but it has only 13 problems. The rest are explanations regarding basic things in R. **Total number of points**: 120. Recall all Homeworks are worth the same weight when I compute your final grade. I will convert each HW to out of 100 then and compute the average of your HW scores.

Instruction. Download the whole folder for this homework and unzip the file. Open the **RMarkdown** document with surfix ".rmd" via **RStudio**. Click `Knit` to create a PDF document (remember to install the necessary packages in your **R**). The "knit" option can be changed to pdf *or* html (or even Word). When submitting the HW, you **will knit to a pdf document print the output and handover in class**. The html file in the folder has clickable links for the various references below on Latex etc. By removing the `results='hide'` and `fig.keep='none'` options in the code chunks, the code outputs and the plots will display in the created file. For more information about the **RStudio**, refer to the section **Getting Started**; about the **RMarkdown**, refer to the online tutorial (http://rmarkdown.rstudio.com/lesson-1.html) and the online manual of `knitr` (https://yihui.name/knitr/) by Yihui Xie from **RStudio, Inc.**

Latex: If the file does not compile properly in the intial go around you might need to install Latex onto your system. See Latex installation (https://www.latex-project.org/get/) for details. You might then need to reinstall RStudio. Latex is a fantastic framework that everyone in the computational world uses to write technical documents. See Tex exchange (https://tex.stackexchange.com/questions/1756/why-should-i-use-latex) or Medium (https://medium.com/@marko_kovic/why-i-write-with-latex-and-why-you-should-too-ba6a764fadf9) article for more details as to why you should use Latex. See this guide (http://www.docs.is.ed.ac.uk/skills/documents/3722/3722-2014.pdf) for a beginners introduction. I do not envision you having to use a ton of Latex in the course but this is important for me when making these assignments for the assignment to typeset properly especially with reference to math symbols.

diagram package: For displaying some of the pictures in the file we used an R package called diagram. You will need to install this before the file knits properly.

> **Please turn off the display of example code chunks (by specifying `include=FALSE`), complete the exercise code chunks (remember to turn on the `eval` option), fill in your name and create a PDF document, then print and submit it.**

Getting Started

R is one of the most common coding language in data analysis nowadays. It is free, open-source and powerful software environment for statistical computing and graphics. A nice description of **R** can be found in the course website (http://data.princeton.edu/R/) of German Rodriguez from Princeton. To download and install **R**, click into the CRAN (Comprehensive R Archive Network) Mirrors (https://cran.r-project.org/mirrors.html) and choose a URL that applies to you. To run with **R**, click "RGui.exe" in Windows system or "R.app" in Mac OS.

The console of **R** is not the most friendly interface to work with, as compared to the more handful editor **RStudio**. You can download and install it via its website (https://www.rstudio.com/products/rstudio/download/) and choose the free version of **RStudio Desktop**. Note that when you run with **RStudio**, the mirror of **R** should have been installed in your system, even though you are not accessing to it directly.

The ensuing sections lead you to the essentials of **R**. To explore more about how **R** works, please refer to (Dalgaard 2008, W. N. Venables and R Core Team (2017)) and the tutorial (http://data.princeton.edu/R/) of German Rodriguez.

Basics in **R**

R is an object-oriented language. Hence the “data” we work on are formatted as a particular object that meets some structural requirements (subjected to a particular class). Thus one should first understand which class of object he/she has on hand, and then figure out the applicable operations on it.

In a hierarchical manner, the more advanced class consists of ingredients from more fundamental classes. Vectors, matrices, lists, data frames and factors are the most commonly used fundamental classes in data analysis.

Vectors and Matrices

Vector is a collection of “data” that share the same type (numeric, character, logic or NULL). Matrix arranges “data” of the same type in two dimensions. Note that there doesn’t exist a “scalar” object, which would be treated as a vector of length 1.

Create a Vector

The concatenation function `c( )` can be used to manually create a vector in **R**. When using the `c( )` function, numbers are entered as a list with commas between each new entry. For example, `x <- c(1, 2)` creates a vector and assigns it to the variable `x` .

To create a vector that repeats n times, we can use the replication function `rep( , n)` . For example, a vector of five TRUE’s can be obtained by `x <- rep(TRUE, 5)` .

Finally, we can create a consecutive sequence of numbers using the sequence generating function `seq(from = , to = , by = )` . Here, the `from` , `to` and `by` arguments specify where the sequence begins, ends, and by how much the sequence increments. For example, the vector $(2, 4, 6, 8)$ can be obtained using `x <- seq(2 , 8, 2)` . A convenient operator `:` similar to `seq` also creates the consecutive sequence with step sizes by 1 or -1. Try `1:4` and `4:1` .

For more information, the commands `?c` , `?rep` and `?seq` access to the online **R** documents for help.

Exercise 1. *(5 pt)* Using the `c` , `rep` or `seq` commands, create the following 6 vectors:

x1 = (2, .5, 4, 2);

x2 = (2, .5, 4, 2, 1, 1, 1, 1);

x3 = (1, 0, -1, -2);

x4 = (“Hello”," “,”World“,”!“,”Hello World!“);

Note: The quotation marks and sometimes the exclamations marks are rendered a little funky in the pdf/html. Just go with it.

Hint. For x4, take this opportunity to experiment with the `paste` function.

x5 = (TRUE, TRUE, NA, FALSE);

Remark. Check `?NA` and `class(NA)` to learn more about the missing value object `NA`. This is not relevant for x5.

x6 = (1, 2, 1, 2, 1, 1, 2, 2).

```
x1 <- c(2, .5, 4, 2)
print(x1)
```

```
## [1] 2.0 0.5 4.0 2.0
```

```
x2 <- c(x1, rep(1, 4))
print(x2)
```

```
## [1] 2.0 0.5 4.0 2.0 1.0 1.0 1.0 1.0
```

```
x3 = -1*seq(from=-1, to=2)
print(x3)
```

```
## [1]  1  0 -1 -2
```

```
temp <- c("Hello", " ", "World", "!")
x4 <- c(temp, paste(temp, sep="", collapse=""))
print(x4)
```

```
## [1] "Hello"        " "            "World"        "!"
## [5] "Hello World!"
```

```
x5 <- c(T, T, NA, F)
print(x5)
```

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

```
x6 <- c(rep(c(1, 2), 2), rep(1, 2), rep(2,2))
print(x6)
```

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

Create a Matrix

An m-by-n matrix can be created by the command `matrix( , m, n)` where the first argument admits a vector with length compatible with the matrix dimensions. For example, `x <- matrix(1:4, 2, 2)` creates a 2-by-2 matrix that arranges the vector (1, 2, 3, 4) by column. To arrange the vector by row, specify the `byrow` option as follows: `x <- matrix(1:4, 2, 2, byrow = TRUE)` .

Moreover, the command binding vectors/matrices by row `rbind` and by column `cbind` are also useful. Check **R** documents for their usages.

Exercise 2. *(5 pt)* Using `matrix` , and `rbind` , create

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & -1 & -2 \\ 2 & .5 & 4 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

More precisely first define a set of four vectors corresponding to the rows of the above matrix and then use rbind to make a corresponding matrix. Note: you will need to play around with the `deparse.level` option in `rbind` to get the matrix as above.

```
r1 <- 1:4
r2 <- -1*seq(from=-1, to=2)
r3 <- c(2, .5, 4, 2)
r4 <- rep(1, 4)

X <- rbind(r1, r2, r3, r4, deparse.level=0) # gets rid of r1, r2, etc.
print(X)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1  2.0    3    4
## [2,]    1  0.0   -1   -2
## [3,]    2  0.5    4    2
## [4,]    1  1.0    1    1
```

Indexing

There are three ways to extract specific components in a vector.

