Introduction to Data Mining with R

Olga Vitek
Acknowledgements: Hadley Wikham, Kyle Bemis

September 17, 2015

Outline

- Getting Started
- R Fundamentals
  - Data Structures
  - Subsetting
  - Functions
- Example of data analysis
  - Summarization
  - Functional Programming
- Data visualization
- Missing data
- The concept of tidy data
- Advanced data manipulation

Getting Started

- Installing R (`cran.r-project.org`)
- RStudio (`www.rstudio.com`)
- `install.packages("ggplot2")`
- `?help`

R Fundamentals

- Open-source implementation of S (Bell Labs, 1980)
- Also inspired by Scheme
- Functional language with OOP features
- Intefaces to C, C++, and FORTRAN
- Note: the R interpreter is written in C
- Thousands of user-written packages
- LaTeX-like documentation

R Fundamentals (cont'd)

```
> 1+1 # basic addition

[1] 2

> x <- 1 # assignment using `<-`
> print(x) # print value of `x`

[1] 1

> x # R automatically prints last value

[1] 1

> x + 1 # addition again

[1] 2
```

R Fundamentals (cont'd)

```
> x <- c(x, 2, 3, 4:9) # build vectors using `c`
> x

[1] 1 2 3 4 5 6 7 8 9

> 4:9 # use `i:j` for sequences from `i` to `j`

[1] 4 5 6 7 8 9

> x[2] # select 2nd element

[1] 2

> x[-2] # select all but 2nd element

[1] 1 3 4 5 6 7 8 9
```

R data structures

	Homogenous	Heterogenous
1d	`atomic`	`list`
2d	`matrix`	`data.frame`
nd	`array`	

Atomic vectors

In R, **everything** is a vector

- `logical` - boolean values (TRUE, FALSE, *and* NA)
- `numeric` - real and integer values
  - `integer` - 32 bit int
  - `double` - 64 bit double
- `character` - (vector of) strings
- `raw` - stream of bytes (rarely used)
- `complex` - complex values (rarely used)

Atomic vectors (cont'd)

```
> logical(1)
[1] FALSE
> numeric(1)
[1] 0
> character(1)
[1] ""
> raw(1)
[1] 00
> complex(1)
[1] 0+0i
```

Matrices and arrays

Matrices and arrays are any of the atomic types plus **dimensions.**

```
> matrix(1:9, nrow=3, ncol=3) # make a 3x3 matrix

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> x <- 1:9 # or turn a 1d vector in a matrix...
> dim(x) <- c(3,3) # ...by giving it dimensions
> x

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Matrices and arrays (cont'd)

```
> array(1:18, dim=c(3,3,2)) # make a 3x3x2 array

, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18
```

Lists

Lists are a vector which can store any object type...

```
> y <- list(TRUE, 1, "a") # a heterogenous list
> y

[[1]]
[1] TRUE

[[2]]
[1] 1

[[3]]
[1] "a"
```

Lists (cont'd)

...including other lists.

```
> list(list(1, "Two"), "One more thing.") # recursive list
[[1]]
[[1]][[1]]
[1] 1

[[1]][[2]]
[1] "Two"


[[2]]
[1] "One more thing."
```

The data.frame object

The data.frame is

- probably the most important data structure in R
- a special kind of list that acts like a matrix with heterogenous columns

```
> df <- data.frame(x=c(1,2), y=c("a","b")) # a data.frame
> df # column `x` is numeric and column `y` is character

  x y
1 1 a
2 2 b
```

Many analysis functions expect data in a data.frame.

Subsetting using [, [[, and $

- `x[i]` and `x[i,j]`
  - select a subset of a vector, `matrix`, `array`, or `data.frame`
  - subset by an `integer`, `logical`, or `character` vectors
- `x[[i]]`
  - selects a single element from a vector
  - selects a single column of a `data.frame`
  - subset by an `integer` or a `character`
- `x$name`
  - selects an element by name from a `list` or `data.frame`
  - subset by a `character`

Subsetting matrices using [

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[1,c(1,2)]
[1] 1 4
> x[c(1,2),] # blank indices means assume all indices
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

Subsetting a data.frame using [, [[, and $

