% DS 201 (Lab 1) % Sidak Yntiso sgy210@nyu.edu, Office Hours: CDS 7th floor, 10-12 pm % February 3, 2020

Game plan

- This presentation, created using knitr, is available on the course website
- We will cover the basics (no familarity required): data structures, conditions + loops, data manipulation (dplyr, data.table)

R Resources

- The Art of R Programming - N. Matloff
- Modern Applied Statistics with S - W. Venables and B. Ripley
- Advanced R Programming - H. Wickham
- The R Inferno - P. Burns
- Reading R documentation: `?lm`
- CRAN documentation: AER
- Journal documentation: `plm`
- Resources for R Markdown: (https://bookdown.org/yihui/rmarkdown/)

Preliminaries

- You will need to install R and an environment e.g. Rstudio
- Before running file, install knitr: install.packages("knitr")

```
require(knitr)
knit(Lab1.Rmd)
```

Data Structures

```
# 3 types of data structures: vectors, arrays, and data frames
#Two types of vectors: atomic/lists
#Difference: atomic vectors are flat and will contain only elements of same type
rm(list = ls())
atom <- c(nums = 1:5,c(1),lets=letters[1:5])
str(atom)
```

```
##  Named chr [1:11] "1" "2" "3" "4" "5" "1" "a" "b" "c" "d" "e"
##  - attr(*, "names")= chr [1:11] "nums1" "nums2" "nums3" "nums4" ...
```

```
lst <- list(nums = 1:5,c(1),lets=letters[1:5])
str(lst)
```

```
## List of 3
##  $ nums: int [1:5] 1 2 3 4 5
##  $     : num 1
##  $ lets: chr [1:5] "a" "b" "c" "d" ...
```

```r
#three ways to index lists
lst$nums
```

```
## [1] 1 2 3 4 5
```

```r
lst[[1]]
```

```
## [1] 1 2 3 4 5
```

```r
lst["nums"]
```

```
## $nums
## [1] 1 2 3 4 5
```

```r
#combine vectors with c
c(lst,lst)
```

```
## $nums
## [1] 1 2 3 4 5
##
## [[2]]
## [1] 1
##
## $lets
## [1] "a" "b" "c" "d" "e"
##
## $nums
## [1] 1 2 3 4 5
##
## [[5]]
## [1] 1
##
## $lets
## [1] "a" "b" "c" "d" "e"
```

```r
#convert lst to atomic
str(unlist(lst))
```

```
##  Named chr [1:11] "1" "2" "3" "4" "5" "1" "a" "b" "c" "d" "e"
##  - attr(*, "names")= chr [1:11] "nums1" "nums2" "nums3" "nums4" ...
```

```r
##Arrays e.g. matrices
mat <- matrix(rnorm(6),nrow=6,ncol=1) #rnorm, runif,rpois, etc generate random variables
mat[,1]
```

```
## [1]  0.03044960 -0.47440558  0.60286533  0.88550721  0.02746754  2.59164258
```

```r
mat[1,]
```

```
## [1] 0.0304496
```

```r
str(mat)
```

```
##  num [1:6, 1] 0.0304 -0.4744 0.6029 0.8855 0.0275 ...
```

```r
ncol(mat)
```