The second approach leads to the so called “conditional selection” technique as follows.

Matrix indexing follows the same manner.

Exercise 3. *(4 pt)*: Consider the matrix X from Exercise 2.

- Make a new vector y1 consisting of all the elements of X which are negative (strictly less than zero).

```
y1 <- X[(X<0)]
print(y1)
```

```
## [1] -1 -2
```

- Make a new vector y2 consisting of all the elements of X which are at strictly positive but less than 2.

```
y2 <- X[(X>0 & X<2)]
print(y2)
```

```
## [1] 1.0 1.0 1.0 0.5 1.0 1.0 1.0
```

List

List is a more flexible container of "data" that permits inhomogeneous types. It's useful if you would like to encapsulate a bunch of components in an object. The `list` function explicitly specifies a list and the combining function `c` is still applicable. For example,

To extract the components in a list, one should use double bracket `[[ ]]` instead of a single bracket. If you've already specified the component names in a list, then the component names can be placed into the bracket directly. For example, `x[["logic"]]` accesses the third component of `x`. A more convenient alternative is `x$logic`.

Punchline Only ONE index instead of a vector of indices can be placed into the double bracket! Explore in the following example to see the difference as compared to the single bracket indexing.

Date Frame

Inheriting from matrix and list, data.frame is a container general enough for us to study a dataset. It permits inhomogeneous data types across columns (components in a list) but forces the components of the list to be vectors of homogeneous length (so as to be columns in a matrix). For example, the following creates a score table of 3 students.

To access the score_A of student 003, one can follow the manner in a matrix: `students[3,2]`, or that in a list: `students[[2]][3]`, `students[["score_A"]][3]` or `students$score_A[3]`.

Exercise 4. *(5 pt)* Applying the conditional selection technique (see the section "indexing" and do not use *subset*), extract the record of student 003 i.e their id number, and their scores in the two tests.

```
#--> Conditionally picking out the student whose id is 003
students[as.character(students$id)=='003',]
```

```
##    id score_A score_B
## 3 003      90      84
```

One can also create a matrix or a legitimate list first and then convert it into a data.frame as follows.

Exercise 5. *(10 pt)* Create a data.frame object to display the calendar for Jan 2018 as follows.

```
## Sun Mon Tue Wed Thu Fri Sat
##      NY   2   3   4   5   6
##   7   8   9  10  11  12  13
##  14 MLK  16  17  18  19  20
##  21  22  23  24  25  26  27
##  28  29  30  31
calendar <- data.frame(Sun = c("", 7, 14, 21, 28),
                       Mon = c("NY", 8, "MLK", 22, 29),
                       Tue = as.factor(seq(from=2, to=30, by=7)),
                       Wed = as.factor(seq(from=3, to=31, by=7)),
                       Thu = c(seq(from=4, to=25, by=7), ""),
                       Fri = c(seq(from=5, to=26, by=7), ""),
                       Sat = c(seq(from=6, to=27, by=7), ""))
print(calendar, row.names=FALSE)
```

```
##  Sun Mon Tue Wed Thu Fri Sat
##       NY   2   3   4   5   6
##    7   8   9  10  11  12  13
##   14 MLK  16  17  18  19  20
##   21  22  23  24  25  26  27
##   28  29  30  31
```

Ignore the ## symbols this was just so the above acts like a comment in R.

1) The character object `""` for the spaces; 2) the option `row.names = FALSE` in `print` function.

Factors

Factor is a special data structure in **R** in representing categorical variables and facilitating the data labels and subgroups. It's basically a character vector that keeps track of its distinct values called levels. Consider the longitudinal layout of the previous score table.

The `factor` function applied to a character vector creates a natural factor. The `gl` and `cut` functions are also useful approaches to patterned factors that are generated from numeric variables.

Exercise 6. *(5 pt)* Create a factor variable `grade` in `students3`, where the `score` variable is divided into $[90, 100]$, $[80, 90)$ and $[0, 80)$ corresponding to A, B and C in `grade` respectively.

Hint. Functions `cut` to obtain the grades and `transform` to obtain the students5 from stuents3.

```
library(tidyverse)
#--> More than one way to skin a cat
students5 <- students3 %>%
  mutate(grade = as.factor(
    ifelse(score < 80, "C", ifelse(score < 90, "B", "A"))))

print(students5)
```

```
##    id subj score grade
## 1 001    A    95     A
## 2 002    A    97     A
## 3 003    A    90     A
## 4 001    B    80     B
## 5 002    B    75     C
## 6 003    B    84     B
```

Operations and Functions

Note that scalar operations on vectors usually apply componentwise.

- **Arithmetic operations:** `+`, `-`, `*`, `/`, `^`, `%/%` (exact division), `%%` (modulus), `sqrt()`, `exp()`, `log()`
- **Logical operations**
  - And `&`, `&&`; or `|`, `||`; not `!`
  - Comparisons: `<`, `<=`, `>`, `>=`, `==` (different from `=`), `!=`
  - Summary functions: `all()`, `any()`
- **Summary statistics:** `length`, `max`, `min`, `sum`, `prod`, `mean`, `var`, `sd`, `median`, `quantile`
- **Matrix operations**
  - Matrix multiplication: `%*%` (different from `*`)
  - Related functions: `t`, `solve`, `det`, `diag`, `eigen`, `svd`, `qr`
  - Marginal operations: `apply`
- **Factors:** `tapply` (to apply operations/functions grouped by a factor)

Exercise 7. *(10 pt)* Without using the `var` and `scale` functions, compute the sample mean and sample covariance `X.var` of the data matrix `X` as in **Exercise 2.** More precisely, think of the i-th row of the matrix as observation of features for i-th individual.

a Create a 4-dimensional vector called `mu` where the i-th row is the mean of the i-th column of X.

```
#--> Sample mean
mu <- colSums(X) / 4
print(mu)
```

```
## [1] 1.250 0.875 1.750 1.250
```

b Create a four-dimensional matrix `X.var`

$$X.var = \frac{1}{3}\sum_{i=1}^{4}(\mathbf{x}_{i\cdot} - \mu)(\mathbf{x}_{i\cdot} - \mu)^T$$

where $\mathbf{x}_{i\cdot}$ is the i-th row.

```
X.var <- rep(NA, 4)

for (j in 1:4) {
  mean <- mu[j]
  vector <- X[,j]
  vector <- vector - mean
  vector <- vector ^ 2
  X.var[j] <- 1/3*sum(vector)
}

print(X.var)
```

```
## [1] 0.2500000 0.7291667 4.9166667 6.2500000
```

Exercise 8. *(10 pt)* Imagine that we wanted to make students aware for each of their subjects, the average score of all other students in that subject. Create a variable (or column) called `score.mean` in `students3`, where next to each student and subject, the value of the score.mean is the average value of all students taking that subject.

```
students3 %>%
  group_by(subj) %>%
  mutate(score.mean = mean(score)) %>%
  ungroup()
```

```
## # A tibble: 6 x 4
##   id    subj  score score.mean
##   <fct> <fct> <dbl>      <dbl>
## 1 001   A        95       94
## 2 002   A        97       94
## 3 003   A        90       94
## 4 001   B        80       79.7
## 5 002   B        75       79.7
## 6 003   B        84       79.7
```

Writing your own functions

It's convenient to create a user-defined **R** function, where you might encapsulate a standard, complicated or tedious procedure/algorithm in a "black box" like any built-in functions as mentioned above, so that other users might only need to care about the input and output of your function regardless the details. See the following toy example.

