Data Science and R

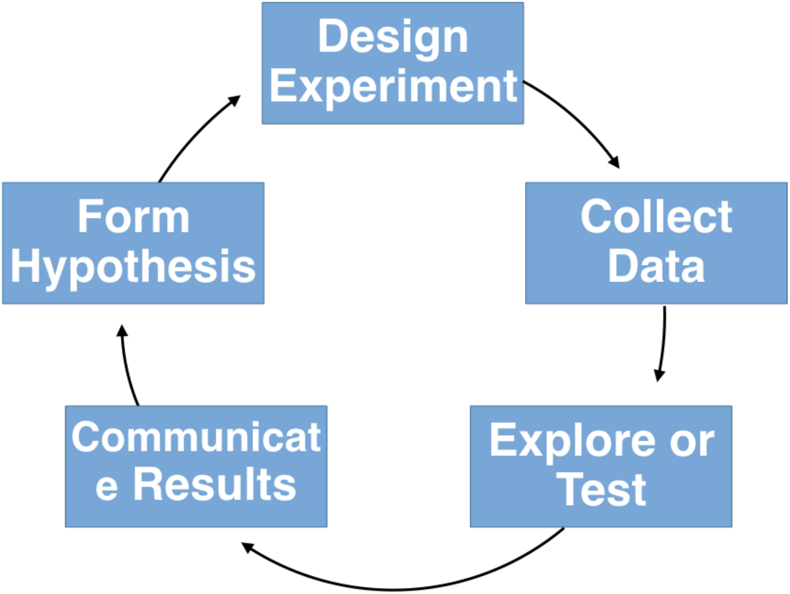
Chang-Ye Tu

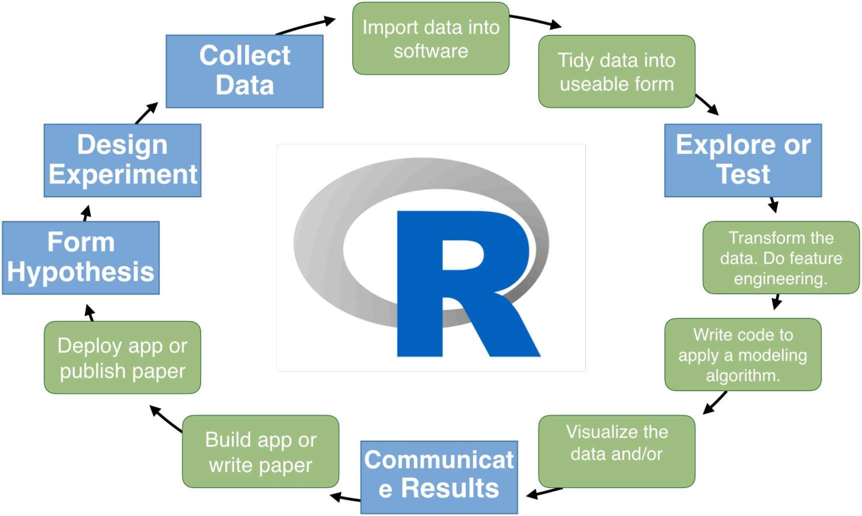
Before We Start

What is Data Science?

Data science is a discipline that turns raw data into understanding, insight, and knowledge.

- Recommender Systems to Improve Sales
- Google AdSense
- Understand the Customer Behavior
- ...


Design Experiment
Collect Data
Explore or Test
Communicat e Results
Form Hypothesis


Import data into software
Tidy data into useable form
Explore or Test
Transform the data. Do feature engineering.
Write code to apply a modeling algorithm.
Visualize the data and/or
Communicate Results
Build app or write paper
Deploy app or publish paper
Form Hypothesis
Design Experiment
Collect Data

I Would Like to Know...

- What is your experience with R?

- What programming experience other than R do you have?

- How are you using, or how do you plan to use, R in your job?

