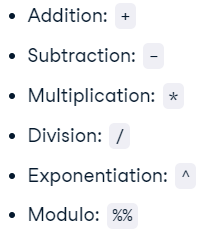
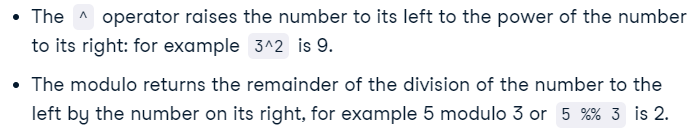
# DataCamp – R

## Introduction to R

### Intro to basics

* Arithmetic with R:





* Assign variable:





* Data type:

# Check class of my\_logical

class(my\_logical)

result:



### Vectors

* Creating a vector:



* Naming a vector:



result:

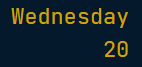


* Choose index of the vector:

poker\_wednesday <- poker\_vector[3]

poker\_wednesday

result:



* Multiple numeric indexing:

poker\_midweek <- poker\_vector[c(2:4)]

result:



* Multimple character indexing:

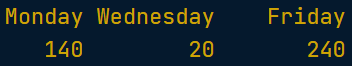
poker\_vector[c("Monday", "Tuesday", "Wednesday")]

* Data subsetting:

selection\_vector <- poker\_vector > 0

poker\_winning\_days <- poker\_vector[selection\_vector]

result:

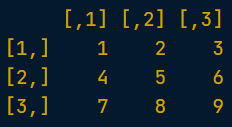


### Matrices

* Creating matrix:

matrix(1:9, byrow=TRUE, nrow=3)

result:



* Creating matrix from vector:

new\_hope <- c(460.998, 314.4)

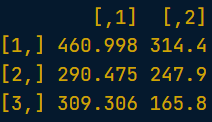
empire\_strikes <- c(290.475, 247.900)

return\_jedi <- c(309.306, 165.8)

box\_office <- c(new\_hope, empire\_strikes, return\_jedi)

star\_wars\_matrix <- matrix(box\_office, byrow=TRUE, nrow=3)

result:



* Naming columns and rows:

rownames(star\_wars\_matrix) = titles

colnames(star\_wars\_matrix) = region

print(star\_wars\_matrix)

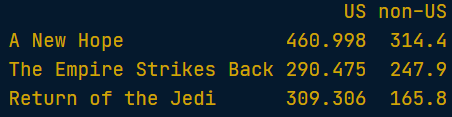
/\

star\_wars\_matrix <- matrix(box\_office,

                      nrow = 3, byrow = TRUE,

                      dimnames = list(titles, region))

result:



* Summing row values:

worldwide\_vector <- rowSums(star\_wars\_matrix)

result:



* Summing colums values:

colSums(all\_wars\_matrix)

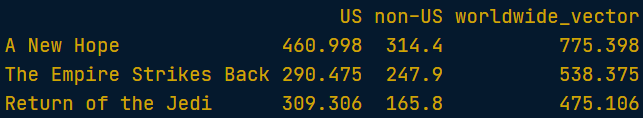
result:



* Adding columns to matrix:

all\_wars\_matrix <- cbind(star\_wars\_matrix, worldwide\_vector)

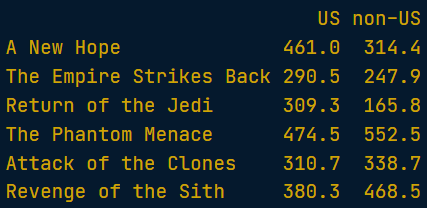
result:



* Adding rows to matrix:

all\_wars\_matrix <- rbind(star\_wars\_matrix, star\_wars\_matrix2)

result:



* Choosing columns and rows from matrix:

non\_us\_all <- all\_wars\_matrix[, 'non-US']

print(mean(non\_us\_all))

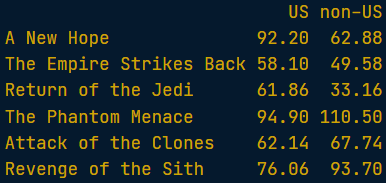
non\_us\_some <- all\_wars\_matrix[1:2, 2]

mean(non\_us\_some)