Flow Control

To help you write down your own **R** programs, the following examples familiarize you with the conditional statements and loops.

Conditional Statements

Loops

Exercise 9. *(15 pt)* The bisection method if a root-finding algorithm from numerical analysis to find a root of a continuous function in an interval $[a, b]$ once you know that function has different signs at the end points of the interval (i.e. $f(a) < 0, f(b) > 0$ or vice-versa). Read about this in the Wikipedia link (https://en.wikipedia.org/wiki/Bisection_method).

Write a function `bisect(f, lower, upper, tol = 1e-6)` to find the root of the univariate function `f` on the interval [`lower` , `upper`] with precision tolerance $\leq$ `tol` (defaulted to be 10^{-6}) via bisection, which returns a list consisting of `root` , `f.root` (`f` evaluated at `root`), `iter` (number of iterations) and `estim.prec` (estimated precision). Apply it to the function

$$f(x) = x^3 - 2x - 1$$

on $[1, 2]$ with precision tolerance 10^{-6}. Compare it with the built-in function `uniroot` .

```
f <- function(x){
  return((x^3) - 2 * x - 1)
}

bisect <- function(f, lower, upper, tol=10^(-6)){
  #--> Initialize variable
  iter <- 0

  #--> Rename for simplicity
  a <- lower
  b <- upper
  c <- (b + a) / 2

  while(abs(f(c)) > tol) {
    # f(a) and f(c) same sign
    if (f(a)*f(c) > 0){
      a <- c
    # Different signs
    } else {
      b <- c
    }
    # Update
    c <- (b + a) / 2
    iter <- iter + 1
  }
  return(list(c("root", c, "f.root", f(c), "iter",
              iter, "estim.prec", abs(2*f(c)))))
}

bisect(f, 1, 2)
```

```
## [[1]]
## [1] "root"                  "1.61803388595581"      "f.root"
## [4] "-6.0176699978598e-07" "iter"                  "20"
## [7] "estim.prec"            "1.20353399957196e-06"
```

```
uniroot(f, c(1, 2))
```

```
## $root
## [1] 1.618036
##
## $f.root
## [1] 9.230512e-06
##
## $iter
## [1] 6
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

Input/Output

`scan` and `read.table` are two main functions to read data. The main difference of them lies in that `scan` reads one component (also called “field”) at a time but `read.table` reads one line at a time. Hence `read.table` requires the data to be well-structured as a table so as to create a data.frame in **R** automatically, while `scan` can be flexible but might require effort in manipulating data after reading. Their usages are quite similar. One should pay attention to the frequently used options `file` , `header` , `sep` , `dec` , `skip` , `nmax` , `nlines` and `nrows` in their **R** documents.

`write.table` is a converse function against `read.table` , while their usages are almost identical. To get familiar with thier features, explore in the next exercise.

To read inline, one can specify `file = stdin()` (or omitted in `scan` function). In that case, it reads from console that the user can input line-by-line, or from the subsequent lines in a program script, until an empty line is read. However, such a trick is NOT compatible in **RMarkdown**.

If you have your data stored in another format, *e.g.* EXCEL or SAS dataset, then you can output it this as a CSV file and read in **R** via `read.csv` function (almost identical to `read.table`).

Exercise 10 *(16 pt)* In the folder for HW 1, you can find data on UNC salaries as a unc_salary_data.csv file (all of which are publicly available and scraped by Ryan Thornburg).

a Read the data using read.csv into a data frame called `salaries`

```
#--> Read in and display data
salaries <- read.csv("unc_salary_data.csv")
```

Use `str(salaries)` and `head(salaries)` to get an idea of the data set.

```
str(salaries)
```

```
## 'data.frame':	12287 obs. of  14 variables:
##  $ name    : Factor w/ 12270 levels "AARON, NANCY G",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ campus  : Factor w/ 1 level "UNC-CH": 1 1 1 1 1 1 1 1 1 1 ...
##  $ dept    : Factor w/ 304 levels "Acad Sup Prog Student-Athletes",..: 234 163 160
175 238 175 55 71 92 150 ...
##  $ position: Factor w/ 4120 levels "(NC DETECT) Program Director",..: 3597 634 347
4 41 3836 2282 41 3369 3361 1136 ...
##  $ exempt2 : Factor w/ 3 levels "Exempt","Non-permanent",..: 1 1 3 3 3 3 3 1 1 1 .
..
##  $ employed: int  9 9 12 12 12 12 12 12 12 12 ...
##  $ hiredate: int  20030701 19990101 20110912 20090420 20120103 20051003 19960923 2
0130401 19870101 20120702 ...
##  $ fte     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ status  : Factor w/ 5 levels "Continuing","Fixed-Term",..: 2 1 3 3 3 3 3 1 1 1
...
##  $ stservyr: int  11 17 3 5 2 15 34 11 27 2 ...
##  $ statesal: int  46350 173000 0 0 41696 56588 41707 0 0 0 ...
##  $ nonstsal: int  0 0 38170 50070 0 4412 0 80227 55803 32889 ...
##  $ totalsal: int  46350 173000 38170 50070 41696 61000 41707 80227 55803 32889 ...
##  $ age     : int  55 57 54 29 35 41 62 36 64 26 ...
```

```
head(salaries)
```

```
##                         name campus                           dept
## 1             AARON, NANCY G UNC-CH              Romance Languages
## 2     ABARBANELL, JEFFERY S UNC-CH Kenan-Flagler Business School
## 3               ABARE, BETSY UNC-CH  Institute of Marine Sciences
## 4             ABATE, AARON B UNC-CH        Medicine Administration
## 5        ABATEMARCO, JODI M UNC-CH            School of Education
## 6 ABBOTT-LUNSFORD, SHELBY L UNC-CH        Medicine Administration
##                     position                         exempt2 employed
## 1            Senior Lecturer                          Exempt        9
## 2        Associate Professor                          Exempt        9
## 3        Research Technician Subject to State Personnel Act       12
## 4      Accounting Technician Subject to State Personnel Act       12
## 5 Student Services Assistant Subject to State Personnel Act       12
## 6              HR Consultant Subject to State Personnel Act       12
##   hiredate fte     status stservyr statesal nonstsal totalsal age
## 1 20030701   1 Fixed-Term       11    46350        0    46350  55
## 2 19990101   1 Continuing       17   173000        0   173000  57
## 3 20110912   1  Permanent        3        0    38170    38170  54
## 4 20090420   1  Permanent        5        0    50070    50070  29
## 5 20120103   1  Permanent        2    41696        0    41696  35
## 6 20051003   1  Permanent       15    56588     4412    61000  41
```

b Make a new data frame called `relevant` consisting only of the columns: name, dept, age,totalsal. (Hint: consider the `subset` function).