Software

The R Programming Language

- Go to the offical site.
- Click “Download R for Mac/Windows”.
- Follow the instructions of the installer

RStudio (IDE)

- Choose “RStudio for Desktop”.
- Select the install file for your OS
- Follow the instructions of the installer

Introductory R

First R Steps I

- R as a calculator — show the numerical result of $1 + 2 \times \frac{3}{4} - 5 + 6 \sin \frac{\pi}{2} - \sqrt[8]{7}$

- Print the content of the built-in dataset **Nile**

First R Steps I

- R as a calculator — show the numerical result of $1 + 2 \times \frac{3}{4} - 5 + 6 \sin \frac{\pi}{2} - \sqrt[8]{7}$

```
1 + 2 * (3 / 4) - 5 + 6 * sin(pi / 2) - 7^(1/8)
# [1] 2.224627
```

- Print the content of the built-in dataset **Nile**

First R Steps I

- R as a calculator — show the numerical result of $1 + 2 \times \frac{3}{4} - 5 + 6 \sin \frac{\pi}{2} - \sqrt[8]{7}$

```
1 + 2 * (3 / 4) - 5 + 6 * sin(pi / 2) - 7^(1/8)
# [1] 2.224627
```

- Print the content of the built-in dataset **Nile**

```
Nile
# Time Series:
# Start = 1871
# End = 1970
# Frequency = 1
#   [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140  995  935 1110  994 1020
#  [16]  960 1180  799  958 1140 1100 1210 1150 1250 1260 1220 1030 1100  774  840
#  [31]  874  694  940  833  701  916  692 1020 1050  969  831  726  456  824  702
#  [46] 1120 1100  832  764  821  768  845  864  862  698  845  744  796 1040  759
#  [61]  781  865  845  944  984  897  822 1010  771  676  649  846  812  742  801
#  [76] 1040  860  874  848  890  744  749  838 1050  918  986  797  923  975  815
#  [91] 1020  906  901 1170  912  746  919  718  714  740
str(Nile)
#  Time-Series [1:100] from 1871 to 1970: 1120 1160 963 1210 1160 1160 813 1230 1370
```

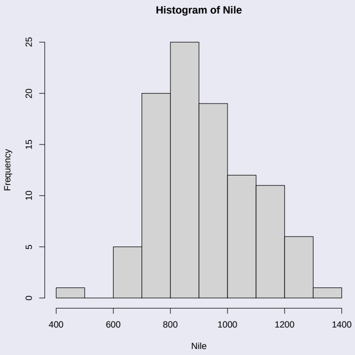
First R Steps II

- Show the histogram of **Nile**
- Show the plot of **Nile**

First R Steps II

- Show the histogram of **Nile**

```r
hist(Nile)
```

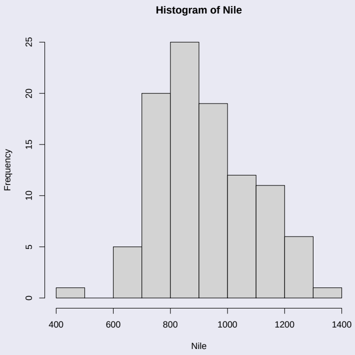


- Show the plot of **Nile**

First R Steps II

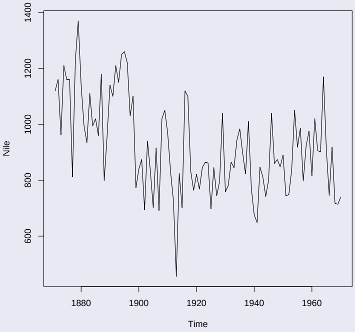
- Show the histogram of **Nile**

```r
hist(Nile)
```



- Show the plot of **Nile**

```r
plot(Nile)
```



Vectors and Indices I

- The 2nd item of the **Nile** dataset

- Subsequence taken from 2nd to 5th item of **Nile**

- Subsequence taken from 2nd to 5th item of **Nile**: using the important concatenate function c

- Subsequence taken from 10th, 3rd, 15th item of **Nile**

Vectors and Indices I

- The 2nd item of the **Nile** dataset

```
Nile[2]
# [1] 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**

- Subsequence taken from 2nd to 5th item of **Nile**: using the important concatenate function c

- Subsequence taken from 10th, 3rd, 15th item of **Nile**

Vectors and Indices I

- The 2nd item of the **Nile** dataset