```
> df
  x y
1 1 a
2 2 b
> df[1,c("x","y")] # using integer and character indices
  x y
1 1 a
> df[[1]] # selecting a column by its index
[1] 1 2
> df$x # selecting a column by name
[1] 1 2
```

Functions

"To understand computations in R, two slogans are helpful:"

- Everything that exists is an object
- Everything that happens is a function call

-John Chambers

```
> foo <- function(x) {
+   x + 1 # functions return the value last evaluated
+ }
> foo(2)

[1] 3

> foo(4:9)

[1]  5  6  7  8  9 10
```

Functions (cont'd)

```
> do.fun <- function(x, f) {
+   f(x) # functions can be passed to other functions
+ }
> do.fun(2, foo)

[1] 3

> do.fun(2, function(x) x + 1) # anonymously

[1] 3

> (function(x) x + 1)(2) # an anonymous function

[1] 3

> 1 + 2 # when you do this...

[1] 3

> `+`(1, 2) # ...R does this

[1] 3
```

Example of data analysis

Example dataset:

- Prices and attributes of almost 54,000 diamonds
- Included with **ggplot2** package

```
> library(ggplot2)
> class(diamonds) # diamonds is a data.frame

[1] "data.frame"

> dim(diamonds) # 53940 observations and 10 columns

[1] 53940    10

> names(diamonds) # names of the columns

 [1] "carat"   "cut"     "color"   "clarity" "depth"
 [6] "table"   "price"   "x"       "y"       "z"
```

Example of data analysis

```
> head(diamonds) # first few rows

  carat       cut color clarity depth table price    x    y
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96
     z
1 2.43
2 2.31
3 2.31
4 2.63
5 2.75
6 2.48
```

Example of data analysis

```
> tail (diamonds) # last few rows

      carat       cut color clarity depth table price    x
53935  0.72   Premium     D     SI1  62.7    59  2757 5.69
53936  0.72     Ideal     D     SI1  60.8    57  2757 5.75
53937  0.72      Good     D     SI1  63.1    55  2757 5.69
53938  0.70 Very Good     D     SI1  62.8    60  2757 5.66
53939  0.86   Premium     H     SI2  61.0    58  2757 6.15
53940  0.75     Ideal     D     SI2  62.2    55  2757 5.83
         y    z
53935 5.73 3.58
53936 5.76 3.50
53937 5.75 3.61
53938 5.68 3.56
53939 6.12 3.74
53940 5.87 3.64
```

A local dataframe: better visualization

https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

```
> library(dplyr)
> tbl_df(diamonds) # better view
```

```
Source: local data frame [53,940 x 10]

   carat       cut  color clarity depth table price     x
   (dbl)    (fctr) (fctr)  (fctr) (dbl) (dbl) (int) (dbl)
1   0.23     Ideal      E     SI2  61.5    55   326  3.95
2   0.21   Premium      E     SI1  59.8    61   326  3.89
3   0.23      Good      E     VS1  56.9    65   327  4.05
4   0.29   Premium      I     VS2  62.4    58   334  4.20
5   0.31      Good      J     SI2  63.3    58   335  4.34
6   0.24 Very Good      J    VVS2  62.8    57   336  3.94
7   0.24 Very Good      I    VVS1  62.3    57   336  3.95
8   0.26 Very Good      H     SI1  61.9    55   337  4.07
9   0.22      Fair      E     VS2  65.1    61   337  3.87
10  0.23 Very Good      H     VS1  59.4    61   338  4.00
..   ...       ...    ...     ...   ...   ...   ...   ...
Variables not shown: y (dbl), z (dbl)
```

Understanding the properties of the data

Suppose we want to check the range of each quantitative variable.

```
> class(diamonds$carat) # quantitative
[1] "numeric"
> range(diamonds$carat)
[1] 0.20 5.01
> class(diamonds$cut) # categorical
[1] "ordered" "factor"
> class(diamonds$depth) # quantitative
[1] "numeric"
> range(diamonds$depth)
[1] 43 79
```

...etc. Must be an easier way.

Summarization

The `summary` function summarizes variables and objects.