```
#--> Subset
relevant <- salaries %>%
  select(name, dept, age, totalsal)

head(relevant)
```

```
##                        name                           dept age totalsal
## 1            AARON, NANCY G             Romance Languages  55    46350
## 2     ABARBANELL, JEFFERY S Kenan-Flagler Business School  57   173000
## 3               ABARE, BETSY  Institute of Marine Sciences  54    38170
## 4            ABATE, AARON B       Medicine Administration  29    50070
## 5        ABATEMARCO, JODI M           School of Education  35    41696
## 6 ABBOTT-LUNSFORD, SHELBY L       Medicine Administration  41    61000
```

c Make a new data frame called `top_200` consisting of the information in `relevant` of faculty who make more than $200,000.

```
top_200 <- relevant %>%
  filter(totalsal > 200000)

head(top_200)
```

```
##                 name                       dept age totalsal
## 1     ADAMS, SASHA D                    Surgery  42   271000
## 2 ADAMSON, WILLIAM T                    Surgery  50   410000
## 3  ADIMORA, ADAORA A                   Medicine  58   230614
## 4   AHALT, STANLEY C Renaissance Computing Inst  60   257940
## 5    AKINTEMI, OLA B                 Pediatrics  60   205919
## 6   AKULIAN, JASON A                   Medicine  38   230000
```

d Choose 3 departments that you are interested in. Compute the average salary of faculty in these 3 departments.

```
relevant %>%
  group_by(dept) %>%
  filter(dept %in% c("Biostatistics", "Mathematics", "Biology")) %>%
  summarize(dept_mean = mean(totalsal))
```

```
## # A tibble: 3 x 2
##   dept          dept_mean
##   <fct>             <dbl>
## 1 Biology          78392.
## 2 Biostatistics   102439.
## 3 Mathematics      93061.
```

Probability and Distributions

This section explores how to create “randomness” in **R** and obtain probabilistic quantities.

Discrete Random Sampling

Much of the earliest work in probability theory starts with random sampling, *e.g.* from a well-shuffled pack of cards or a well-stirred urn. The `sample` function applies such procedure to a vector in **R**. Learn more from the **R** documents.

The following exercise means to create a five-fold cross-validating sets, which would be the starting point to assess the performance of a learned machine in, for example, classification errors.

Exercise 11. *(10 pt)* `iris` is a built-in dataset in **R**. Check `?iris` for more information. This dataset has data on 50 flowers each from 3 species of Iris (setosa, versicolor, and virginica). Randomly divide `iris` into five subsets `iris1` to `iris5` (without replacement), thus each subset has 30 rows of the iris data and further stratified to `iris$Species` (namely every subset should have 10 rows from each of the 3 species).

```
sample_index <- sample(1:150, 150, replace = FALSE, prob = NULL) # sample (almost lik
e scrambling)
iris1 <- iris[sample_index[1:30], ]
iris2 <- iris[sample_index[31:60], ]
iris3 <- iris[sample_index[61:90], ]
iris4 <- iris[sample_index[91:120], ]
iris5 <- iris[sample_index[121:150], ]

iris.5fold <- list(iris1, iris2, iris3, iris4, iris5)
iris.5fold
```

```
## [[1]]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 137          6.3         3.4          5.6         2.4  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 102          5.8         2.7          5.1         1.9  virginica
## 109          6.7         2.5          5.8         1.8  virginica
## 103          7.1         3.0          5.9         2.1  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 53           6.9         3.1          4.9         1.5 versicolor
## 78           6.7         3.0          5.0         1.7 versicolor
## 30           4.7         3.2          1.6         0.2     setosa
## 101          6.3         3.3          6.0         2.5  virginica
## 73           6.3         2.5          4.9         1.5 versicolor
## 14           4.3         3.0          1.1         0.1     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 79           6.0         2.9          4.5         1.5 versicolor
## 10           4.9         3.1          1.5         0.1     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 34           5.5         4.2          1.4         0.2     setosa
## 148          6.5         3.0          5.2         2.0  virginica
## 76           6.6         3.0          4.4         1.4 versicolor
## 98           6.2         2.9          4.3         1.3 versicolor
## 123          7.7         2.8          6.7         2.0  virginica
## 16           5.7         4.4          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 39           4.4         3.0          1.3         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 126          7.2         3.2          6.0         1.8  virginica
## 106          7.6         3.0          6.6         2.1  virginica
## 56           5.7         2.8          4.5         1.3 versicolor
## 108          7.3         2.9          6.3         1.8  virginica
##
## [[2]]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 97           5.7         2.9          4.2         1.3 versicolor
## 22           5.1         3.7          1.5         0.4     setosa
## 86           6.0         3.4          4.5         1.6 versicolor
## 83           5.8         2.7          3.9         1.2 versicolor
## 117          6.5         3.0          5.5         1.8  virginica
## 58           4.9         2.4          3.3         1.0 versicolor
## 146          6.7         3.0          5.2         2.3  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 4            4.6         3.1          1.5         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 49           5.3         3.7          1.5         0.2     setosa
## 63           6.0         2.2          4.0         1.0 versicolor
## 113          6.8         3.0          5.5         2.1  virginica
## 149          6.2         3.4          5.4         2.3  virginica
```

```
## 100          5.7         2.8          4.1         1.3 versicolor
## 71           5.9         3.2          4.8         1.8 versicolor
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 27           5.0         3.4          1.6         0.4     setosa
## 50           5.0         3.3          1.4         0.2     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 92           6.1         3.0          4.6         1.4 versicolor
## 120          6.0         2.2          5.0         1.5  virginica
## 12           4.8         3.4          1.6         0.2     setosa
## 40           5.1         3.4          1.5         0.2     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 144          6.8         3.2          5.9         2.3  virginica
## 57           6.3         3.3          4.7         1.6 versicolor
## 88           6.3         2.3          4.4         1.3 versicolor
##
## [[3]]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 47           5.1         3.8          1.6         0.2     setosa
## 90           5.5         2.5          4.0         1.3 versicolor
## 138          6.4         3.1          5.5         1.8  virginica
## 89           5.6         3.0          4.1         1.3 versicolor
## 80           5.7         2.6          3.5         1.0 versicolor
## 41           5.0         3.5          1.3         0.3     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 36           5.0         3.2          1.2         0.2     setosa
## 119          7.7         2.6          6.9         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
## 44           5.0         3.5          1.6         0.6     setosa
## 133          6.4         2.8          5.6         2.2  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 93           5.8         2.6          4.0         1.2 versicolor
## 82           5.5         2.4          3.7         1.0 versicolor
## 7            4.6         3.4          1.4         0.3     setosa
## 116          6.4         3.2          5.3         2.3  virginica
## 111          6.5         3.2          5.1         2.0  virginica
## 64           6.1         2.9          4.7         1.4 versicolor
## 91           5.5         2.6          4.4         1.2 versicolor
## 60           5.2         2.7          3.9         1.4 versicolor
## 81           5.5         2.4          3.8         1.1 versicolor
## 134          6.3         2.8          5.1         1.5  virginica
## 114          5.7         2.5          5.0         2.0  virginica
## 105          6.5         3.0          5.8         2.2  virginica
## 59           6.6         2.9          4.6         1.3 versicolor
## 94           5.0         2.3          3.3         1.0 versicolor
## 31           4.8         3.1          1.6         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 75           6.4         2.9          4.3         1.3 versicolor
##
## [[4]]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
```