```
Nile[2]
# [1] 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**

```
Nile[2:5]
# [1] 1160  963 1210 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**: using the important concatenate function c

- Subsequence taken from 10th, 3rd, 15th item of **Nile**

Vectors and Indices I

- The 2nd item of the **Nile** dataset

```
Nile[2]
# [1] 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**

```
Nile[2:5]
# [1] 1160  963 1210 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**: using the important concatenate function c

```
Nile[c(2, 3, 4, 5)]
# [1] 1160  963 1210 1160
```

- Subsequence taken from 10th, 3rd, 15th item of **Nile**

Vectors and Indices I

- The 2nd item of the **Nile** dataset

```
Nile[2]
# [1] 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**

```
Nile[2:5]
# [1] 1160  963 1210 1160
```

- Subsequence taken from 2nd to 5th item of **Nile**: using the important concatenate function c

```
Nile[c(2, 3, 4, 5)]
# [1] 1160  963 1210 1160
```

- Subsequence taken from 10th, 3rd, 15th item of **Nile**

```
Nile[c(10, 3, 15)]
# [1] 1140  963 1020
```

Vectors and Indices II

- Let `x` be a (row) vector $(10, 20, -1, 3, 5, 6, 30, 9)$, print `x`:
- Let `y` be a vector formed by taking the 2nd element out of `x`; print `y`:
- Let `z` be a vector formed by taking the 2nd, 3rd, 6th element out of `x`; print `z`:
- `w <- c(x, y, z)`; print `w`:

Vectors and Indices II

- Let `x` be a (row) vector $(10, 20, -1, 3, 5, 6, 30, 9)$, print `x`:

```
x <- c(10, 20, -1, 3, 5, 6, 30, 9); x
# [1] 10 20 -1  3  5  6 30  9
```

- Let `y` be a vector formed by taking the 2nd element out of `x`; print `y`:

- Let `z` be a vector formed by taking the 2nd, 3rd, 6th element out of `x`; print `z`:

- `w <- c(x, y, z)`; print `w`:

Vectors and Indices II

- Let x be a (row) vector $(10, 20, -1, 3, 5, 6, 30, 9)$, print x:

```
x <- c(10, 20, -1, 3, 5, 6, 30, 9); x
# [1] 10 20 -1  3  5  6 30  9
```

- Let y be a vector formed by taking the 2nd element out of x; print y:

```
y <- x[-2]; y
# [1] 10 -1  3  5  6 30  9
```

- Let z be a vector formed by taking the 2nd, 3rd, 6th element out of x; print z:

- `w <- c(x, y, z)`; print w:

Vectors and Indices II

- Let x be a (row) vector $(10, 20, -1, 3, 5, 6, 30, 9)$, print x:

```
x <- c(10, 20, -1, 3, 5, 6, 30, 9); x
# [1] 10 20 -1  3  5  6 30  9
```

- Let y be a vector formed by taking the 2nd element out of x; print y:

```
y <- x[-2]; y
# [1] 10 -1  3  5  6 30  9
```

- Let z be a vector formed by taking the 2nd, 3rd, 6th element out of x; print z:

```
z <- x[c(-2, -3, -6)]; z
# [1] 10  3  5 30  9
```

- `w <- c(x, y, z)`; print `w`:

Vectors and Indices II

- Let x be a (row) vector $(10, 20, -1, 3, 5, 6, 30, 9)$, print x:

```
x <- c(10, 20, -1, 3, 5, 6, 30, 9); x
# [1] 10 20 -1  3  5  6 30  9
```

- Let y be a vector formed by taking the 2nd element out of x; print y:

```
y <- x[-2]; y
# [1] 10 -1  3  5  6 30  9
```

- Let z be a vector formed by taking the 2nd, 3rd, 6th element out of x; print z:

```
z <- x[c(-2, -3, -6)]; z
# [1] 10  3  5 30  9
```

- `w <- c(x, y, z)`; print w:

```
w <- c(x, y, z); w
#  [1] 10 20 -1  3  5  6 30  9 10 -1  3  5  6 30  9 10  3  5 30  9
```

Vectors and Indices III

- Subsequence taken from 81th to 95th item of **Nile**

- Mean of the subsequence `Nile[81:95]`

- Lengths of **Nile**

- Lengths of the subsequence `Nile[81:95]`

Vectors and Indices III

- Subsequence taken from 81th to 95th item of **Nile**

```
Nile[81:95]
#  [1]  744  749  838 1050  918  986  797  923  975  815 1020  906  901 1170  912
```