```
> summary(diamonds$carat) # for a quantitative variable

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2000  0.4000  0.7000  0.7979  1.0400  5.0100

> summary(diamonds$cut) # for a categorical variable

     Fair      Good Very Good   Premium     Ideal
     1610      4906     12082     13791     21551

> summary(diamonds[1:3]) # summarize first 3 variables

     carat               cut        color
 Min.   :0.2000   Fair     : 1610   D: 6775
 1st Qu.:0.4000   Good     : 4906   E: 9797
 Median :0.7000   Very Good:12082   F: 9542
 Mean   :0.7979   Premium  :13791   G:11292
 3rd Qu.:1.0400   Ideal    :21551   H: 8304
 Max.   :5.0100                     I: 5422
                                    J: 2808
```

Summarization (cont'd)

How about a for loop?

```
> ranges <- list()
> for ( name in names(diamonds) )
+   if ( is.numeric(diamonds[[name]]) )
+     ranges[[name]] <- range(diamonds[[name]])
> ranges[1:4]

$carat
[1] 0.20 5.01

$depth
[1] 43 79

$table
[1] 43 95

$price
[1]   326 18823
```

Summarization (cont'd)

Can we put this into a function?

```
> check.ranges <- function(data) {
+   ranges <- list()
+   for ( name in names(data) )
+     if ( is.numeric(data[[name]]) )
+       ranges[[name]] <- range(data[[name]])
+   ranges
+ }
```

Can we make it more generic?

```
> map <- function(data, fun) {
+   out <- list()
+   for ( i in seq_along(data) )
+     out[[i]] <- fun(data[[i]])
+   out
+ }
```

Summarization (cont'd)

Functions for this already exist in R.

```
> quantitative <- sapply(diamonds, is.numeric)
> ranges <- lapply(diamonds[quantitative], range)
> ranges[1:4]

$carat
[1] 0.20 5.01

$depth
[1] 43 79

$table
[1] 43 95

$price
[1]   326 18823
```

`lapply` and friends

- Apply functions over vectors
- Similar to "map" in many other languages
- Family includes:
  - `lapply` applies functions over vectors (usually lists)
  - `sapply` performs `lapply` and simplifies results
  - `mapply` performs `lapply` over multiple vectors
  - `tapply` applies functions according to some condition
  - `apply` applies functions over margins of a matrix or array

Data summarization example

Summarize the price of diamonds for each quality of cut.

```
> tapply(diamonds$price, diamonds$cut, summary)

$Fair
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    337    2050    3282    4359    5206   18570

$Good
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    327    1145    3050    3929    5028   18790

$`Very Good`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    336     912    2648    3982    5373   18820

$Premium
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326    1046    3185    4584    6296   18820

$Ideal
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     878    1810    3458    4678   18810
```

Data visualization

Can we use visualization to find a variable that explains the price of diamonds?

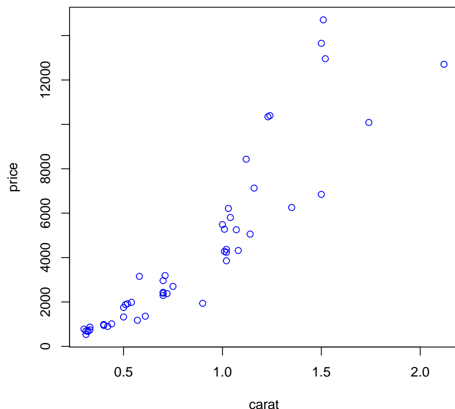
```
> set.seed(123)
> index <- sample(1:nrow(diamonds), 50) # try a subset first
> diamonds2 <- diamonds[index,]
```

First we try visualizing a small subset of the dataset.

Data visualization (cont'd)

Looks like larger diamonds cost more.