```
## 127          6.2         2.8          4.8         1.8  virginica
## 84           6.0         2.7          5.1         1.6 versicolor
## 48           4.6         3.2          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 62           5.9         3.0          4.2         1.5 versicolor
## 99           5.1         2.5          3.0         1.1 versicolor
## 128          6.1         3.0          4.9         1.8  virginica
## 85           5.4         3.0          4.5         1.5 versicolor
## 110          7.2         3.6          6.1         2.5  virginica
## 38           4.9         3.6          1.4         0.1     setosa
## 135          6.1         2.6          5.6         1.4  virginica
## 1            5.1         3.5          1.4         0.2     setosa
## 45           5.1         3.8          1.9         0.4     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 136          7.7         3.0          6.1         2.3  virginica
## 107          4.9         2.5          4.5         1.7  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 19           5.7         3.8          1.7         0.3     setosa
## 142          6.9         3.1          5.1         2.3  virginica
## 26           5.0         3.0          1.6         0.2     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 72           6.1         2.8          4.0         1.3 versicolor
## 35           4.9         3.1          1.5         0.2     setosa
## 112          6.4         2.7          5.3         1.9  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 13           4.8         3.0          1.4         0.1     setosa
## 46           4.8         3.0          1.4         0.3     setosa
## 77           6.8         2.8          4.8         1.4 versicolor
## 124          6.3         2.7          4.9         1.8  virginica
##
## [[5]]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 37           5.5         3.5          1.3         0.2     setosa
## 139          6.0         3.0          4.8         1.8  virginica
## 67           5.6         3.0          4.5         1.5 versicolor
## 87           6.7         3.1          4.7         1.5 versicolor
## 69           6.2         2.2          4.5         1.5 versicolor
## 65           5.6         2.9          3.6         1.3 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
## 118          7.7         3.8          6.7         2.2  virginica
## 11           5.4         3.7          1.5         0.2     setosa
## 96           5.7         3.0          4.2         1.2 versicolor
## 104          6.3         2.9          5.6         1.8  virginica
## 51           7.0         3.2          4.7         1.4 versicolor
## 9            4.4         2.9          1.4         0.2     setosa
## 147          6.3         2.5          5.0         1.9  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 33           5.2         4.1          1.5         0.1     setosa
## 55           6.5         2.8          4.6         1.5 versicolor
## 15           5.8         4.0          1.2         0.2     setosa
```

```
## 42          4.5         2.3          1.3         0.3     setosa
## 66          6.7         3.1          4.4         1.4 versicolor
## 70          5.6         2.5          3.9         1.1 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 140         6.9         3.1          5.4         2.1  virginica
## 54          5.5         2.3          4.0         1.3 versicolor
## 115         5.8         2.8          5.1         2.4  virginica
## 95          5.6         2.7          4.2         1.3 versicolor
## 5           5.0         3.6          1.4         0.2     setosa
## 43          4.4         3.2          1.3         0.2     setosa
## 68          5.8         2.7          4.1         1.0 versicolor
```

Distributions

R is endowed with a set of statistical tables. To obtain the density function, cumulative distribution function (CDF), quantile (inverse CDF) and pseudo-random numbers from a specific distribution, one only needs to prefix the distribution name given below by `d` , `p` , `q` and `r` respectively.

Distributions	R Names	Key Arguments
Uniform	`unif`	`min` , `max`
Normal	`norm`	`mean` , `sd`
χ^2	`chisq`	`df` , `ncp`
Student's t	`t`	`df` , `ncp`
F	`f`	`df1` , `df2` , `ncp`
Exponential	`exp`	`rate`
Gamma	`gamma`	`shape` , `scale`
Beta	`beta`	`shape1` , `shape2` , `ncp`
Logistic	`logis`	`location` , `scale`
Binomial	`binom`	`size` , `prob`
Poisson	`pois`	`lambda`
Geometric	`geom`	`prob`
Hypergeometric	`hyper`	`m` , `n` , `k`
Negative Binomial	`nbinom`	`size` , `prob`

Check from their plots.