- Mean of the subsequence `Nile[81:95]`

- Lengths of **Nile**

- Lengths of the subsequence `Nile[81:95]`

Vectors and Indices III

- Subsequence taken from 81th to 95th item of **Nile**

```
Nile[81:95]
#  [1]  744  749  838 1050  918  986  797  923  975  815 1020  906  901 1170  912
```

- Mean of the subsequence `Nile[81:95]`

```
mean(Nile[81:95])
# [1] 913.6
```

- Lengths of **Nile**

- Lengths of the subsequence `Nile[81:95]`

Vectors and Indices III

- Subsequence taken from 81th to 95th item of **Nile**

```
Nile[81:95]
#  [1]  744  749  838 1050  918  986  797  923  975  815 1020  906  901 1170  912
```

- Mean of the subsequence `Nile[81:95]`

```
mean(Nile[81:95])
# [1] 913.6
```

- Lengths of **Nile**

```
length(Nile)
# [1] 100
```

- Lengths of the subsequence `Nile[81:95]`

Vectors and Indices III

- Subsequence taken from 81th to 95th item of **Nile**

```
Nile[81:95]
#  [1]  744  749  838 1050  918  986  797  923  975  815 1020  906  901 1170  912
```

- Mean of the subsequence `Nile[81:95]`

```
mean(Nile[81:95])
# [1] 913.6
```

- Lengths of **Nile**

```
length(Nile)
# [1] 100
```

- Lengths of the subsequence `Nile[81:95]`

```
length(Nile[81:95])
# [1] 15
```

Vectors and Indices IV

- Let the vector `x <- c(20, 1, 15, 13, 12)`. Print `x > 14`:

- How many items in `x` that > 14 ?

- Which of the items in **Nile** are > 1200 ?

- Print out the items in **Nile** that > 1200.

Vectors and Indices IV

- Let the vector `x <- c(20, 1, 15, 13, 12)`. Print `x > 14`:

```
x <- c(20, 1, 15, 13, 12); x > 14
# [1]  TRUE FALSE  TRUE FALSE FALSE
```

- How many items in `x` that > 14 ?

- Which of the items in **Nile** are > 1200 ?

- Print out the items in **Nile** that > 1200.

Vectors and Indices IV

- Let the vector `x <- c(20, 1, 15, 13, 12)`. Print `x > 14`:

```
x <- c(20, 1, 15, 13, 12); x > 14
# [1]  TRUE FALSE  TRUE FALSE FALSE
```

- How many items in `x` that > 14 ?

```
sum(x > 14)
# [1] 2
```

- Which of the items in **Nile** are > 1200 ?

- Print out the items in **Nile** that > 1200.

Vectors and Indices IV

- Let the vector `x <- c(20, 1, 15, 13, 12)`. Print `x > 14`:

```
x <- c(20, 1, 15, 13, 12); x > 14
# [1]  TRUE FALSE  TRUE FALSE FALSE
```

- How many items in `x` that > 14 ?

```
sum(x > 14)
# [1] 2
```

- Which of the items in **Nile** are > 1200 ?

```
which(Nile > 1200)
# [1]  4  8  9 22 24 25 26
```

- Print out the items in **Nile** that > 1200.

Vectors and Indices IV

- Let the vector `x <- c(20, 1, 15, 13, 12)`. Print `x > 14`:

```
x <- c(20, 1, 15, 13, 12); x > 14
# [1]  TRUE FALSE  TRUE FALSE FALSE
```

- How many items in `x` that > 14 ?

```
sum(x > 14)
# [1] 2
```

- Which of the items in **Nile** are > 1200 ?

```
which(Nile > 1200)
# [1]  4  8  9 22 24 25 26
```

- Print out the items in **Nile** that > 1200.

```
Nile[Nile > 1200]
# [1] 1210 1230 1370 1210 1250 1260 1220
```

Data Frame I

- data frame: a rectangular table consisting of one row for each data point. Built-in data frame **ToothGrowth**, assigned `tg`.
- len: tooth length; supp: supplement VC (vitamin c) or OJ (orange juice); dose
- Each column is a vector; use $ to extract