* Arithmetic with matrices:

visitors <- all\_wars\_matrix / 5

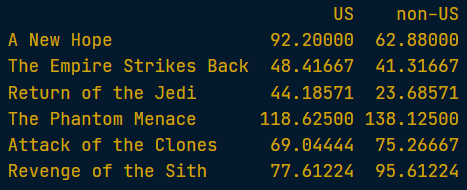
result:



* Matrix operations over matrix:

visitors <- all\_wars\_matrix / ticket\_prices\_matrix

result:



### Factors

**In R, categorical data is stored in factors.**

* Convert vector to categorical variable (factor):

sex\_vector <- c("Male", "Female", "Female", "Male", "Male")

factor\_sex\_vector <- factor(sex\_vector)

print(factor\_sex\_vector)

result:



* Types of cat\_data:
  + Ordinal:

factor\_temperature\_vector <- factor(temperature\_vector, order = TRUE, levels = c("Low", "Medium", "High"))

result:



* + Nominal:

factor\_animals\_vector <- factor(animals\_vector)

result:



* Changing categories names:

levels(factor\_survey\_vector) <- c('Female', 'Male')

factor\_survey\_vector

result:



* Summary info:

survey\_vector <- c("M", "F", "F", "M", "M")

factor\_survey\_vector <- factor(survey\_vector)

levels(factor\_survey\_vector) <- c("Female", "Male")

factor\_survey\_vector

summary(survey\_vector)

summary(factor\_survey\_vector)

result:





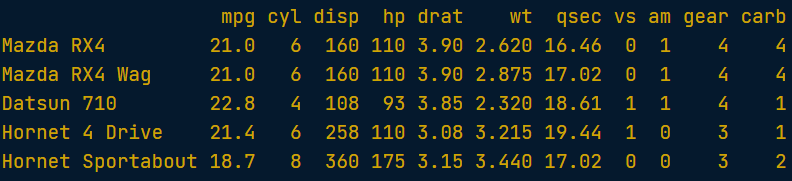
### Data frames

* Head and tail of the DF:

head(mtcars, 5)

tail(mtcars, 5)

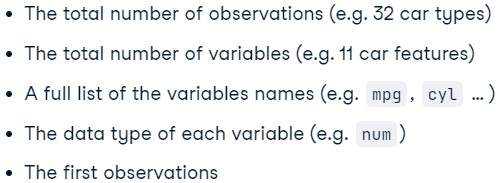
result:



* Overview of the DataFrame:

str(mtcars)

result:

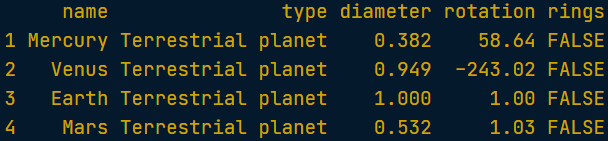


* Creating DataFrame:

planets\_df <- data.frame(name, type, diameter, rotation, rings)

planets\_df

result:



* Indexing DataFrame:

planets\_df[1, 3]

planets\_df[4, ]

/\

planets\_df[1:5, 'diameter']

result:



* Selecting specific column:

rings\_vector <- planets\_df$rings

rings\_vector

result:



* Subsetting dataframe:

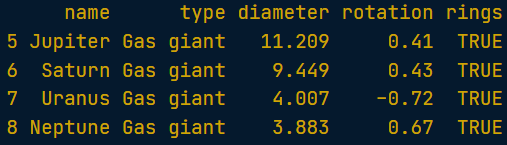
planets\_df[rings\_vector,]

reviews\_df[['comments']]

/\

subset(planets\_df, subset=diameter<1)

result:



* Ordering values in matrix:

c = c(45, 12, 156)

c[order(c)]

result:

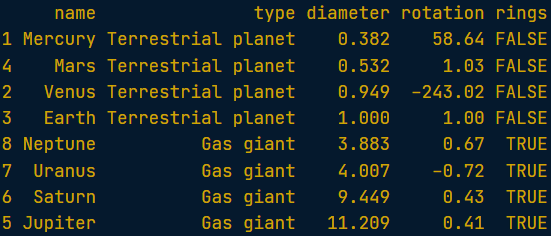


* Sorting dataframe:

positions = order(planets\_df$diameter)

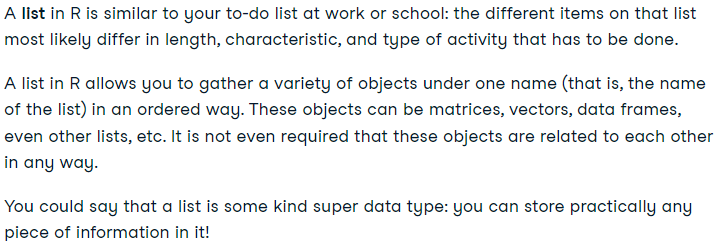
planets\_df[positions, ]

result:



### Lists

* Lists, why would you need them?