```
plot(dnorm, xlim = c(-5, 5))    # bell curve of Normal density
plot(plogis, xlim = c(-5, 5))   # Logistic/Sigmoid function (CDF of Logistic distribut
ion)
```

The following two-sample t-test shows the usages of `qt` , `pt` and `rnorm` . Recall that a two-sample homoscedastic t-test statistic is

$$\hat{\sigma}^2 = \frac{(n_X - 1)S_X^2 + (n_Y - 1)S_Y^2}{n_X + n_Y - 2}, \quad T = \frac{\bar{X} - \bar{Y}}{\hat{\sigma}\sqrt{\frac{1}{n_X} + \frac{1}{n_Y}}} \overset{d}{\sim} t_{n_X + n_Y - 2} \text{ under } H_0 :\ \mu_X = \mu_Y.$$

Data Exploration and Manipulation

Data analysis in **R** starts with reading data in a data.frame object via `scan` and `read.table` as discussed before. Then one would explore the profiles of data via various descriptive statistics whose usages are also introduced in the previous sections. Calling the `summary` function with a data.frame input also provides appropriate summaries, *e.g.* means and quantiles for numeric variables (columns) and frequencies for factor variables.

This section explores more features that can be achieved through **R**.

Tables

Statistician often works with catergorical variables via tables. Even for continuous variables, segregating them into catergorical ones in a meaningful way might provide more insights. `table` function generates frequency tables for factor variables. Multi-way tables, marginal and proportional displays and independence test are explored in the following example. Recall that the Pearson's χ^2 independence test statistic on an r-by-c contingency table

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\left(n_{ij} - \frac{n_{i\cdot}n_{\cdot j}}{n_{\cdot\cdot}}\right)^2}{n_{i\cdot}n_{\cdot j}/n_{\cdot\cdot}} \overset{d}{\approx} \chi^2_{(r-1)(c-1)} \text{ under } H_0 : p_{ij} = p_i p_j.$$

Plots

Compared to other statistical softwares in data analysis, **R** is good at graphic generation and manipulation. The plotting functions in **R** can be classified into the high-level ones and the low-level ones. The high-level functions create complete, new plots on the graphics device while the low-level functions only add extra information to the current plots.

`plot` is the most generic high-level plotting function in **R**. It will be compatble with most class of objects that the user input and produce appropriate graphics. For example, a numeric vector input results in a scatter plot with respect to its index and a factor vector results in a bar plot of its frequency table. Advanced class of object like `lm` (fitted result by a linear model) can also be called in `plot` . Methods will be discussed in specific documents like `?plot.lm` .

Other plotting features include

- High-level plotting options: `type` , `main` , `sub` , `xlab` , `ylab` , `xlim` , `ylim`

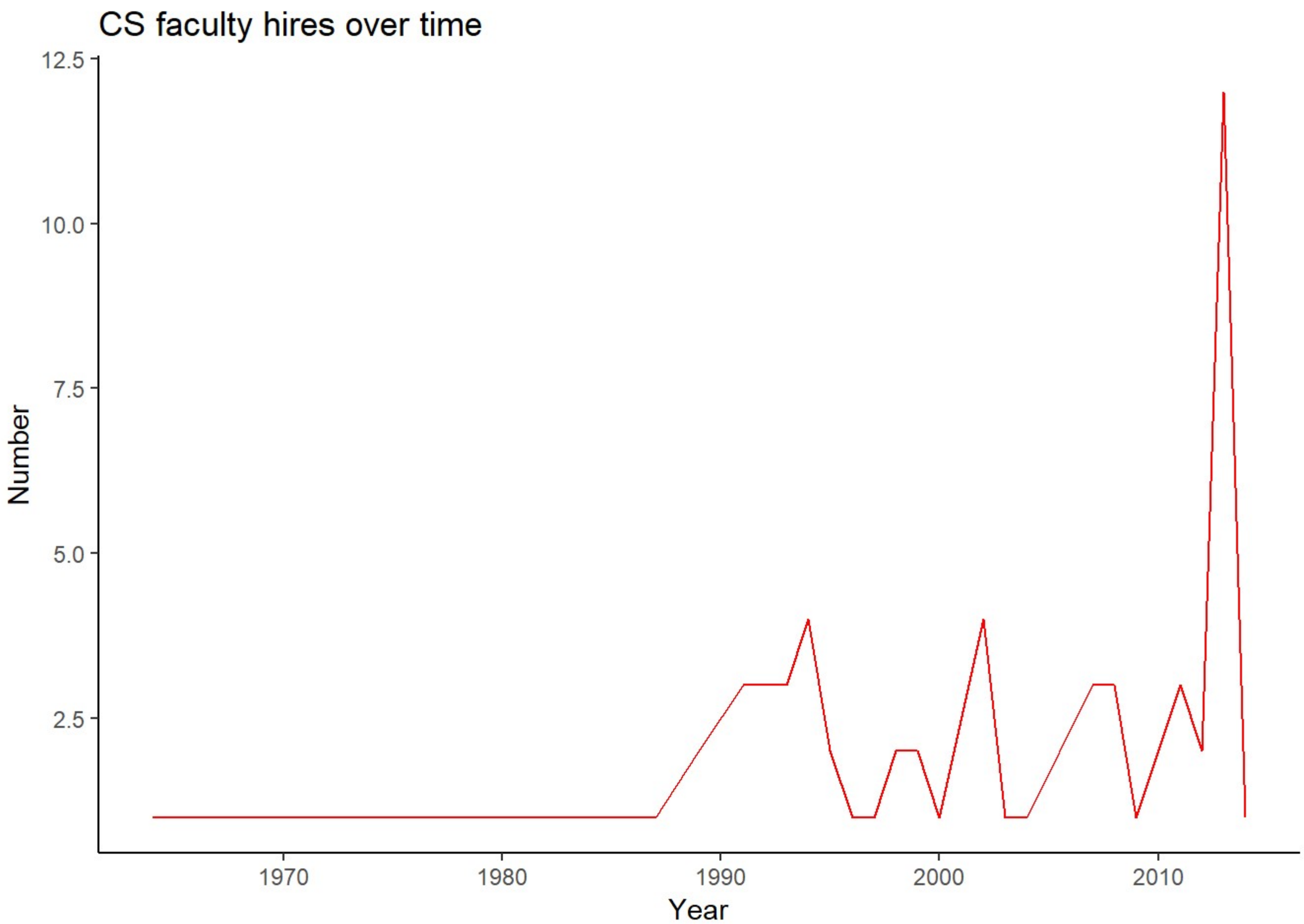
- Low-level plotting functions
  - **Symbols:** `points` , `lines` , `text` , `abline` , `segments` , `arrows` , `rect` , `polygon`
  - **Decorations:** `title` , `legend` , `axis`
- Environmental graphic options (`?par`)
  - **Symbols and texts:** `pch` , `cex` , `col` , `font`
  - **Lines:** `lty` , `lwd`
  - **Axes:** `tck` , `tcl` , `xaxt` , `yaxt`
  - **Windows:** `mfcol` , `mfrow` , `mar` , `new`
- User interaction: `location`

I'll suggest the beginners learn from examples and grab when needed instead of going over such an overwhelming brochure. The following sections illustrate two basic senarios in data analysis. More high-level plotting functions will also be introduced.

Exercise 12 *(10 pt)*

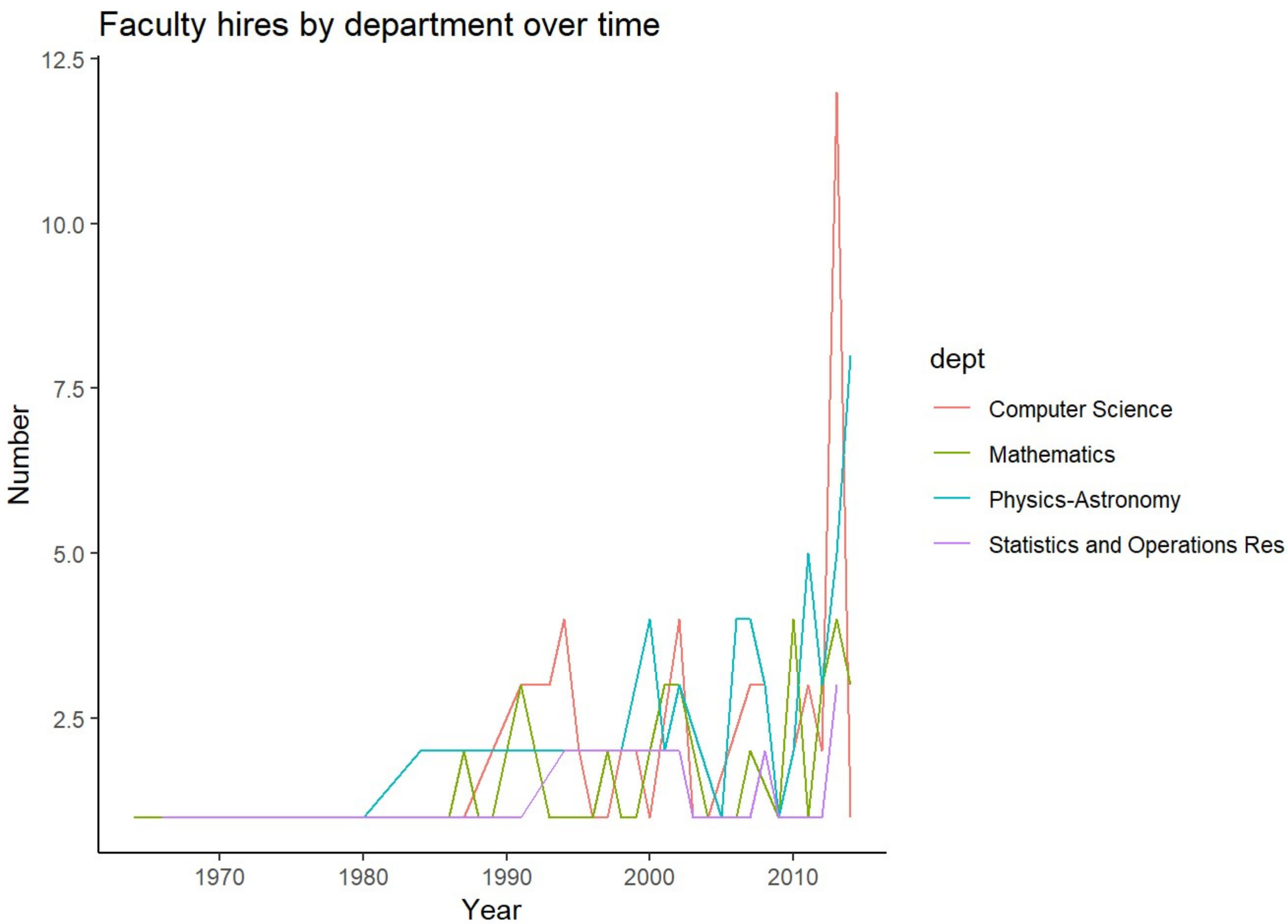
a Recall the UNC salary data set. From the `salaries` data frame plot the number of CS faculty hired per year vs year.