Data Frame I

- data frame: a rectangular table consisting of one row for each data point. Built-in data frame **ToothGrowth**, assigned `tg`.
- len: tooth length; supp: supplement VC (vitamin c) or OJ (orange juice); dose

```
head(ToothGrowth)
#    len supp dose
# 1  4.2   VC  0.5
# 2 11.5   VC  0.5
# 3  7.3   VC  0.5
# 4  5.8   VC  0.5
# 5  6.4   VC  0.5
# 6 10.0   VC  0.5
```

- Each column is a vector; use $ to extract

Data Frame I

- data frame: a rectangular table consisting of one row for each data point. Built-in data frame **ToothGrowth**, assigned `tg`.
- len: tooth length; supp: supplement VC (vitamin c) or OJ (orange juice); dose

```
head(ToothGrowth)

#    len supp dose
# 1  4.2   VC  0.5
# 2 11.5   VC  0.5
# 3  7.3   VC  0.5
# 4  5.8   VC  0.5
# 5  6.4   VC  0.5
# 6 10.0   VC  0.5
```

- Each column is a vector; use $ to extract

```
tg <- ToothGrowth; tg$len

#  [1]  4.2 11.5  7.3  5.8  6.4 10.0 11.2 11.2  5.2  7.0 16.5 16.5 15.2 17.3 22.5
# [16] 17.3 13.6 14.5 18.8 15.5 23.6 18.5 33.9 25.5 26.4 32.5 26.7 21.5 23.3 29.5
# [31] 15.2 21.5 17.6  9.7 14.5 10.0  8.2  9.4 16.5  9.7 19.7 23.3 23.6 26.4 20.0
# [46] 25.2 25.8 21.2 14.5 27.3 25.5 26.4 22.4 24.5 24.8 30.9 26.4 27.3 29.4 23.0
```

Data Frame II

- To get the item of `tg` in row 3, column 1

- To get the item of `tg` in row 3, column 1: using that `tg$len` is a vector

- Extract rows 2 through 5, and columns 1 and 3, assigning the result to `z`

Data Frame II

- To get the item of `tg` in row 3, column 1

```
tg[1, 3]
# [1] 0.5
```

- To get the item of `tg` in row 3, column 1: using that `tg$len` is a vector

- Extract rows 2 through 5, and columns 1 and 3, assigning the result to `z`

Data Frame II

- To get the item of `tg` in row 3, column 1

```
tg[1, 3]
# [1] 0.5
```

- To get the item of `tg` in row 3, column 1: using that `tg$len` is a vector

```
tg$len[3]
# [1] 7.3
```

- Extract rows 2 through 5, and columns 1 and 3, assigning the result to `z`

Data Frame II

- To get the item of tg in row 3, column 1

```
tg[1, 3]
# [1] 0.5
```

- To get the item of tg in row 3, column 1: using that tg$len is a vector

```
tg$len[3]
# [1] 7.3
```

- Extract rows 2 through 5, and columns 1 and 3, assigning the result to z

```
z <- tg[2:5, c(1, 3)]; z
#    len dose
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
```

Data Frame III

- To get the items of `tg` in columns 1, 3
- To get the items of `tg` in columns 1, 3: remove column 2
- Create your own data frame from vectors of same lengths
- Note that vector of different lengths are not allowed !

Data Frame III

- To get the items of tg in columns 1, 3

```
head(tg[, c(1, 3)])
#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- To get the items of tg in columns 1, 3: remove column 2

- Create your own data frame from vectors of same lengths

- Note that vector of different lengths are not allowed !

Data Frame III

- To get the items of tg in columns 1, 3

```
head(tg[, c(1, 3)])
#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- To get the items of tg in columns 1, 3: remove column 2

```
head(tg[, -2])
#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- Create your own data frame from vectors of same lengths

- Note that vector of different lengths are not allowed !

Data Frame III

- To get the items of tg in columns 1, 3

```
head(tg[, c(1, 3)])

#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- To get the items of tg in columns 1, 3: remove column 2

```
head(tg[, -2])

#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- Create your own data frame from vectors of same lengths