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

```r
nrow(mat)
```

```
## [1] 6
```

```r
#add/remove columns
mat <- cbind(1,mat)
rbind(mat,mat)
```

```
##       [,1]        [,2]
##  [1,]    1  0.03044960
##  [2,]    1 -0.47440558
##  [3,]    1  0.60286533
##  [4,]    1  0.88550721
##  [5,]    1  0.02746754
##  [6,]    1  2.59164258
##  [7,]    1  0.03044960
##  [8,]    1 -0.47440558
##  [9,]    1  0.60286533
## [10,]    1  0.88550721
## [11,]    1  0.02746754
## [12,]    1  2.59164258
```

```r
#matrices are easy to manipulate
#transpose
t(mat)
```

```
##           [,1]       [,2]      [,3]      [,4]       [,5]     [,6]
## [1,] 1.0000000  1.0000000 1.0000000 1.0000000 1.00000000 1.000000
## [2,] 0.0304496 -0.4744056 0.6028653 0.8855072 0.02746754 2.591643
```

```r
#X'X
(t(mat) %*% mat)
```

```
##          [,1]     [,2]
## [1,] 6.000000 3.663527
## [2,] 3.663527 8.090923
```

```r
#(X'X)^-1
solve((t(mat) %*% mat))
```

```
##            [,1]       [,2]
## [1,]  0.2303524 -0.1043023
## [2,] -0.1043023  0.1708228
```

```r
#OLS
y <- 2 + 2*mat[,2] + rnorm(6)
solve(t(mat) %*% mat) %*% t(mat) %*% y
```

```
##          [,1]
## [1,] 1.467493
## [2,] 1.780102
```

```r
##Dataframes are matrices that allow mixing different types
dat <- data.frame(mat)
dat$lst <- lst
str(dat)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ X1 : num  1 1 1 1 1 1
##  $ X2 : num  0.0304 -0.4744 0.6029 0.8855 0.0275 ...
##  $ lst:List of 6
##   ..$ nums: int  1 2 3 4 5
```

```
##   ..$     : num 1
##   ..$ lets: chr  "a" "b" "c" "d" ...
##   ..$ nums: int  1 2 3 4 5
##   ..$     : num 1
##   ..$ lets: chr  "a" "b" "c" "d" ...
```

```
#rename columns or rows
colnames(dat) <- letters[1:ncol(dat)]
rownames(dat) <- c(1:nrow(dat))
```

Conditions and Loops

```
#loops are easy to write but can be slow. vectorize when possible
for (st in c(1:length(state.abb))){
  print(state.abb[st])
}
```

```
## [1] "AL"
## [1] "AK"
## [1] "AZ"
## [1] "AR"
## [1] "CA"
## [1] "CO"
## [1] "CT"
## [1] "DE"
## [1] "FL"
## [1] "GA"
## [1] "HI"
## [1] "ID"
## [1] "IL"
## [1] "IN"
## [1] "IA"
## [1] "KS"
## [1] "KY"
## [1] "LA"
## [1] "ME"
## [1] "MD"
## [1] "MA"
## [1] "MI"
## [1] "MN"
## [1] "MS"
## [1] "MO"
## [1] "MT"
## [1] "NE"
## [1] "NV"
## [1] "NH"
## [1] "NJ"
## [1] "NM"
## [1] "NY"
## [1] "NC"
## [1] "ND"
## [1] "OH"
## [1] "OK"
```

```
## [1] "OR"
## [1] "PA"
## [1] "RI"
## [1] "SC"
## [1] "SD"
## [1] "TN"
## [1] "TX"
## [1] "UT"
## [1] "VT"
## [1] "VA"
## [1] "WA"
## [1] "WV"
## [1] "WI"
## [1] "WY"
```

```
#functions
sum_US_area <- function(state.abb,state.area){
  US_area <- 0
  for (st in c(1:length(state.abb))){
    US_area <- US_area + state.area[st]
  }
  return(US_area)
}
sum_US_area(state.abb,state.area)
```

```
## [1] 3618399
```

```
#useful functions you will likely use
print(
  c("print", "cat", "paste", "with", "length", "sort", "order", "unique", "rep", "nrow", "ncol",
    "complete.cases", "subset", "merge", "mean", "sum", "sd", "var", "lag","lm", "model.matrix","coef",
    "residuals", "vcovHC (from sandwich)", "ivreg (from AER)", "summary", "pdf", "plot")
)
```

```
##  [1] "print"                  "cat"                    "paste"
##  [4] "with"                   "length"                 "sort"
##  [7] "order"                  "unique"                 "rep"
## [10] "nrow"                   "ncol"                   "complete.cases"
## [13] "subset"                 "merge"                  "mean"
## [16] "sum"                    "sd"                     "var"
## [19] "lag"                    "lm"                     "model.matrix"
## [22] "coef"                   "vcov"                   "residuals"
## [25] "vcovHC (from sandwich)" "ivreg (from AER)"       "summary"
## [28] "pdf"                    "plot"
```

```
#use apply, lapply, sapply for similar reasons
#apply (aptly named) applies some function to rows (1) or columns (2) of matrix
head(state.x77)
```

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
```

```r
apply(state.x77,2,mean)
```

```
## Population     Income Illiteracy   Life Exp     Murder    HS Grad      Frost
##  4246.4200  4435.8000     1.1700    70.8786     7.3780    53.1080   104.4600
##       Area
## 70735.8800
```

```r
#sapply applies some function to every element of a vector
sapply(mat, function(x) x+1)
```

