CSSS 569 Visualizing Data and Models
Lab 1: Supplemental R resource

Ramses Llobet\textsuperscript{1}

Department of Political Science, UW

January 7, 2022

\textsuperscript{1}Originally provided by former TA Brian Leung.
Useful R resources

▶ R

▶ R for Data Science (Grolemund and Wickham 2016)

▶ Quantitative Social Science: An Introduction (Imai 2017)

▶ DataCamp: https://www.datacamp.com

▶ R cheat sheets: https://rstudio.com/resources/cheatsheets/

▶ R Markdown

▶ R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)

▶ Data visualization

▶ Data Visualization: A Practical Introduction (Healy 2018)

▶ Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (Wilke 2019)

▶ Others

▶ Stack Overflow: https://stackoverflow.com

▶ TidyTuesday Project: https://github.com/rfordatascience/tidytuesday
Useful R resources

- R
  - R for Data Science (Grolemund and Wickham 2016)
Useful R resources

- R
  - *R for Data Science* (Grolemund and Wickham 2016)
  - *Quantitative Social Science: An Introduction* (Imai 2017)
Useful R resources

- R
  - *R for Data Science* (Grolemund and Wickham 2016)
  - *Quantitative Social Science: An Introduction* (Imai 2017)
  - DataCamp: [https://www.datacamp.com](https://www.datacamp.com)

- R cheat sheets: [https://rstudio.com/resources/cheatsheets/](https://rstudio.com/resources/cheatsheets/)

- R Markdown

- Data visualization
  - *Data Visualization: A Practical Introduction* (Healy 2018)
  - *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures* (Wilke 2019)

- Others
  - Stack Overflow: [https://stackoverflow.com](https://stackoverflow.com)
  - TidyTuesday Project: [https://github.com/rfordatascience/tidytuesday](https://github.com/rfordatascience/tidytuesday)
Useful R resources

▶ R
  ▶ R for Data Science (Grolemund and Wickham 2016)
  ▶ Quantitative Social Science : An Introduction (Imai 2017)
  ▶ DataCamp: https://www.datacamp.com
  ▶ R cheat sheets: https://rstudio.com/resources/cheatsheets/
  ▶ R Markdown
    ▶ R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)
  ▶ Data visualization
    ▶ Data Visualization: A Practical Introduction (Healy 2018)
    ▶ Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (Wilke 2019)
  ▶ Others
    ▶ Stack Overflow: https://stackoverflow.com
    ▶ TidyTuesday Project: https://github.com/rfordatascience/tidytuesday
Useful R resources

- R
  - R for Data Science (Grolemund and Wickham 2016)
  - Quantitative Social Science: An Introduction (Imai 2017)
  - DataCamp: https://www.datacamp.com
  - R cheat sheets: https://rstudio.com/resources/cheatsheets/

- R Markdown
  - R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)
  - Data visualization
    - Data Visualization: A Practical Introduction (Healy 2018)
    - Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (Wilke 2019)

- Others
  - Stack Overflow: https://stackoverflow.com
  - TidyTuesday Project: https://github.com/rfordatascience/tidytuesday
Useful R resources

- R
  - R for Data Science (Grolemund and Wickham 2016)
  - Quantitative Social Science: An Introduction (Imai 2017)
  - DataCamp: https://www.datacamp.com
  - R cheat sheets: https://rstudio.com/resources/cheatsheets/

- R Markdown
  - R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)
Useful R resources

- **R**
  - *R for Data Science* (Grolemund and Wickham 2016)
  - *Quantitative Social Science : An Introduction* (Imai 2017)
  - DataCamp: https://www.datacamp.com
  - R cheat sheets: https://rstudio.com/resources/cheatsheets/

- **R Markdown**

- Data visualization
Useful R resources

- R
  - *R for Data Science* (Grolemund and Wickham 2016)
  - *Quantitative Social Science: An Introduction* (Imai 2017)
  - DataCamp: https://www.datacamp.com
  - R cheat sheets: https://rstudio.com/resources/cheatsheets/

- R Markdown

- Data visualization
  - *Data Visualization: A Practical Introduction* (Healy 2018)
Useful R resources

▶ R
  ▶ R for Data Science (Grolemund and Wickham 2016)
  ▶ Quantitative Social Science: An Introduction (Imai 2017)
  ▶ DataCamp: https://www.datacamp.com
  ▶ R cheat sheets: https://rstudio.com/resources/cheatsheets/

▶ R Markdown
  ▶ R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)

▶ Data visualization
  ▶ Data Visualization: A Practical Introduction (Healy 2018)
  ▶ Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (Wilke 2019)
Useful R resources

▶ R
  ▶ *R for Data Science* (Grolemund and Wickham 2016)
  ▶ *Quantitative Social Science: An Introduction* (Imai 2017)
  ▶ DataCamp: https://www.datacamp.com
  ▶ R cheat sheets: https://rstudio.com/resources/cheatsheets/

▶ R Markdown

▶ Data visualization
  ▶ *Data Visualization: A Practical Introduction* (Healy 2018)
  ▶ *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures* (Wilke 2019)

▶ Others
Useful R resources

▶ R
  ▶ R for Data Science (Grolemund and Wickham 2016)
  ▶ Quantitative Social Science: An Introduction (Imai 2017)
  ▶ DataCamp: https://www.datacamp.com
  ▶ R cheat sheets: https://rstudio.com/resources/cheatsheets/

▶ R Markdown
  ▶ R Markdown: The Definitive Guide (Xie, Allaire, and Grolemund 2019)

▶ Data visualization
  ▶ Data Visualization: A Practical Introduction (Healy 2018)
  ▶ Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (Wilke 2019)

▶ Others
  ▶ Stack Overflow: https://stackoverflow.com
Useful R resources

- **R**
  - *R for Data Science* (Grolemund and Wickham 2016)
  - *Quantitative Social Science: An Introduction* (Imai 2017)
  - DataCamp: https://www.datacamp.com
  - R cheat sheets: https://rstudio.com/resources/cheatsheets/

- **R Markdown**

- **Data visualization**
  - *Data Visualization: A Practical Introduction* (Healy 2018)
  - *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures* (Wilke 2019)