```
> plot(price ~ carat, data=diamonds2, col="blue")
```

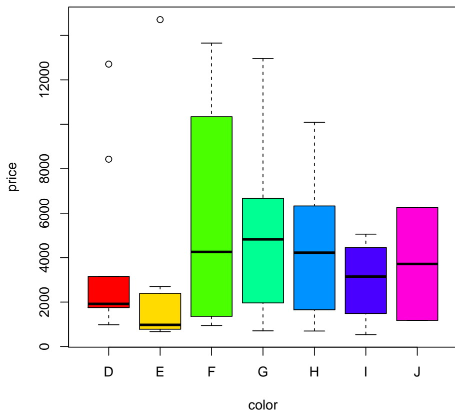


Use `pdf('fileName.pdf'); plot(...); dev.off()` to save as pdf

Data visualization (cont'd)

No apparent relationship between price and color.

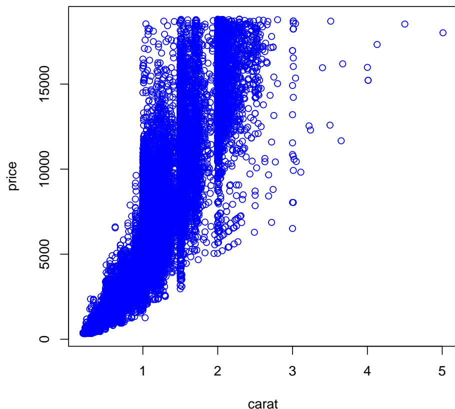
```
> plot(price ~ color, data=diamonds2, col=rainbow(7))
```



Data visualization (cont'd)

Try with the full dataset. Same relationship for carat.

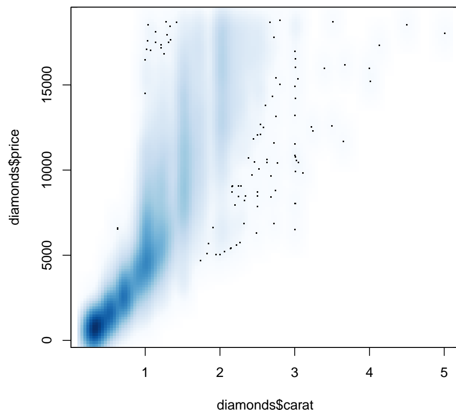
```
> plot(price ~ carat, data=diamonds, col="blue")
```



Data visualization (cont'd)

Use `smoothScatter` improve dense visualization.
Use `pairs` to simultaneously display all pairs of points

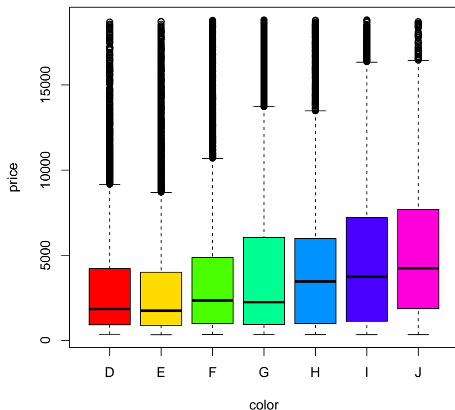
```
> smoothScatter(diamonds$carat, diamonds$price)
```



Data visualization (cont'd)

But worse colors cost more! (D best, J worst)
Not apparent when looking at subset.

```
> plot(price ~ color, data=diamonds, col=rainbow(7))
```



`lapply` and friends

- Apply functions over vectors
- Similar to "map" in many other languages
- Family includes:
  - `lapply` applies functions over vectors (usually lists)
  - `sapply` performs `lapply` and simplifies results
  - `mapply` performs `lapply` over multiple vectors
  - `tapply` applies functions according to some condition
  - `apply` applies functions over margins of a matrix or array

Data summarization example (cont'd)

Summarize the price of diamonds for different sizes (carats).

```
> sizes <- cut(diamonds$carat, breaks=quantile(diamonds$carat),
+              include.lowest=TRUE, labels=paste0("Q", 1:4))
> tapply(diamonds$price, sizes, summary)
```

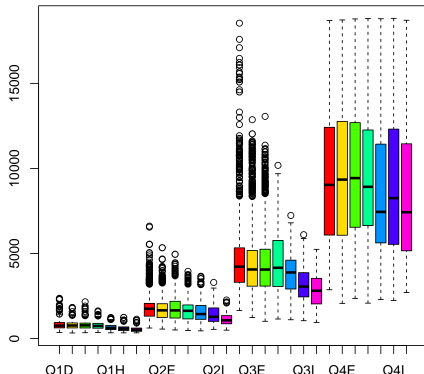
```
$Q1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  326.0   579.0   718.0   739.3   877.0  2366.0

$Q2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    452    1169    1626    1667    2016    6607

$Q3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    945    2936    3888    4189    4899   18540

$Q4
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2066    5818    8449    9274   12150   18820
```

Data visualization (cont'd)

Need to look at both carat and color together.

Use the 'sizes' variable we defined earlier as quantiles of carat.