```
age <- c(55, 58, 45)
name <- c('Alice', 'Bill', 'Cathy')
d <- data.frame(age, name)
d

#   age  name
# 1  55 Alice
# 2  58  Bill
# 3  45 Cathy
```

- Note that vector of different lengths are not allowed !

Data Frame III

- To get the items of tg in columns 1, 3

```
head(tg[, c(1, 3)])

#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- To get the items of tg in columns 1, 3: remove column 2

```
head(tg[, -2])

#    len dose
# 1  4.2  0.5
# 2 11.5  0.5
# 3  7.3  0.5
# 4  5.8  0.5
# 5  6.4  0.5
# 6 10.0  0.5
```

- Create your own data frame from vectors of same lengths

```
age <- c(55, 58, 45)
name <- c('Alice', 'Bill', 'Cathy')
d <- data.frame(age, name)
d

#   age  name
# 1  55 Alice
# 2  58  Bill
# 3  45 Cathy
```

- Note that vector of different lengths are not allowed !

```
blood <- c('O', 'B', 'A', 'AB')
data.frame(age, name, blood)

# Error in data.frame(age, name,
blood): arguments imply differing
number of rows: 3, 4
```

Extracting Rows / Columns from Data Frames

- In `tg`, compute mean tooth length for supplement VC and OJ

- Extract the sub data frame from `tg` with supplement OJ and length < 8.8

- Note the existence and the postion of the comma (,) and the use of == !

- Extract the sub data frame from `tg` with length > 28 or dose $= 1.0$

Extracting Rows / Columns from Data Frames

- In tg, compute mean tooth length for supplement VC and OJ

```
tg_vc <- tg[tg$supp == 'VC',]
tg_oj <- tg[tg$supp == 'OJ',]
mean(tg_vc$len)
# [1] 16.96333
mean(tg_oj$len)
# [1] 20.66333
```

- Extract the sub data frame from tg with supplement OJ and length < 8.8

- Note the existence and the postion of the comma (,) and the use of == !

- Extract the sub data frame from tg with length > 28 or dose $= 1.0$

Extracting Rows / Columns from Data Frames

- In tg, compute mean tooth length for supplement VC and OJ

```
tg_vc <- tg[tg$supp == 'VC',]
tg_oj <- tg[tg$supp == 'OJ',]
mean(tg_vc$len)
# [1] 16.96333
mean(tg_oj$len)
# [1] 20.66333
```

- Extract the sub data frame from tg with supplement OJ and length < 8.8

```
tg[tg$supp == 'OJ' & tg$len < 8.8,]
#    len supp dose
# 37 8.2   OJ  0.5
```

- Note the existence and the postion of the comma (,) and the use of == !

- Extract the sub data frame from tg with length > 28 or dose $= 1.0$

Extracting Rows / Columns from Data Frames

- In tg, compute mean tooth length for supplement VC and OJ

```
tg_vc <- tg[tg$supp == 'VC',]
tg_oj <- tg[tg$supp == 'OJ',]
mean(tg_vc$len)
# [1] 16.96333
mean(tg_oj$len)
# [1] 20.66333
```

- Extract the sub data frame from tg with supplement OJ and length < 8.8

```
tg[tg$supp == 'OJ' & tg$len < 8.8,]
#    len supp dose
# 37 8.2   OJ  0.5
```

- Note the existence and the postion of the comma (,) and the use of == !

- Extract the sub data frame from tg with length > 28 or dose $= 1.0$

```
w <- tg[tg$len == 28 | tg$dose == 1.0,]
head(w, 15)
#     len supp dose
# 11 16.5   VC    1
# 12 16.5   VC    1
# 13 15.2   VC    1
# 14 17.3   VC    1
# 15 22.5   VC    1
# 16 17.3   VC    1
# 17 13.6   VC    1
# 18 14.5   VC    1
# 19 18.8   VC    1
# 20 15.5   VC    1
# 41 19.7   OJ    1
# 42 23.3   OJ    1
# 43 23.6   OJ    1
# 44 26.4   OJ    1
# 45 20.0   OJ    1
```

The `tapply` Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}

- Using `tapply` in `tg` to compute the mean length for each supplement group.