```
##  [1] 2.0000000 2.0000000 2.0000000 2.0000000 2.0000000 2.0000000 1.0304496
##  [8] 0.5255944 1.6028653 1.8855072 1.0274675 3.5916426
```

```r
#lapply is the same as sapply except it returns list output (instead of vector output)
class(sapply(mat, function(x) x+1))
```

```
## [1] "numeric"
```

```r
class(lapply(mat, function(x) x+1))
```

```
## [1] "list"
```

```r
#if/else
if ("NY" %in% state.abb){
  print("NY is a state")
} else {
  print("NY is not a state")
}
```

```
## [1] "NY is a state"
```

Data Manipulation

```r
data(iris)
#to load datasets use haven::read_dta,read.table,data.table::fread, etc

#Many ways to handle data in R. base (above), dplyr and data.table (below)

#dplyr syntax
#install.packages("dplyr")
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
#summarize data by column
iris %>%
  summarise(av_sepal_len=mean(Sepal.Length))
```

```
##   av_sepal_len
## 1     5.843333
```

```r
#summarize data by column and group
iris %>%
  group_by(Species) %>%
  summarise(av_sepal_len=mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    av_sepal_len
##   <fct>             <dbl>
## 1 setosa             5.01
## 2 versicolor         5.94
## 3 virginica          6.59
```

```r
#classify data by column and group
iris %>%
  group_by(Species) %>%
  mutate(high_sep_len=if_else(Sepal.Length > mean(Sepal.Length),"High","Low"))
```

```
## # A tibble: 150 x 6
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species high_sep_len
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>   <chr>
##  1          5.1         3.5          1.4         0.2 setosa  High
##  2          4.9         3            1.4         0.2 setosa  Low
##  3          4.7         3.2          1.3         0.2 setosa  Low
##  4          4.6         3.1          1.5         0.2 setosa  Low
##  5          5           3.6          1.4         0.2 setosa  Low
##  6          5.4         3.9          1.7         0.4 setosa  High
##  7          4.6         3.4          1.4         0.3 setosa  Low
##  8          5           3.4          1.5         0.2 setosa  Low
##  9          4.4         2.9          1.4         0.2 setosa  Low
## 10          4.9         3.1          1.5         0.1 setosa  Low
## # ... with 140 more rows
```

```r
#creating new data
iris_average <- iris %>%
    group_by(Species) %>%
    summarise(av_sepal_len=mean(Sepal.Length))

#data.table synatx
#install.packages(data.table)
#data.table is faster with larger datasets
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
iris <- data.table(iris)

#subsetting data by selecting rows
head(iris[Species == 'virginica'])
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1:          6.3         3.3          6.0         2.5 virginica
## 2:          5.8         2.7          5.1         1.9 virginica
## 3:          7.1         3.0          5.9         2.1 virginica
## 4:          6.3         2.9          5.6         1.8 virginica
## 5:          6.5         3.0          5.8         2.2 virginica
## 6:          7.6         3.0          6.6         2.1 virginica
```

```r
#selecting columns. the dot symbol calls lists
iris[, .(Species)]
```

```
##        Species
##   1:    setosa
##   2:    setosa
##   3:    setosa
##   4:    setosa
##   5:    setosa
##  ---
## 146: virginica
## 147: virginica
## 148: virginica
## 149: virginica
## 150: virginica
```

```r
#selecting rows and columns. all columns except Petal.Length. with=F
head(iris[Species == 'virginica', -c("Petal.Length")])
```

```
##    Sepal.Length Sepal.Width Petal.Width   Species
## 1:          6.3         3.3         2.5 virginica
## 2:          5.8         2.7         1.9 virginica
## 3:          7.1         3.0         2.1 virginica
## 4:          6.3         2.9         1.8 virginica
## 5:          6.5         3.0         2.2 virginica
## 6:          7.6         3.0         2.1 virginica
```

```r
#computations
iris[Species == 'virginica', .( mean_petal_length = mean(Petal.Length)) ]
```

```
##    mean_petal_length
## 1:             5.552
```

```r
#computations by group
iris[, .( mean_petal_length = mean(Petal.Length),
                              sd_petal_length = sd(Petal.Length),
                              N = .N),
     by = .(Species)]
```

```
##       Species mean_petal_length sd_petal_length  N
## 1:     setosa             1.462       0.1736640 50
## 2: versicolor             4.260       0.4699110 50
## 3:  virginica             5.552       0.5518947 50
```