- **Others**
  - Stack Overflow: https://stackoverflow.com
  - TidyTuesday Project: https://github.com/rfordatascience/tidytuesday
R boot camp

- R is a language and environment for statistical computing and graphics
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
  - System-supplied or user-defined functionality as *functions*
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
  - System-supplied or user-defined functionality as *functions*
  - Extended via *packages*
R boot camp

R is a language and environment for statistical computing and graphics

- *Object-oriented* style of programming
- System-supplied or user-defined functionality as *functions*
- Extended via *packages*

RStudio is an integrated development environment for R, which includes:

- A console to run R code
- An editor to write code and text
- Tools for plotting, history, debugging and workspace management
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
  - System-supplied or user-defined functionality as *functions*
  - Extended via *packages*
- RStudio is an integrated development environment for R, which includes:
  - a console to run R code
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
  - System-supplied or user-defined functionality as *functions*
  - Extended via *packages*

- RStudio is an integrated development environment for R, which includes:
  - A console to run R code
  - An editor to write code and text
R boot camp

- R is a language and environment for statistical computing and graphics
  - *Object-oriented* style of programming
  - System-supplied or user-defined functionality as *functions*
  - Extended via *packages*

- RStudio is an integrated development environment for R, which includes:
  - A console to run R code
  - An editor to write code and text
  - Tools for plotting, history, debugging and workspace management

Let's open RStudio and a plain R Script.
R boot camp

- R is a language and environment for statistical computing and graphics
  - Object-oriented style of programming
  - System-supplied or user-defined functionality as functions
  - Extended via packages
- RStudio is an integrated development environment for R, which includes:
  - a console to run R code
  - an editor to write code and text
  - tools for plotting, history, debugging and workspace management
- Let's open RStudio and a plain R Script
Running R code and operators

# Arithmetic Operators
1 + 1

## [1] 2

2 * 8

## [1] 16

9 / 3

## [1] 3

2^3

## [1] 8
Running R code and operators

# Relational Operators

10 > 8

## [1] TRUE

7 <= 6

## [1] FALSE

(2 * 5) == 10

## [1] TRUE

1 != 2

## [1] TRUE
Objects in R: vectors and assignment

```r
# Concatenate vectors into a new vector
c(1, 2, 3)

## [1] 1 2 3

# Assign them to a new object for manipulation
x <- c(1, 2, 3)
print(x) # or simply, x

## [1] 1 2 3

# Operators on vector
x + 1

## [1] 2 3 4

x == 1

## [1] TRUE FALSE FALSE
```

## [1]  TRUE FALSE FALSE
# Use an object as input to a function

```r
x <- c(1, 2, 3)

class(x)
```

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

```r
length(x)
```

```r
## [1] 3
```

```r
mean(x)
```

```r
## [1] 2
```
Objects in R: three beginner tips

1. Unless you assign (\texttt{<-}) some operations or transformations to an object, those chances will not be registered

```r
x <- c(1, 2, 3)
print(x + 1)
## [1] 2 3 4

print(x)
## [1] 1 2 3

x <- x + 1
print(x)
## [1] 2 3 4
```
2. New assignment will overwrite the original values if you assign some values to an existing object. It is a **major** source of errors. One advise is to keep distinct object names

```r
x <- c(1, 2, 3)
length(x)
## [1] 3

x <- c(1, 2, 3, 4, 5)
length(x)
## [1] 5
```
Objects in R: three beginner tips

3. When using functions, we often bump into unexpected outputs, or error messages:

```r
y <- c(1, 2, 3, NA)
mean(y)
```

```r
## [1] NA
```

```
# It's essential to know how to seek help:
help(mean)
```

```
# starting httpd help server ... done
```

```
?mean
```

```
# Specify appropriate arguments for functions:
mean(y, na.rm = TRUE)
```

```
## [1] 2
```
Objects in R: atomic vectors

- What are vectors exactly?
Objects in R: atomic vectors

- What are vectors exactly?
  - (Atomic) vectors are the most basic units of data in R
Objects in R: atomic vectors

- What are vectors exactly?
  - (Atomic) vectors are the most basic units of data in R
  - Most common types of atomic vectors: `numeric` (integer, double), logical, character
Objects in R: atomic vectors

- Most common types of atomic vectors: numeric (integer, double), logical, character

```r
x <- c(1, 2, 3)
class(x)

## [1] "numeric"

y <- c(TRUE, FALSE, FALSE)
class(y)

## [1] "logical"

names <- c("Peter", "Paul", "Mary")
class(names)

## [1] "character"
```
Objects in R: atomic vectors

You can also coerce one type of vector into another:

```r
x <- c(1, 2, 3)
x <- as.character(x)

print(x)
```

```r
# [1] "1" "2" "3"
```

```r
class(x)
```

```r
# [1] "character"
```
To deal with massive data, we need efficient data structures to store and manipulate vectors: matrices and data frames.
Objects in R: matrix and data frame

To create a matrix:

```r
# Create a vector
numbers <- 1:12
print(numbers)

## [1]  1  2  3  4  5  6  7  8  9 10 11 12

# Store it as a matrix
matrix1 <- matrix(data = numbers, nrow = 3, byrow = TRUE)
print(matrix1)

## [1,]  1  2  3  4
## [2,]  5  6  7  8
## [3,]  9 10 11 12
```
# Basic information

```r
class(matrix1)
```

## [1] "matrix" "array"

```r
dim(matrix1) # dimensions
```

## [1] 3 4
We can change the row/column names of matrices

rownames(matrix1)

## NULL

rownames(matrix1) <- c("row1", "row2", "row3")
print(matrix1)

## row1 1 2 3 4
## row2 5 6 7 8
## row3 9 10 11 12
Objects in R: matrix and data frame

# Automate any repetitive process
col_names <- paste0("column", 1:4)
print(col_names)

## [1] "column1" "column2" "column3" "column4"

colnames(matrix1) <- col_names
print(matrix1)

##      column1 column2 column3 column4
## row1      1      2      3      4
## row2      5      6      7      8
## row3      9     10     11     12
Objects in R: matrix and data frame

```r
# To augment the matrix with new column
column5 <- c(13, 14, 15)
matrix1 <- cbind(matrix1, column5)
print(matrix1)
```

```r
table(matrix1)
```

```
## column1 column2 column3 column4 column5
## row1    1     2     3     4    13
## row2    5     6     7     8    14
## row3    9     10    11    12    15
```
Objects in R: matrix and data frame