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

The `tapply` Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}
- Split `age` into groups according to the corresponding elements of `gender`; find the mean in each group.

- Using `tapply` in `tg` to compute the mean length for each supplement group.

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

The tapply Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}
- Split age into groups according to the corresponding elements of `gender`; find the mean in each group.

```
age <- c(20, 16, 38, 55, 25)
gender <- c('M', 'F', 'M', 'F', 'F')
z <- tapply(age, gender, mean); z
#  F  M
# 32 29
```

- Using `tapply` in `tg` to compute the mean length for each supplement group.

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

The tapply Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}
- Split age into groups according to the corresponding elements of `gender`; find the mean in each group.

```
age <- c(20, 16, 38, 55, 25)
gender <- c('M', 'F', 'M', 'F', 'F')
z <- tapply(age, gender, mean); z
#  F  M
# 32 29
```

- Using `tapply` in `tg` to compute the mean length for each supplement group.

```
tapply(tg$len, tg$supp, mean)
#       OJ       VC
# 20.66333 16.96333
```

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

The tapply Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}
- Split age into groups according to the corresponding elements of `gender`; find the mean in each group.

```
age <- c(20, 16, 38, 55, 25)
gender <- c('M', 'F', 'M', 'F', 'F')
z <- tapply(age, gender, mean); z
#  F  M
# 32 29
```

- Using `tapply` in `tg` to compute the mean length for each supplement group.

```
tapply(tg$len, tg$supp, mean)
#       OJ       VC
# 20.66333 16.96333
```

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

```
tapply(mtcars$cyl, mtcars$cyl, length)
#  4  6  8
# 11  7 14
```

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

The tapply Function

- Imaging you have (age, gender) pairs {(20, 'M'), (16, 'F'), (38, 'M'), (55, 'F'), (25, 'F')}
- Split age into groups according to the corresponding elements of `gender`; find the mean in each group.

```
age <- c(20, 16, 38, 55, 25)
gender <- c('M', 'F', 'M', 'F', 'F')
z <- tapply(age, gender, mean); z
#  F  M
# 32 29
```

- Using `tapply` in `tg` to compute the mean length for each supplement group.

```
tapply(tg$len, tg$supp, mean)
#       OJ       VC
# 20.66333 16.96333
```

- In the built-in data **mtcars**, using `tapply` to find how many cars there are in each cylinder category.

```
tapply(mtcars$cyl, mtcars$cyl, length)
#  4  6  8
# 11  7 14
```

- In **mtcars**, using `tapply` to find mean and standard deviation of miles per gallon in each cylinder category.

```
tapply(mtcars$mpg, mtcars$cyl, mean)
#        4        6        8
# 26.66364 19.74286 15.10000
tapply(mtcars$mpg, mtcars$cyl, sd)
#        4        6        8
# 4.509828 1.453567 2.560048
```

The R List Class I

- In **mtcars**, split the miles-per-gallon (mpg) data according to the number of cylinders (cyl):

- Now vectors in `mtl`, a R list, can be accessed individually with `` $` ` `` and `[[ ]]`:

The R List Class I

- In **mtcars**, split the miles-per-gallon (mpg) data according to the number of cylinders (cyl):

```
mtl <- split(mtcars$mpg, mtcars$cyl); mtl
# $`4`
#  [1] 22.8 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
#
# $`6`
# [1] 21.0 21.0 21.4 18.1 19.2 17.8 19.7
#
# $`8`
#  [1] 18.7 14.3 16.4 17.3 15.2 10.4 10.4 14.7 15.5 15.2 13.3 19.2 15.8 15.0
class(mtl)
# [1] "list"
```

- Now vectors in `mtl`, a R list, can be accessed individually with `` $` ` `` and `[[ ]]`:

The R List Class I

- In **mtcars**, split the miles-per-gallon (mpg) data according to the number of cylinders (cyl):

```
mtl <- split(mtcars$mpg, mtcars$cyl); mtl
# $`4`
#  [1] 22.8 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
#
# $`6`
# [1] 21.0 21.0 21.4 18.1 19.2 17.8 19.7
#
# $`8`
#  [1] 18.7 14.3 16.4 17.3 15.2 10.4 10.4 14.7 15.5 15.2 13.3 19.2 15.8 15.0
class(mtl)
# [1] "list"
```

- Now vectors in `mtl`, a R list, can be accessed individually with `` $` ` `` and `[[ ]]`:

```
mtl$`6`
# [1] 21.0 21.0 21.4 18.1 19.2 17.8 19.7
mtl[[2]]
# [1] 21.0 21.0 21.4 18.1 19.2 17.8 19.7
```

The R List Class II

- In R, data frames can't hold vectors of different lengths, but lists can.