```
> size.color <- as.factor(paste0(sizes, diamonds$color))
> plot(size.color, diamonds$price, col=rep(rainbow(7), 4))
```



Missing data example

Is there missing data in this dataset?

```
> NA == NA # careful!! cannot use `==` with NAs

[1] NA

> is.na(NA) # must use is.na

[1] TRUE

> any(c(TRUE, TRUE, FALSE)) # check if any elements are TRUE

[1] TRUE

> sapply(diamonds, function(x) any(is.na(x))) # no missing data

  carat     cut   color clarity   depth   table   price
  FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
      x       y       z
  FALSE   FALSE   FALSE
```

Missing data example (cont'd)

What if there were missing data we wanted to impute?

```
> set.seed(123)
> price2 <- diamonds$price
> index <- sample(seq_along(price2), 50)
> price2[index] <- NA
> any(is.na(price2)) # are there any missing values?

[1] TRUE

> sum(is.na(price2)) # how many missing values? (coercion!)

[1] 50

> price2[is.na(price2)] <- mean(price2, na.rm=TRUE) # impute
> any(is.na(price2))

[1] FALSE
```

The concept of tidy data

http://www.jstatsoft.org/v59/i10/paper

Example:
GDP per capita data (source wikipedia)
Country, year = variables
GDP = values
Every line is a country (=variable)
These are untidy data

Data input and output

Data input and output: define small datasets through command line

```
> df_economy <- tbl_df(data.frame(
+   Country=c('US','FR','CH'),
+   `2000`=c(36.4,22.4,37.8),
+   `2005`=c(44.3,34.9,54.8),
+   `2010`=c(48.7,40.7,74.3),check.names=FALSE))
> df_economy
```

```
Source: local data frame [3 x 4]

  Country  2000  2005  2010
   (fctr) (dbl) (dbl) (dbl)
1      US  36.4  44.3  48.7
2      FR  22.4  34.9  40.7
3      CH  37.8  54.8  74.3
```

The concept of tidy data

We would like:

Observations = lines
Variables (year, GDP) = columns

```
> library(tidyr)
> gather(df_economy,'Year','GDP',2:4)
```

```
Source: local data frame [9 x 3]

  Country   Year   GDP
   (fctr) (fctr) (dbl)
1      US   2000  36.4
2      FR   2000  22.4
3      CH   2000  37.8
4      US   2005  44.3
5      FR   2005  34.9
6      CH   2005  54.8
7      US   2010  48.7
8      FR   2010  40.7
9      CH   2010  74.3
```

The function is similar to `reshape`, but has an easier syntax.

Another example of untidy data

```
> df_pollution <- tbl_df(data.frame(
+   City=c(rep('NYC',2),rep('London',2),rep('Beijing',2)),
+   Size=rep(c('Large','Small'),3),
+   Amount=c(23,14,22,16,121,56)))
> df_pollution
```

```
Source: local data frame [6 x 3]

     City   Size Amount
   (fctr) (fctr)  (dbl)
1     NYC  Large     23
2     NYC  Small     14
3  London  Large     22
4  London  Small     16
5 Beijing  Large    121
6 Beijing  Small     56
```

Tidying the untidy data

Here the observational unit is a city, and we take two measurements for each city. Need one line per observational unit.

```
> spread(df_pollution,Size,Amount)

Source: local data frame [3 x 3]

     City Large Small
   (fctr) (dbl) (dbl)
1 Beijing   121    56
2  London    22    16
3     NYC    23    14
```

Read data from csv files.