# To augment the matrix with new row
row4 <- c("a", "b", "c", "d", "e")
matrix1 <- rbind(matrix1, row4)
print(matrix1)

## column1 column2 column3 column4 column5
## row1 "1" "2" "3" "4" "13"
## row2 "5" "6" "7" "8" "14"
## row3 "9" "10" "11" "12" "15"
## row4 "a" "b" "c" "d" "e"

Why do all vectors become characters?
Objects in R: matrix and data frame

- Matrices vs. data frames

Matrices can only contain one homogenous type of vectors.
Data frames can contain heterogeneous types of vectors, and thus are more flexible.
Objects in R: matrix and data frame

- Matrices vs. data frames
  - Matrices can only contain one **homogenous** type of vectors
Matrices vs. data frames

- Matrices can only contain one **homogenous** type of vectors.
- Data frames can contain **heterogeneous** types of vectors, and thus are more flexible.
Data frames can contain heterogeneous types of vectors, and thus are more flexible.

```r
df1 <- data.frame(
  names = c("Peter", "Paul", "Mary"),
  age = c(14, 15, 16),
  female = c(FALSE, FALSE, TRUE),
  stringsAsFactors = FALSE
)

print(df1)
```

```
##     names age female
## 1  Peter  14    FALSE
## 2  Paul  15    FALSE
## 3  Mary  16     TRUE
```
Objects in R: matrix and data frame

# Basic information
class(df1)

## [1] "data.frame"

dim(df1)

## [1] 3 3

str(df1)

## 'data.frame': 3 obs. of 3 variables:
## $ names : chr  "Peter" "Paul" "Mary"
## $ age   : num 14 15 16
## $ female: logi FALSE FALSE TRUE
Objects in R: subsetting data

- There are several ways to subset data: row/column indices, variable names, or evaluations

```r
# 1) Subsetting by row/column indices
# For the element in row 1, column 1
df1[1, 1]

## [1] "Peter"

## [1] "Peter" "Paul" "Mary"
```
# 2) Subsetting by variable names

df1$names

## [1] "Peter" "Paul" "Mary"

df1$age

## [1] 14 15 16

df1$female

## [1] FALSE FALSE TRUE
# 3) Subsetting by evaluations

df1[df1$age >= 15, ]

## names age female
## 2 Paul 15 FALSE
## 3 Mary 16 TRUE

df1[df1$female == TRUE, ]

## names age female
## 3 Mary 16 TRUE

df1[df1$name %in% c("Peter", "Paul"), ]

## names age female
## 1 Peter 14 FALSE
## 2 Paul 15 FALSE
Objects in R: creating new variable in data frame

```r
print(df1)

## names  age  female
## 1 Peter 14  FALSE
## 2  Paul 15  FALSE
## 3  Mary 16   TRUE

df1$edu

## NULL

df1$edu <- c("hs", "col", "phd")

print(df1)

## names  age  female  edu
## 1 Peter 14  FALSE  hs
## 2  Paul 15  FALSE  col
## 3  Mary 16   TRUE  phd
```
### Summary of data structures in R

<table>
<thead>
<tr>
<th></th>
<th>Homogeneous</th>
<th>Heterogeneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d</td>
<td>Atomic vector</td>
<td>List</td>
</tr>
<tr>
<td>2d</td>
<td>Matrix</td>
<td>Data frame</td>
</tr>
<tr>
<td>nd</td>
<td>Array</td>
<td></td>
</tr>
</tbody>
</table>

- Another important data structure: `factor` for categorical data, which will be important for visualization purpose
Vector practices

- Create the following objects:

  1. vector1: \{a1, a2, a3, b1, b2, b3, c1, c2, c3 ... z1, z2, z3\}
    
    Hint: break down the question into two parts; check out function `rep(..., times = ..., each = ...)`

  2. vector2: The sequence from 1 to 49 by an increment of 2
    
    Hint: check out function `seq(...)`

  - Subset the 3rd, 16th, and 25th elements of the vector
  - Subset those elements whose values are either smaller than 10, or greater than 40
Vector practices

- Create the following objects:

1. vector1: \{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 \ldots z_1, z_2, z_3\}
Vector practices

Create the following objects:

1. vector1: \{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 \ldots z_1, z_2, z_3\}
   
   Hint: break downs the question into two parts; check out function `rep(..., times = ..., each = ...)`. 
Vector practices

- Create the following objects:

1. vector1: \{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 \ldots z_1, z_2, z_3\}
   - Hint: break down the question into two parts; check out function `rep(..., times = ..., each = ...)`

2. vector2: The sequence from 1 to 49 by an increment of 2
Vector practices

Create the following objects:

1. vector1: \{a1, a2, a3, b1, b2, b3, c1, c2, c3 \ldots z1, z2, z3\}
   - Hint: break downs the question into two parts; check out function rep(..., times = ..., each = ...)

2. vector2: The sequence from 1 to 49 by an increment of 2
   - Hint: check out function seq(...)
Vector practices

1. Create the following objects:
   1. `vector1`: \{a1, a2, a3, b1, b2, b3, c1, c2, c3 ... z1, z2, z3\}
      - Hint: break down the question into two parts, check out function `rep(..., times = ..., each = ...)`
   2. `vector2`: The sequence from 1 to 49 by an increment of 2
      - Hint: check out function `seq(...)`
      - Subset the 3rd, 16th, and 25th elements of the vector

2. Subset those elements whose values are either smaller than 10, or greater than 40
Create the following objects:

1. vector1: \{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 \ldots z_1, z_2, z_3\}
   - Hint: break down the question into two parts; check out function \texttt{rep(..., times = ..., each = ...)}

2. vector2: The sequence from 1 to 49 by an increment of 2
   - Hint: check out function \texttt{seq(...)}
   - Subset the 3rd, 16th, and 25th elements of the vector
   - Subset those elements whose values are either smaller than 10, or greater than 40
Vector practices