```
salaries %>%
  filter(dept == "Computer Science") %>%
  mutate(hire_year = as.integer(hiredate/10000)) %>%
  count(hire_year) %>%
  ggplot(aes(x=hire_year, y=n)) +
  geom_line(color="red") +
  ggtitle("CS faculty hires over time") +
  xlab("Year") +
  ylab("Number") +
  theme_classic()
```



b Now add STOR, Math and Physics to the above plot

```
salaries %>%
  filter(dept %in% c("Computer Science", "Mathematics",
                     "Physics-Astronomy", "Statistics and Operations Res")) %>%
  group_by(dept) %>%
  mutate(hire_year = as.integer(hiredate/10000)) %>%
  count(hire_year)%>%
  ggplot(aes(x=hire_year, y=n, color=dept)) +
  geom_line() +
  theme_classic() +
  xlab("Year") +
  ylab("Number") +
  ggtitle("Faculty hires by department over time")
```



Access the empirical distribution

Histogram: The `hist` function creates a histogram in **R**, where the `breaks` and `freq` options are frequently called. The `barplot` function could also realize the same result as illustrated in section **Tables**.

Kernel Density Curve: The `density` function estimates an empirical density of the data and gives a `density` object. One can call `plot` function with such an object as input and picture a density curve, which is anticipated to closely envelop its historgram. Note that the `bw` (bandwidth) and `kernel` options should be carefully considered in methodology.

Empirical CDF (ECDF): The `ecdf` function generates an empirical CDF (`"ecdf"`) object that can be called by the `plot` function, which results in a step function graphic for the empirical cumulative distribution function. Creating a graph containing multiple CDFs or ECDFs visually displays the good-of-fitness or discrepancy among them. The statistically quantified Kolmogorov-Smirnov test with statistic

$$D_n = \sup_{x \in \mathbb{R}} |F_n(x) - F(x)|$$

realized by the `ks.test` function is based on it.

Q-Q Plots: “Q-Q” stands for the sample quantiles versus that of a given distribution or another sample. `qqnorm` , `qqline` and `qqplot` together is the set of functions in realizing it. **R** documents also illustrate how to create Q-Q plots against distributions other than Normal.

Box-and-Whisker Plots: With `boxplot` function in **R**, we can describe the data with box associated with certain quantiles and the whiskers for extremes. The box in the middle indicates “hinges” (nearly quartiles, see `?boxplot.stats`) and median. The lines (“whiskers”) show the largest/smallest observation that falls within a distance of 1.5 times the box size from the nearest hinge. If any observations fall farther away, the additional points are considered “extreme” values and are shown separately.

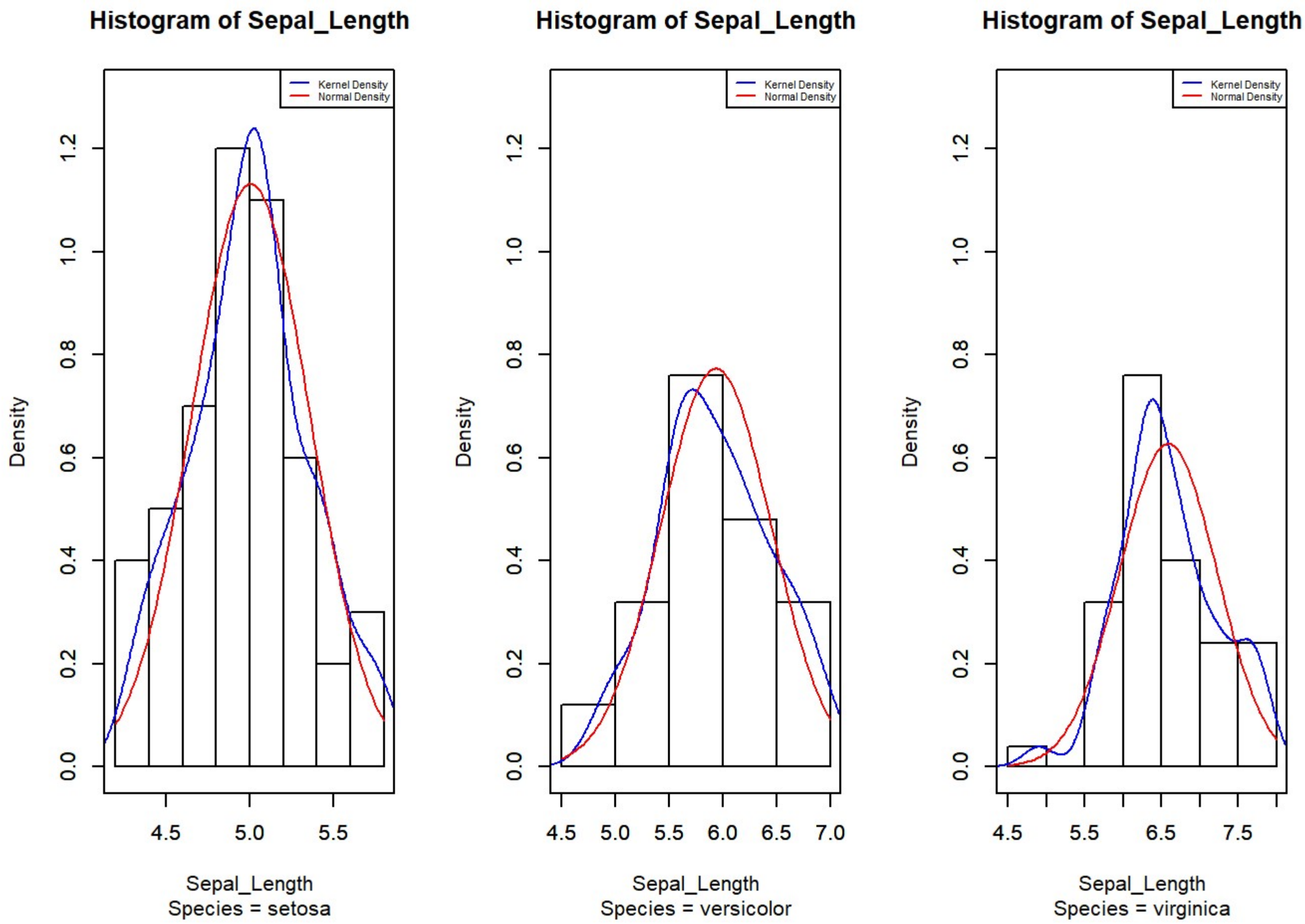
Exercise 13. *(15 pt)* The following code generates the ensuing plot about `Sepal.Length` in `iris`.