```
> setwd('/Users/ovitek/Dropbox/Olga/Teaching/CS6220/Fall15/LectureNotes/1-intro/
> flights_in <-read.csv('SFO_landings_out.csv',header=TRUE)
> head(flights_in)

  Period      Airline Landing.Aircraft.Type
1 200507      ABX Air             Freighter
2 200507      ABX Air             Freighter
3 200507 ATA Airlines             Passenger
4 200507 ATA Airlines             Passenger
5 200507   Air Canada             Passenger
6 200507   Air Canada             Passenger
  Aircraft.Manufacturer Aircraft.Model Aircraft.Version
1     McDonnell Douglas           DC-9               30
2     McDonnell Douglas           DC-9               41
3                Boeing            757              200
4                Boeing            757              300
5                Boeing            767              333
6                Airbus           A319              114
  Landing.Count Total.Landed.Weight
1            40             4066000
2             1              102000
3             2              396000
4           167            37408000
5             1              320000
6           160            21520000
```

Read data from csv files.

```
> tbl_df(flights_in)

Source: local data frame [16,042 x 8]

   Period      Airline Landing.Aircraft.Type
    (int)       (fctr)                (fctr)
1  200507      ABX Air             Freighter
2  200507      ABX Air             Freighter
3  200507 ATA Airlines             Passenger
4  200507 ATA Airlines             Passenger
5  200507   Air Canada             Passenger
6  200507   Air Canada             Passenger
7  200507   Air Canada             Passenger
8  200507   Air Canada             Passenger
9  200507    Air China             Passenger
10 200507   Air France             Passenger
..    ...          ...                   ...
Variables not shown: Aircraft.Manufacturer (fctr),
  Aircraft.Model (fctr), Aircraft.Version (fctr),
  Landing.Count (int), Total.Landed.Weight (int)
```

Advanced data manipulation with `dplyr`.

- Extract existing columns (=variables): `select()`
- Extract existing lines (observations): `filter()`
- Create new columns: `mutate()`
- Reduce dimension of `data`: `summarize()`
- Order data: `arrange()`
- Write code better/faster: `%>%`
- Group observations `group_by()`

Select 3 columns

```
> library(dplyr)
> flights_df <- tbl_df(flights_in)
> select(flights_df, Period,Airline, Total.Landed.Weight)

Source: local data frame [16,042 x 3]

   Period      Airline Total.Landed.Weight
    (int)       (fctr)               (int)
1  200507      ABX Air             4066000
2  200507      ABX Air              102000
3  200507 ATA Airlines              396000
4  200507 ATA Airlines            37408000
5  200507   Air Canada              320000
6  200507   Air Canada            21520000
7  200507   Air Canada            20761200
8  200507   Air Canada              343040
9  200507    Air China            14490000
10 200507   Air France            18855500
..    ...          ...                 ...
```

Select all columsn except...

```
> select(flights_df, -Period, -Airline, -Total.Landed.Weight)

Source: local data frame [16,042 x 5]

   Landing.Aircraft.Type Aircraft.Manufacturer
                  (fctr)                (fctr)
1             Freighter     McDonnell Douglas
2             Freighter     McDonnell Douglas
3             Passenger                Boeing
4             Passenger                Boeing
5             Passenger                Boeing
6             Passenger                Airbus
7             Passenger                Airbus
8             Passenger                Airbus
9             Passenger                Boeing
10            Passenger                Boeing
..                  ...                   ...
Variables not shown: Aircraft.Model (fctr),
  Aircraft.Version (fctr), Landing.Count (int)
```

Select all columsn in range...

```
> select(flights_df, Aircraft.Model:Landing.Count)

Source: local data frame [16,042 x 3]

   Aircraft.Model Aircraft.Version Landing.Count
           (fctr)           (fctr)         (int)
1            DC-9               30            40
2            DC-9               41             1
3             757              200             2
4             757              300           167
5             767              333             1
6            A319              114           160
7            A320              211           146
8            A321              211             2
9             747              400            23
10            747              400            31
..            ...              ...           ...
```

Many other selection functions available (See cheatsheet/manual)

Extract observations

```
> filter(flights_df,Landing.Count > 2000) %>%
+   select(Airline, Landing.Count)
Source: local data frame [19 x 2]