# Q1
chr <- rep(letters, each = 3)
print(chr)

## [1] "a" "a" "a" "b" "b" "b" "c" "c" "c" "d" "d"
## [12] "d" "e" "e" "e" "f" "f" "f" "g" "g" "g" "h"
## [23] "h" "h" "i" "i" "i" "j" "j" "j" "k" "k" "k"
## [34] "l" "l" "l" "m" "m" "m" "n" "n" "n" "o" "o"
## [45] "o" "p" "p" "p" "q" "q" "q" "r" "r" "r" "s"
## [56] "s" "s" "t" "t" "t" "u" "u" "u" "v" "v" "v"
## [67] "w" "w" "w" "x" "x" "x" "y" "y" "y" "z" "z"
## [78] "z"

num <- rep(1:3, times = length(letters))
print(num)

## [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
## [24] 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
## [47] 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
## [70] 1 2 3 1 2 3 1 2 3
# Q1
vector1 <- paste0(chr, num)
print(vector1)

## [1]  "a1"  "a2"  "a3"  "b1"  "b2"  "b3"  "c1"  "c2"  "c3"
## [10] "d1"  "d2"  "d3"  "e1"  "e2"  "e3"  "f1"  "f2"  "f3"
## [19] "g1"  "g2"  "g3"  "h1"  "h2"  "h3"  "i1"  "i2"  "i3"
## [28] "j1"  "j2"  "j3"  "k1"  "k2"  "k3"  "l1"  "l2"  "l3"
## [37] "m1"  "m2"  "m3"  "n1"  "n2"  "n3"  "o1"  "o2"  "o3"
## [46] "p1"  "p2"  "p3"  "q1"  "q2"  "q3"  "r1"  "r2"  "r3"
## [55] "s1"  "s2"  "s3"  "t1"  "t2"  "t3"  "u1"  "u2"  "u3"
## [64] "v1"  "v2"  "v3"  "w1"  "w2"  "w3"  "x1"  "x2"  "x3"
## [73] "y1"  "y2"  "y3"  "z1"  "z2"  "z3"
Vector practices

# Q2

```r
to <- seq(from = 1, to = 49, by = 2)
print(vector2)
```

```r
## [1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
## [16] 31 33 35 37 39 41 43 45 47 49
```

```r
vector2[c(3, 16, 25)]
```

```r
## [1] 5 31 49
```

```r
vector2[vector2 < 10 | vector2 > 40]
```

```r
## [1] 1 3 5 7 9 41 43 45 47 49
```
3. matrix1: a 5 by 5 matrix containing values from vector2
3. matrix1: a 5 by 5 matrix containing values from vector2
   ▶ Assign the row names: row_a, row_b, row_c, row_d, row_e
Vector practices

3. matrix1: a 5 by 5 matrix containing values from vector2
   ▶ Assign the row names: row_a, row_b, row_c, row_d, row_e
   ▶ Assign the column names: col1, col2, col3, col4, col5
3. **matrix1**: a 5 by 5 matrix containing values from vector2
   - Assign the row names: `row_a`, `row_b`, `row_c`, `row_d`, `row_e`
   - Assign the column names: `col1`, `col2`, `col3`, `col4`, `col5`
   - Multiply the values in the first column of matrix 1 by 100; overwrite the original column
Vector practices

3. matrix1: a 5 by 5 matrix containing values from vector2
   ▶ Assign the row names: row_a, row_b, row_c, row_d, row_e
   ▶ Assign the column names: col1, col2, col3, col4, col5
   ▶ Multiply the values in the first column of matrix 1 by 100; overwrite the original column

4. df1: a dataframe with two variables:
Vector practices

3. matrix1: a 5 by 5 matrix containing values from vector2
   - Assign the row names: row_a, row_b, row_c, row_d, row_e
   - Assign the column names: col1, col2, col3, col4, col5
   - Multiply the values in the first column of matrix 1 by 100; overwrite the original column

4. df1: a dataframe with two variables:
   - country = {US, UK, CA, FR, IT}
3. matrix1: a 5 by 5 matrix containing values from vector2
   - Assign the row names: row_a, row_b, row_c, row_d, row_e
   - Assign the column names: col1, col2, col3, col4, col5
   - Multiply the values in the first column of matrix 1 by 100; overwrite the original column

4. df1: a dataframe with two variables:
   - country = \{US, UK, CA, FR, IT\}
   - pop = \{327, 66, 37, 67, 60\}
Vector practices

3. matrix1: a 5 by 5 matrix containing values from vector2
   - Assign the row names: row_a, row_b, row_c, row_d, row_e
   - Assign the column names: col1, col2, col3, col4, col5
   - Multiply the values in the first column of matrix 1 by 100; overwrite the original column

4. df1: a dataframe with two variables:
   - country = {US, UK, CA, FR, IT}
   - pop = {327, 66, 37, 67, 60}
   - Subset top-three observations in term of the level of population
3. **matrix1**: a 5 by 5 matrix containing values from `vector2`
   - Assign the row names: `row_a`, `row_b`, `row_c`, `row_d`, `row_e`
   - Assign the column names: `col1`, `col2`, `col3`, `col4`, `col5`
   - Multiply the values in the first column of matrix 1 by 100; overwrite the original column

4. **df1**: a dataframe with two variables:
   - `country = {US, UK, CA, FR, IT}`
   - `pop = {327, 66, 37, 67, 60}`
   - Subset top-three observations in term of the level of population
   - Hint: check out function `order(...)"
Vector practices

# Q3
matrix1 <- matrix(data = vector2, nrow = 5, ncol = 5)
rownames(matrix1) <- paste("row", letters[1:5], sep = "_")
colnames(matrix1) <- paste0("col", 1:5)
matrix1[, 1] <- matrix1[, 1] * 100
print(matrix1)

##
## col1  col2  col3  col4  col5
## row_a 100   11   21   31   41
## row_b 300   13   23   33   43
## row_c 500   15   25   35   45
## row_d 700   17   27   37   47
## row_e 900   19   29   39   49
Vector practices

# Q4
```r
df1 <- data.frame(country = c("US", "UK", "CA", "FR", "IT"),
                   pop = c(327, 66, 37, 67, 60))
print(df1)
```

```r
## country pop
## 1 US 327
## 2 UK 66
## 3 CA 37
## 4 FR 67
## 5 IT 60
```

```r
order(df1$pop, decreasing = TRUE)
```