The R List Class II

- In R, data frames can't hold vectors of different lengths, but lists can.

```
vn <- c(1, 1.2, 2.3, 3.4, 4.5)
vb <- c(TRUE, TRUE, FALSE)
vc <- c('limestone', 'marl', 'oolite', 'CaCO3')
```

The R List Class II

- In R, data frames can't hold vectors of different lengths, but lists can.

```
vn <- c(1, 1.2, 2.3, 3.4, 4.5)
vb <- c(TRUE, TRUE, FALSE)
vc <- c('limestone', 'marl', 'oolite', 'CaCO3')
```

```
data.frame(vn, vb, vc)
# Error in data.frame(vn, vb, vc): arguments imply differing number of rows: 5, 3,
4
```

The R List Class II

- In R, data frames can't hold vectors of different lengths, but lists can.

```
vn <- c(1, 1.2, 2.3, 3.4, 4.5)
vb <- c(TRUE, TRUE, FALSE)
vc <- c('limestone', 'marl', 'oolite', 'CaCO3')
```

```
data.frame(vn, vb, vc)
# Error in data.frame(vn, vb, vc): arguments imply differing number of rows: 5, 3,
4
```

```
list(vn, vb, vc)
# [[1]]
# [1] 1.0 1.2 2.3 3.4 4.5
#
# [[2]]
# [1]  TRUE  TRUE FALSE
#
# [[3]]
# [1] "limestone" "marl"      "oolite"    "CaCO3"
```

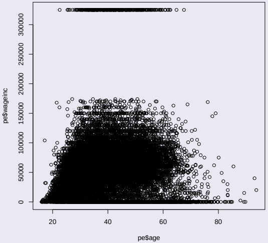
Base R Graphics I

```r
pe <- read.table('data/prgeng.txt', header=TRUE)
head(pe)

#          age educ occ sex wageinc wkswrkd
# 1 50.30082   13 102   2   75000      52
# 2 41.10139    9 101   1   12300      20
# 3 24.67374    9 102   2   15400      52
# 4 50.19951   11 100   1       0      52
# 5 51.18112   11 100   2     160       1
# 6 57.70413   11 100   1       0       0
```

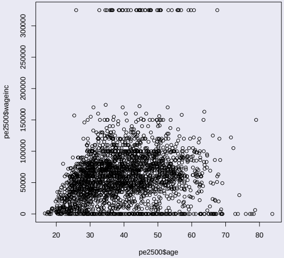
Base R Graphics I

```r
pe <- read.table('data/prgeng.txt', header=TRUE)
head(pe)

#         age educ occ sex wageinc wkswrkd
# 1 50.30082   13 102   2   75000      52
# 2 41.10139    9 101   1   12300      20
# 3 24.67374    9 102   2   15400      52
# 4 50.19951   11 100   1       0      52
# 5 51.18112   11 100   2     160       1
# 6 57.70413   11 100   1       0       0
```

```r
plot(pe$age, pe$wageinc)
```

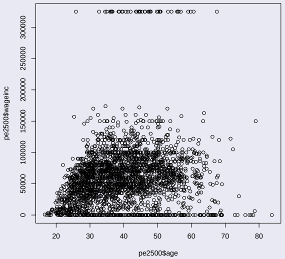


```
pe2500 <- pe[sample(1:nrow(pe), 2500),]
plot(pe2500$age, pe2500$wageinc)
```



Base R Graphics II

```r
pe2500 <- pe[sample(1:nrow(pe), 2500),]
plot(pe2500$age, pe2500$wageinc)
```




```r
plot(pe2500$age, pe2500$wageinc,
     col=as.factor(pe2500$sex),
     xlab='age', ylab='wage', cex=0.6)
```