           Airline Landing.Count
            (fctr)         (int)
1  SkyWest Airlines          2015
2  SkyWest Airlines          2067
3  SkyWest Airlines          2094
4  SkyWest Airlines          2001
5  SkyWest Airlines          2083
6  SkyWest Airlines          2017
7  SkyWest Airlines          2129
8  SkyWest Airlines          2192
9  SkyWest Airlines          2245
10 SkyWest Airlines          2071
11 SkyWest Airlines          2156
12 SkyWest Airlines          2018
13 SkyWest Airlines          2054
14 SkyWest Airlines          2033
15 SkyWest Airlines          2011
16 SkyWest Airlines          2154
17 SkyWest Airlines          2141
18 SkyWest Airlines          2028
19 SkyWest Airlines          2016
```

Create new variables

```
any_dplyr_function(data_frame, argument) is equivalent to
data_frame %>% any_dplyr_function(argument)
```

```
> mutate(flights_df,average.weight= Total.Landed.Weight/Landing.Count) %>%
+   select(Airline, average.weight)

Source: local data frame [16,042 x 2]

        Airline average.weight
         (fctr)          (dbl)
1       ABX Air       101650.0
2       ABX Air       102000.0
3  ATA Airlines       198000.0
4  ATA Airlines       224000.0
5    Air Canada       320000.0
6    Air Canada       134500.0
7    Air Canada       142200.0
8    Air Canada       171520.0
9     Air China       630000.0
10   Air France       608241.9
..          ...            ...
```

Summarize the content

```
> summarize(flights_df, sum(Landing.Count))

Source: local data frame [1 x 1]

  sum(Landing.Count)
               (int)
1            1812165

> summarize(flights_df, n_distinct(Aircraft.Manufacturer))

Source: local data frame [1 x 1]

  n_distinct(Aircraft.Manufacturer)
                              (int)
1                                15
```

Arrange

```
> arrange(flights_df,Total.Landed.Weight) %>%
+   select(Airline, Total.Landed.Weight)

Source: local data frame [16,042 x 2]

      Airline Total.Landed.Weight
       (fctr)               (int)
1  Ameriflight                7000
2  Ameriflight                7000
3  Ameriflight                7000
4  Ameriflight                7000
5  Ameriflight                7000
6  Ameriflight                7000
7  Ameriflight                7000
8  Ameriflight                7000
9  Ameriflight               10900
10 Ameriflight               10900
..         ...                 ...
```

Arrange in the opposite order

```
> arrange(flights_df,-Total.Landed.Weight) %>%
+   select(Airline, Total.Landed.Weight)

Source: local data frame [16,042 x 2]

                         Airline Total.Landed.Weight
                          (fctr)               (int)
1  United Airlines - Pre 07/01/2013           273042000
2  United Airlines - Pre 07/01/2013           269676000
3  United Airlines - Pre 07/01/2013           267300000
4  United Airlines - Pre 07/01/2013           264528000
5  United Airlines - Pre 07/01/2013           263736000
6  United Airlines - Pre 07/01/2013           261162000
7  United Airlines - Pre 07/01/2013           260370000
8  United Airlines - Pre 07/01/2013           260172000
9  United Airlines - Pre 07/01/2013           258192000
10 United Airlines - Pre 07/01/2013           257202000
..                              ...                 ...
```

Advanced data manipulation with dplyr.

```
> flights_df %>% group_by(Airline,Aircraft.Model) %>%
+   summarize(total.landings= sum(Landing.Count))

Source: local data frame [243 x 3]
Groups: Airline [?]

       Airline Aircraft.Model total.landings
        (fctr)         (fctr)          (int)
1      ABX Air            767           5743
2      ABX Air           DC-8              7
3      ABX Air           DC-9            864
4   Aer Lingus           A330            806
5   Aeromexico            737           3693
6   Aeromexico            777              2
7   Aeromexico            ERJ             37
8   Air Berlin           A330            174
9  Air Canada             767            270
10 Air Canada            A319          10535
..         ...            ...            ...
```

Save/write and load the data.

```
> save.image(file='myFileName.RData')
> save(flights_in, file='myFlightsFileName.RData')
> load('myFlightsFileName.RData')
```