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

```r
top3 <- order(df1$pop, decreasing = TRUE)[1:3]
df1[top3, ]
```

```r
## country pop
## 1 US 327
## 4 FR 67
## 2 UK 66
```
Workflow in R

- Usual workflow for data analysis (Grolemund and Wickham 2016):
Tidyverse and tidy data

- Tidyverse is a collection of packages designed for data science with unified grammar and data structures.
Tidyverse and tidy data

- Tidyverse is a collection of packages designed for data science with unified grammar and data structures
- *Tidy data:*
  - Each variable must have its own column
  - Each observation must have its own row
  - Each value must have its own cell
Tidyverse and tidy data

- Tidyverse is a collection of packages designed for data science with unified grammar and data structures

- *Tidy data:*
  - Each **variable** must have its own **column**
Tidyverse and tidy data

- Tidyverse is a collection of packages designed for data science with unified grammar and data structures

- *Tidy data:*
  - Each **variable** must have its own **column**
  - Each **observation** must have its own **row**
Tidyverse and tidy data

- Tidyverse is a collection of packages designed for data science with unified grammar and data structures

- **Tidy data:**
  - Each **variable** must have its own **column**
  - Each **observation** must have its own **row**
  - Each value must have its own cell
Tidyverse and tidy data

- To install Tidyverse package, run:

```
install.packages("tidyverse")
```

- To load a package, run (usually at the top of your R document):

```
library(tidyverse)
```
Importing data in R

```r
# Load package
library(tidyverse)

# Load econ.csv
econ <- read_csv("https://students.washington.edu/rllobet/vis/lab1/data/econ.csv")

## Rows: 557 Columns: 4

## -- Column specification -------------------------------------
## Delimiter: ","
## chr (1): country
## dbl (3): GWn, year, gdpPerCap

# tibble (tbl) is a special class of data frame
class(econ)

## [1] "spec_tbl_df" "tbl_df" "tbl"
```
Importing data in R

```r
# Get a sense of the dataset
glimpse(econ)
```

```r
## Rows: 557
## Columns: 4
## $ country  <chr>  "Afghanistan",  "Afghanistan",  ~
## $ GWn       <dbl>  700,  700,  700,  339,  615,  615,  ~
## $ gdpPercap <dbl>  862.5477,  818.9504,  600.5932,  ~
```

```r
head(econ)
```

```r
## # A tibble: 6 x 4
## country    GWn year gdpPercap
## <chr>   <dbl> <dbl>  <dbl>
## 1 Afghanistan  700 1983   863.
## 2 Afghanistan  700 1985   819.
## 3 Afghanistan  700 1991   601.
## 4 Albania      339 2000  2962.
## 5 Algeria       615 1967  1824.
## 6 Algeria       615 1968  1977.
```
Basic data wrangling

- Below are just scratching the surface; check out...
Basic data wrangling

- Below are just scratching the surface; check out
  - Introductory course to tidyverse at DataCamp
Basic data wrangling

- Below are just scratching the surface; check out
  - Introductory course to tidyverse at DataCamp
  - Cheat sheet for data wrangling
Basic data wrangling

- Below are just scratching the surface; check out
  - Introductory course to tidyverse at DataCamp
  - Cheat sheet for data wrangling
  - *R for Data Science*
Basic data wrangling: `count()`

Count number of rows in each group:

```r
econ %>%
  count(country)
```

## A tibble: 146 x 2
##
country     n
<chr>       <int>
## 1 Afghanistan 3
## 2 Albania    1
## 3 Algeria    5
## 4 Angola     3
## 5 Argentina  5
## 6 Australia  3
## 7 Austria    9
## 8 Bahrain    2
## 9 Bangladesh 5
## 10 Belarus (Byelorussia) 1
## # ... with 136 more rows
Basic data wrangling: %>%

▶ What is %>% (“pipe”)?
▶ x %>% fun(y) is equivalent to fun(x, y)
▶ Its advantage will be apparent when you perform numerous steps of manipulation

```r
count(econ, country) # Equivalent to econ %>% count(country)
```

```
## # A tibble: 146 x 2
##    country       n
##  <chr>    <int>
##1  Afghanistan     3
##2    Albania       1
##3    Algeria       5
##4   Angola        3
##5  Argentina      5
##6   Australia     3
##7    Austria      9
##8   Bahrain       2
##9  Bangladesh     5
##10  Belarus (Byelorussia) 1
## # ... with 136 more rows
```
Basic data wrangling: `arrange()`

Order rows by values of column(s) from low to high:

```r
econ %>%
  count(country) %>%
  arrange(n)  # Rather than: arrange(count(econ, country), n)
```

```r
## # A tibble: 146 x 2
##    country                        n
##     <chr>             <int>
##  1  Albania           1
##  2  Belarus (Byelorussia) 1
##  3     Cambodia (Kampuchea) 1
##  4 Central African Republic 1
##  5      Chile           1
##  6      China           1
##  7     Dominican Republic 1
##  8      Estonia          1
##  9       Gabon           1
## 10     Ghana            1
## # ... with 136 more rows
```
Basic data wrangling: `arrange()`

Order rows by values of column(s) from high to low:

```r
econ %>%
  count(country) %>%
  arrange(desc(n))
```

## # A tibble: 146 x 2
## #  country          n
## #  <chr>            <int>
## #  United States of America  112
## #  Mexico              10
## #  Austria             9
## #  Uruguay             9
## #  Philippines         8
## #  Denmark             7
## #  Norway              7
## #  Portugal            7
## #  Trinidad and Tobago  7
## #  Venezuela           7
## #  ... with 136 more rows
Basic data wrangling: \texttt{filter()}

Extract rows that meet logical criteria:

\begin{verbatim}
  econ %>%
    filter(country == "Brazil")
\end{verbatim}

\begin{verbatim}
## # A tibble: 3 x 4
## country     GWN  year  gdpPerCap
##<chr> <dbl> <dbl>    <dbl>
## 1 Brazil 140 1954  1848.
## 2 Brazil 140 1989  5224.
## 3 Brazil 140 2002  5481.
\end{verbatim}
Basic data wrangling: `filter()`