```
opar <- par(mfrow = c(1,3))
for(l in levels(iris$Species))
{
  Sepal_Length <- subset(iris, Species == l, select = Sepal.Length)[[1]]
  h <- hist(Sepal_Length, sub = paste("Species =", l), ylim = c(0,1.3), freq = FALSE)

  par(new = TRUE)   # add to the current plot
  # Empirical density curve
  lines(density(Sepal_Length),
       xlim = range(h$breaks), ylim = c(0,1.3),  # to match the plotting range
       col = "blue",
       main = "", sub = "", xlab = "", ylab = ""    # to supress labels
       )
  par(new = TRUE)   # add to the current plot

  # Normal density curve
  curve(dnorm(x, mean = mean(Sepal_Length), sd = sd(Sepal_Length)),
        xlim = range(h$breaks), ylim = c(0,1.3),  # to match the plotting range
        col = "red",
        main = "", sub = "", xlab = "", ylab = ""    # to supress labels
        )

  legend("topright",
         legend = c("Kernel Density", "Normal Density"),
         col = c("blue", "red"), lty = 1, cex = 0.5)
}
```




```
par(opar)
```

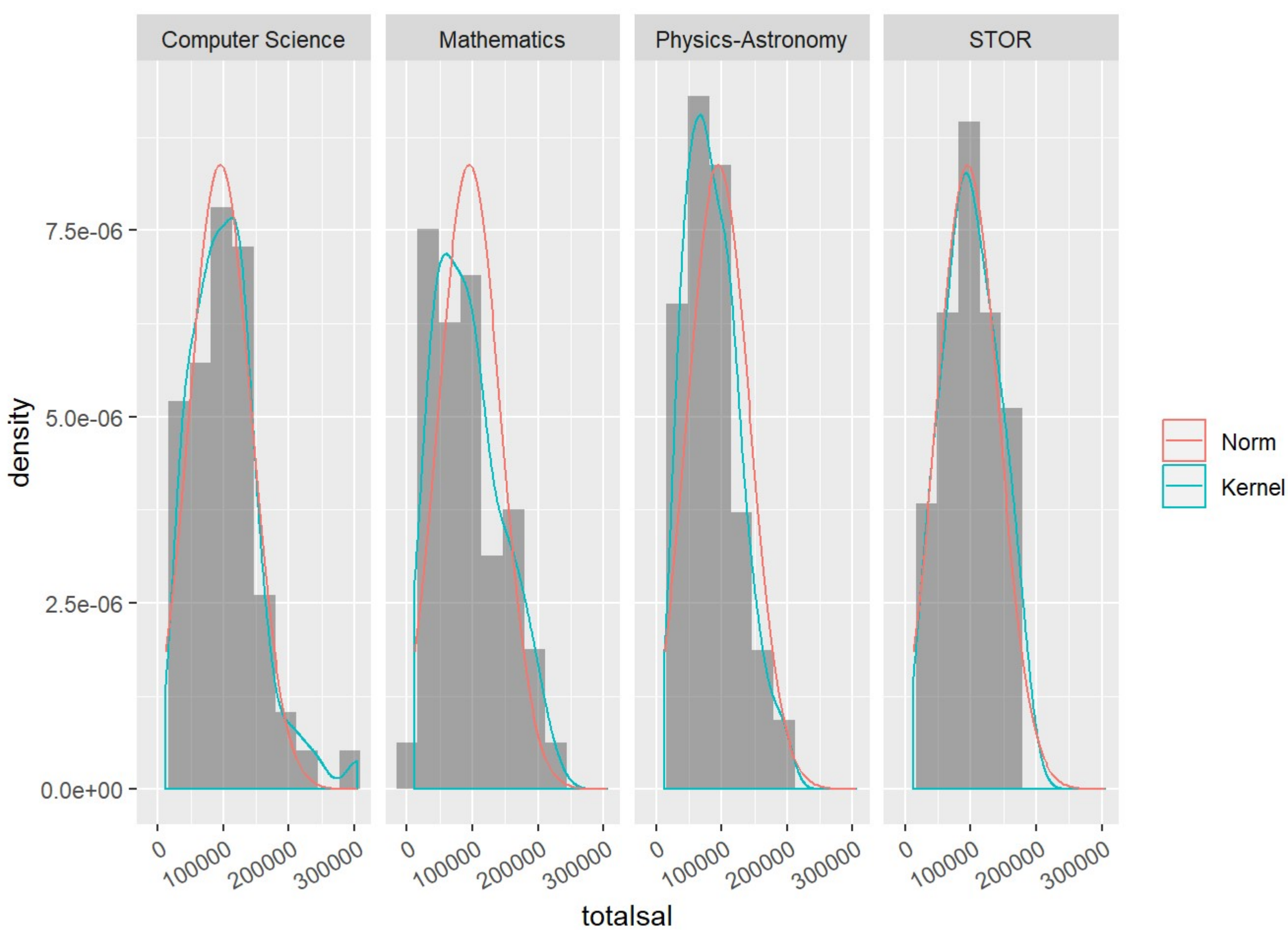
Plot generated:

Either modify the code above or use your own code to obtain similar plots with histograms, kernel density plots and normal density plots for the salary of faculty in CS, Math and Physics from UNC salary data.

```r
#--> Filter and group
salaries2 <- salaries %>%
  filter(dept %in% c("Computer Science", "Mathematics",
                     "Physics-Astronomy", "Statistics and Operations Res"))

#--> Add new factor level
levels(salaries2$dept) <- c(levels(salaries2$dept), "STOR")
salaries2$dept[salaries2$dept == 'Statistics and Operations Res'] <- 'STOR'


#--> Plot
salaries2 %>%
  ggplot(aes(x=totalsal)) +
  geom_histogram(aes(y=..density..), bins=10, alpha = .5)+
  geom_density(aes(color="red")) +
  stat_function(fun = dnorm,
                args = with(salaries2, c(mean = mean(totalsal),
                                         sd = sd(totalsal))), aes(color="blue")) +
  scale_x_continuous(labels=function(n){format(n, scientific = FALSE)}) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  scale_color_discrete(name = "", labels = c("Norm", "Kernel")) +
  facet_grid(. ~ dept)
```



Demostrate correlation

`plot(x, y)` directly creates a scatter plot between the vector `x` and `y`. For a data.frame input `X`, `plot(X)` would conduct pairwise scatter plots among its columns. A fitted regression line can also be added to an existing scatter plot via the `abline` function. And the function `coplot` is a power function in creating conditioning plots given a variable segregated into different levels.

*Create advanced plots

If you are a JAVA programmer, then you might anticipate a plotting toolbox to establish graphs layer-by-layer interactively. The `ggplot2` package endows **R** with more advanced and powerful visualization techniques like this. Explore more in the online package manual (https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf). The following example solves **Exercise 11** without segregating with respect to `Species` in `iris`.

*Data Manipulation

If you are more familiar with the SQL language in manimulating data, then the `dplyr` package in **R** is a powerful toolkit in providing functions similar to the SQL operations (*e.g.* `select`, `filter`, `arrange`, `mutate`, `inner_join`, `group_by`, `summarise` and the pipe operator `%>%`). The following example realizes the conditional selection technique in the previous exercises in a much cleaner way. Explore more in the online package mannual (https://cran.r-project.org/web/packages/dplyr/dplyr.pdf).

Bibliography

Dalgaard, Peter. 2008. *Introductory Statistics with R: Statistics and Computing*. Springer.

W. N. Venables, D. M. Smith, and the R Core Team. 2017. *An Introduction to R: Notes on R: A Programming Environment for Data Analysis and Graphics* (version 3.4.3).