Extract rows that meet multiple logical criteria:

```r
econ %>%
  filter(
    country == "Brazil" | country == "Russia (Soviet Union)" |
    country == "India" | country == "China"
  )
```

## # A tibble: 9 x 4
## #  country   GWn year gdpPercap
## 1 Brazil 140 1954 1848.
## 2 Brazil 140 1989 5224.
## 3 Brazil 140 2002 5481.
## 4 China 710 1996 2892.
## 5 India 750 1943 698.
## 6 India 750 1961 758.
## 7 India 750 1992 1350.
## 8 Russia (Soviet Union) 365 1982 6536.
## 9 Russia (Soviet Union) 365 2005 7269.
Basic data wrangling: filter()

Alternatively:

```r
econ %>%
  filter(country %in% c("Brazil", "Russia (Soviet Union)", "India", "China"))
```

```r
## # A tibble: 9 x 4
## #  country    GWn  year  gdpPerCap
## <chr>  <dbl> <dbl>    <dbl>
## 1 Brazil    140 1954    1848.
## 2 Brazil    140 1989    5224.
## 3 Brazil    140 2002    5481.
## 4 China     710 1996    2892.
## 5 India     750 1943    698.
## 6 India     750 1961    758.
## 7 India     750 1992    1350.
## 8 Russia (Soviet Union) 365 1982    6536.
## 9 Russia (Soviet Union) 365 2005    7269.
```
Basic data wrangling: `select()`

Extract columns (variables):

```r
econ %>%
  select(country, year, gdpPercap)
```

```
## # A tibble: 557 x 3
## #  country     year gdpPercap
## # <chr>  <dbl>    <dbl>
## 1 Afghanistan 1983  863.
## 2 Afghanistan 1985  819.
## 3 Afghanistan 1991  601.
## 4 Albania     2000  2962.
## 5 Algeria     1967  1824.
## 6 Algeria     1968  1977.
## 7 Algeria     1977  2759.
## 8 Algeria     1986  3301.
## 9 Algeria     2006  3386.
## 10 Angola     1953  1126.
## # ... with 547 more rows
```
Basic data wrangling: `filter()` & `select()`

Filter USA observations from 2000 to 2010 with `year` and `gdpPercap` as the only variables:

```
USAdata <- econ %>%
  filter(country == "United States of America",
          year %in% 2000:2010) %>%
  select(year, gdpPercap)
print(USAdata)
```

```
## # A tibble: 11 x 2
##   year  gdpPercap
##   <dbl>    <dbl>
## 1 2000  28702.
## 2 2001  28726.
## 3 2002  28977.
## 4 2003  29459.
## 5 2004  30200.
## 6 2005  30842.
## 7 2006  31358.
## 8 2007  31655.
## 9 2008  31251.
##10 2009  29899.
##11 2010  30491.
```
Basic data wrangling: `summarize()`

Compute table of summaries:

```r
USAdataset %>%
  summarize(avg_gdpPercap = mean(gdpPercap))
```

```r
## # A tibble: 1 x 1
## avg_gdpPercap
## <dbl>
## 1 30142.
```

What if we want to calculate the average GDP per capita for all countries in our data set?
Basic data wrangling: `group_by()` & `summarize()`

- Create a grouped version of the table with `group_by()`
- Subsequent functions will manipulate each group *separately*

```r
econ %>%
  group_by(country) %>%
  summarize(avg_gdpPercap = mean(gdpPercap)) %>%
  arrange(desc(avg_gdpPercap))
```

```r
## # A tibble: 146 x 2
##   country avg_gdpPercap
##    <chr>       <dbl>
## 1 Qatar       39157.
## 2 Kuwait      16288.
## 3 German Federal Republic 15739.
## 4 Norway      14846.
## 5 Ireland     14353.
## 6 Belarus (Byelorussia) 13659.
## 7 United States of America 13623.
## 8 United Arab Emirates 12812.
## 9 Belgium     12053.
## 10 Austria    11794.
## # ... with 136 more rows
```
Basic data wrangling: more summarize()

What if we want to know the numbers of distinct countries and years in the data set?

```r
econ %>%
  summarize_at(c("country", "year"), n_distinct)
```

```
# A tibble: 1 x 2
##  country  year
##     <int> <int>
## 1     146   111
```
Basic data wrangling: `mutate()`

Compute new columns (variables):

```r
econ %>%
  mutate(
    id = row_number(),
    decade = year %/% 10 * 10
  ) %>%
  select(id, country, G WN, year, decade, gdpPercap)
```

```r
# A tibble: 557 x 6
##  id country   GWN year decade gdpPercap
##  <int> <chr>  <dbl> <dbl> <dbl>     <dbl>
## 1     1 Afghanistan 700 1983 1980  863.  
## 2     2 Afghanistan 700 1985 1980  819.  
## 5     5 Algeria     615 1967 1960 1824.  
## 7     7 Algeria     615 1977 1970 2759.  
## 8     8 Algeria     615 1986 1980 3301.  
## 9     9 Algeria     615 2006 2000 3386.  
## 10    10 Angola     540 1953 1950 1126.  
# ... with 547 more rows
```
Basic data wrangling: `group_by()` & `summarize()`

What if we want to know countries’ average GDP per capita over decades?

```r
econ %>%
  mutate(decade = year %% 10 * 10) %>%
  group_by(country, decade) %>%
  summarize(decAvg_gdp = mean(gdpPercap))
```

```
## 'summarise()' has grouped output by 'country'. You can override using the '.groups' argument.
## # A tibble: 382 x 3
## # Groups: country [146]
##   country decade decAvg_gdp
##   <chr>   <dbl>     <dbl>
## 1 Afghanistan 1980     841.
## 2 Afghanistan 1990     601.
## 3 Albania 2000     2962.
## 4 Algeria 1960     1901.
## 5 Algeria 1970     2759.
## 6 Algeria 1980     3301.
## 7 Algeria 2000     3386.
## 8 Angola 1950     1161.
## 9 Angola 2000      825.
##10 Argentina 1900     2992.
## # ... with 372 more rows
```
Saving wrangled data

When you save the wrangled data, don’t overwrite the original data with the same file name:

```python
write_csv(econ, "econ_wrangled.csv")
```
Intermediate data wrangling: second data set

```r
pop <- read_csv("https://students.washington.edu/rllobet/vis/lab1/data/pop.csv")
head(pop)
```

```r
## # A tibble: 6 x 5
## #  country        GWn  year     pop  region
## #  <chr>    <dbl> <dbl>    <dbl> <chr>
## 1 Afghanistan 700 1983 15177000 Asia: Southern Asia
## 2 Afghanistan 700 1985 14519000 Asia: Southern Asia
## 3 Afghanistan 700 1991 15403000 Asia: Southern Asia
## 4 Albania     339 2000  3113000  Europe: Southern Europe
## 5 Algeria     615 1967 13078000 Africa: Northern Africa
## 6 Algeria     615 1968 13495000 Africa: Northern Africa
```

```r
# Compare with econ
head(econ)
```

```r
## # A tibble: 6 x 4
## #  country           GWn  year  gdpPercap
## #  <chr>     <dbl> <dbl>     <dbl>
## 1 Afghanistan 700 1983      863.
## 2 Afghanistan 700 1985      819.
## 3 Afghanistan 700 1991      601.
## 4 Albania     339 2000      2962.
## 5 Algeria     615 1967      1824.
## 6 Algeria     615 1968      1977.
```
Intermediate data wrangling: join family

How do we combine two data sets such that:

```r
## # A tibble: 559 x 6
## #  country     GWN  year gdpPercap   pop region
## # <chr>    <dbl> <dbl>      <dbl>     <dbl> <chr>
## 1 Afghanistan 700 1983    863. 15177000 Asia: Southern Asia
## 2 Afghanistan 700 1985    819. 14519000 Asia: Southern Asia
## 3 Afghanistan 700 1991    601. 15403000 Asia: Southern Asia
## 4 Albania 339 2000    2962. 3113000 Europe: Southern Europe
## 5 Algeria  615 1967   1824. 13078000 Africa: Northern Africa
## 6 Algeria  615 1968   1977. 13495000 Africa: Northern Africa
## 7 Algeria  615 1977   2759. 17058000 Africa: Northern Africa
## 8 Algeria  615 1986   3301. 22520000 Africa: Northern Africa
## 9 Algeria  615 2006   3386. 33749328 Africa: Northern Africa
## 10 Angola   540 1953  1126. NA NA: NA
## # ... with 549 more rows
```
Intermediate data wrangling: join family

Family of join functions: `inner_join`, `left_join`, `right_join`, `full_join`...

data <- econ %>%
  left_join(pop, by = c("GWnd", "year")) %>%
  select(-country.y) %>%
  rename(country = country.x)

## # A tibble: 559 x 6
## # A tibble: 559 x 6
## country  GWnd  year  gdpPercap  pop  region
## <chr>   <dbl> <dbl>      <dbl> <dbl> <chr>
## 1 Afghanistan 700 1983   863. 15177000 Asia: Southern Asia
## 2 Afghanistan 700 1985   819. 14519000 Asia: Southern Asia
## 3 Afghanistan 700 1991   601. 15403000 Asia: Southern Asia
## 4 Albania 339 2000   2962. 3113000 Europe: Southern Europe
## 5 Algeria 615 1967  1824. 13078000 Africa: Northern Africa
## 6 Algeria 615 1968  1977. 13495000 Africa: Northern Africa
## 7 Algeria 615 1977  2759. 17058000 Africa: Northern Africa
## 8 Algeria 615 1986  3301. 22520000 Africa: Northern Africa
## 9 Algeria 615 2006  3386. 33749328 Africa: Northern Africa
## 10 Angola 540 1953  1126.  NA  NA: NA
## # ... with 549 more rows
Intermediate data wrangling: separate (or Regex)

How to separate the region column into continent and sub_region?

```r
## # A tibble: 559 x 7
## country GWn year gdpPercap pop continent sub_region
## <chr> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 Afghanistan 700 1983 863. 15177000 Asia Southern Asia
## 2 Afghanistan 700 1985 819. 14519000 Asia Southern Asia
## 3 Afghanistan 700 1991 601. 15403000 Asia Southern Asia
## 4 Albania 339 2000 2962. 3113000 Europe Southern Europe
## 5 Algeria 615 1967 1824. 13078000 Africa Northern Africa
## 6 Algeria 615 1968 1977. 13495000 Africa Northern Africa
## 7 Algeria 615 1977 2759. 17058000 Africa Northern Africa
## 8 Algeria 615 1986 3301. 22520000 Africa Northern Africa
## 9 Algeria 615 2006 3386. 33749328 Africa Northern Africa
## 10 Angola 540 1953 1126. NA NA NA
## # ... with 549 more rows
```
Intermediate data wrangling: separate (or Regex)

How to separate the region column into continent and sub_region?

```r
data %>%
  separate(region, into = c("continent", "sub_region"), sep = ": ")
```

```r
# A tibble: 559 x 7
# country  GWn  year  gdpPercap  pop continent      sub_region
# <chr>    <dbl> <dbl>   <dbl>  <dbl> <chr>          <chr>
# 1 Afghanistan  700  1983  863. 15177000 Asia Southern Asia
# 2 Afghanistan  700  1985  819. 14519000 Asia Southern Asia
# 3 Afghanistan  700  1991  601. 15403000 Asia Southern Asia
# 4 Albania       339  2000 2962. 3113000 Europe Southern Europe
# 5 Algeria       615  1967 1824. 13078000 Africa Northern Africa
# 6 Algeria       615  1968 1977. 13495000 Africa Northern Africa
# 7 Algeria       615  1977 2759. 17058000 Africa Northern Africa
# 8 Algeria       615  1986 3301. 22520000 Africa Northern Africa
# 9 Angola        540  1953 1126.   NA     NA            NA
# ... with 549 more rows
```
Intermediate data wrangling: separate (or Regex)

How to separate the region column into continent and sub_region?

```r
# Or using regular expression
data %>%
  mutate(continent = str_extract(region, ".*(?=: )"),
         sub_region = str_extract(region, "(?<=: ).*")) %>%
  select(-region)
```

```r
## # A tibble: 559 x 7
## #  country       GWn year gdpPercap pop continent sub_region
## #  <chr> <dbl> <dbl> <dbl> <dbl> <chr>   <chr>
##  1 Afghanistan 700 1983 863. 15177000 Asia Southern Asia
##  2 Afghanistan 700 1985 819. 14519000 Asia Southern Asia
##  3 Afghanistan 700 1991 601. 15403000 Asia Southern Asia
##  4 Albania 339 2000 2962. 3113000 Europe Southern Europe
##  5 Algeria 615 1967 1824. 13078000 Africa Northern Africa
##  6 Algeria 615 1968 1977. 13495000 Africa Northern Africa
##  7 Algeria 615 1977 2759. 17058000 Africa Northern Africa
##  8 Algeria 615 1986 3301. 32520000 Africa Northern Africa
##  9 Algeria 615 2006 3386. 33749328 Africa Northern Africa
## 10 Angola 540 1953 1126. NA NA NA
## # ... with 549 more rows
```
Intermediate data wrangling: case_when

▶ How to convert pop into a new categorical variable, called popCat:
Intermediate data wrangling: case_when

- How to convert pop into a new categorical variable, called popCat:
  - Countries with pop value lower than the first quartile of all pop is classified as “low”
Intermediate data wrangling: *case_when*

- How to convert `pop` into a new categorical variable, called `popCat`:
  - Countries with `pop` value lower than the first quartile of all `pop` is classified as “low”
  - Countries with `pop` value equal to or higher than the first quartile, but lower than the third quartile is classified as “middle”
Intermediate data wrangling: case_when

- How to convert pop into a new categorical variable, called popCat:
  - Countries with pop value lower than the first quartile of all pop is classified as “low”
  - Countries with pop value equal to or higher than the first quartile, but lower than the third quartile is classified as “middle”
  - Countries with pop value equal to or higher than the third quartile is classified as “high”
Intermediate data wrangling: case_when

```r
Qts <- quantile(data$pop, prob = c(0.25, 0.75), na.rm = TRUE)
print(Qts)
```

```r
## 25% 75%
## 3805000 81896000
```

```r
Q1 <- Qts[1]
Q3 <- Qts[2]
data <- data %>%
  mutate(popCat = case_when(pop < Q1 ~ "low",
                            pop >= Q1 & pop < Q3 ~ "middle",
                            pop > Q3 ~ "high"))
```

```r
## # A tibble: 559 x 8
## #  country    GWn  year gdpPercap  pop  continent sub_region popCat
## # <chr> <dbl> <dbl> <dbl> <dbl> <chr> <chr> <chr>
## 1 Afghanistan 700 1983 863. 15177000 Asia Southern Asia middle
## 2 Afghanistan 700 1985 819. 14519000 Asia Southern Asia middle
## 3 Afghanistan 700 1991 601. 15403000 Asia Southern Asia middle
## 4 Albania 339 2000 2962. 3113000 Europe Southern Europe low
## 5 Algeria 615 1967 1824. 13078000 Africa Northern Africa middle
## 6 Algeria 615 1968 1977. 13495000 Africa Northern Africa middle
## 7 Algeria 615 1977 2759. 17058000 Africa Northern Africa middle
## 8 Algeria 615 1986 3301. 22520000 Africa Northern Africa middle
## 9 Algeria 615 2006 3386. 33749328 Africa Northern Africa middle
## 10 Angola 540 1953 1126. NA NA NA NA
## # ... with 549 more rows
```
Focus on USA data again. How to create a variable, named `growth`, that computes the percentage change in `gdpPercap` compared to the immediate last year?

```r
## # A tibble: 114 x 4
##   country                      year gdpPercap growth
##   <chr>        <dbl>      <dbl>    <dbl>
## 1 United States of America 1900  4091.    NA
## 2 United States of America 1901  4464.  0.0912
## 3 United States of America 1902  4421. -0.0097
## 4 United States of America 1903  4551.  0.0295
## 5 United States of America 1904  4410. -0.0311
## 6 United States of America 1905  4642.  0.0528
## 7 United States of America 1906  5079.  0.0941
## 8 United States of America 1907  5065. -0.0028
## 9 United States of America 1908  4561. -0.0996
##10 United States of America 1909  5017.  0.100
### ... with 104 more rows
```
Intermediate data wrangling: mutate and lag

# Extract USA data
USADATA <- data %>%
  filter(country == "United States of America") %>%
  select(country, year, gdpPercap)

# Use `lag` to create a column of gdpPercap in past year
USADATA <- USADATA %>%
  mutate(gdpPercap_lag1 = lag(gdpPercap, n = 1))

print(USADATA)

## # A tibble: 114 x 4
## country         year gdpPercap gdpPercap_lag1
## <chr>           <dbl>    <dbl>          <dbl>
## 1 United States of America 1900  4091.       NA
## 2 United States of America 1901  4464. 4091.
## 3 United States of America 1902  4421. 4464.
## 4 United States of America 1903  4551. 4421.
## 5 United States of America 1904  4561. 4551.
## 6 United States of America 1905  5017. 4561.
## 7 United States of America 1906  5017. 5017.
## 8 United States of America 1907  5017. 5017.
## 9 United States of America 1908  5017. 5017.
## 10 United States of America 1909  5017. 5017.
## # ... with 104 more rows
Intermediate data wrangling: mutate and lag

USAdata <- USAdata %>%
  mutate(growth = (gdpPercap - gdpPercap_lag1) / gdpPercap_lag1)

print(USAdata)

## # A tibble: 114 x 5
## country year gdpPercap gdpPercap_lag1 growth
## <chr> <dbl> <dbl> <dbl> <dbl>
## 1 United States of America 1900 4091. NA NA
## 2 United States of America 1901 4464. 4091. 0.0912
## 3 United States of America 1902 4421. 4464. -0.00969
## 4 United States of America 1903 4551. 4421. 0.0295
## 5 United States of America 1904 4642. 4551. 0.0528
## 6 United States of America 1905 5079. 4642. 0.0941
## 7 United States of America 1906 5017. 5079. -0.00280
## 8 United States of America 1907 5065. 5017. -0.0996
## 9 United States of America 1908 4561. 5065. -0.00010
## 10 United States of America 1909 4561. 4561. 0.100
## # ... with 104 more rows
References