* Creating lists:

# Vector with numerics from 1 up to 10

my\_vector <- 1:10

# Matrix with numerics from 1 up to 9

my\_matrix <- matrix(1:9, ncol = 3)

# First 10 elements of the built-in data frame mtcars

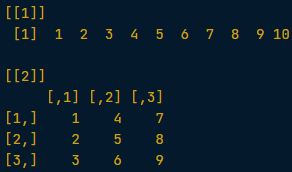
my\_df <- mtcars[1:10,]

# Construct list with these different elements:

my\_list <- list(my\_vector, my\_matrix, my\_df)

my\_list

result:



* Naming lists:

my\_list <- list(vec=my\_vector,

                mat=my\_matrix,

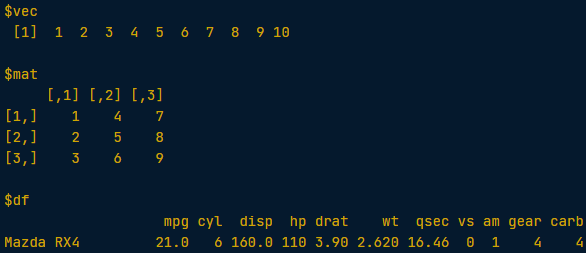
                df=my\_df)

/\

shining\_list <- list(mov, act, rev)

names(shining\_list) = c('moviename', 'actors', 'reviews')

result:



* Selecting lists components:
  + Select list component:

shining\_list[['actors']]

/\

shining\_list$actors

result:



* + Select elements from the component:

shining\_list[['actors']][2]

result:



* All knowladge:

# Use the table from the exercise to define the comments and scores vectors

scores <- c(4.6, 5, 4.8, 5, 4.2)

comments <- c("I would watch it again", "Amazing!", "I liked it", "One of the best movies", "Fascinating plot")

# Save the average of the scores vector as avg\_review

avg\_review = mean(scores)

# Combine scores and comments into the reviews\_df data frame

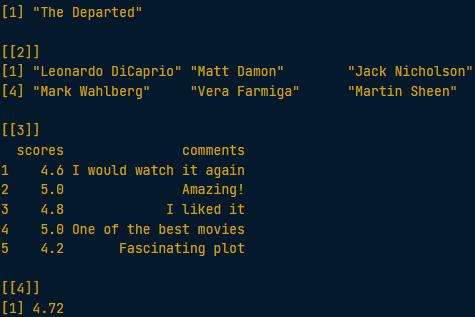
reviews\_df = data.frame(scores, comments)

# Create and print out a list, called

departed\_list = list(movie\_title, movie\_actors, reviews\_df, avg\_review)

departed\_list

result:



## Intermediate R

### Conditionals and Control Flow

#### Relational Operators

* Types:

==

!=

>

<

>=

<=

* Comparing matrix:

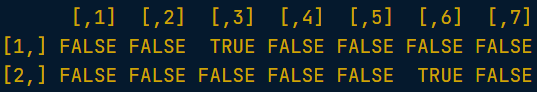
linkedin <- c(16, 9, 13, 5, 2, 17, 14)

facebook <- c(17, 7, 5, 16, 8, 13, 14)

views <- matrix(c(linkedin, facebook), nrow = 2, byrow = TRUE)

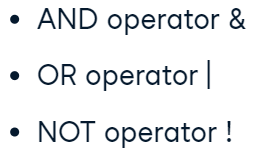
views == 13

result:



#### Logical Operators

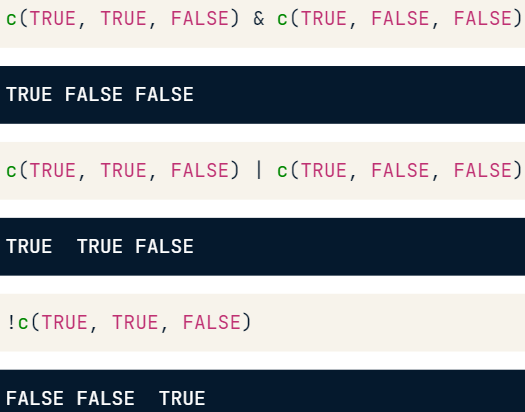
* Types:



* Data type inspection:



* Logical operations with vectors:

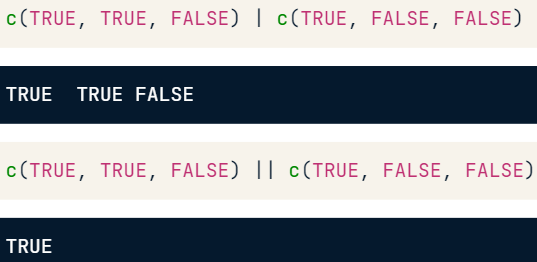


* Difference between “&” and “&&”:

The single sign version returns an entire vector.

The double sign version returns only the result of the OR/AND operator on the first element of each vector.

result:



* Code:

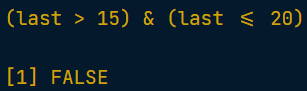
linkedin <- c(16, 9, 13, 5, 2, 17, 14)

last <- tail(linkedin, 1)

(last < 5) | (last > 10)

(last > 15) & (last <= 20)

result:



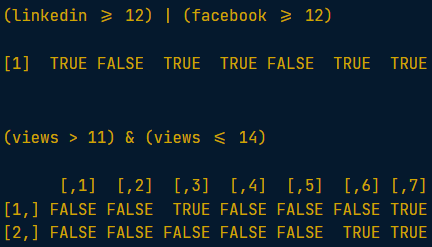
* Comparing vectors and matrices:

(linkedin > 10) & (facebook < 10)

(linkedin >= 12) | (facebook >= 12)

(views > 11) & (views <= 14)

result:



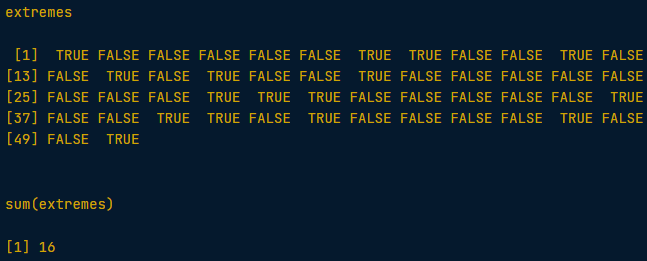
* Choosing vector:

second <- li\_df[,'day2']

extremes = (second>25) | (second<5)

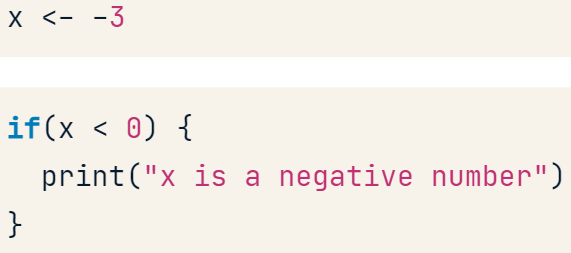
sum(extremes)

result:

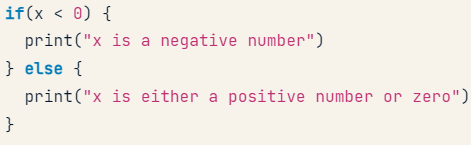


#### Conditional Statements

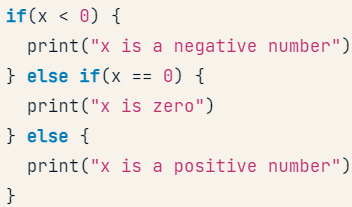
* IF statement:



* ELSE statement:



* ELSE IF statement:



* IF code:

medium <- "LinkedIn"

num\_views <- 14

if (medium == "LinkedIn") {

  print("Showing LinkedIn information")

}

if (num\_views > 15) {

  print("You are popular!")

}

* If-Else statement:

medium <- "LinkedIn"

num\_views <- 14

if (medium == "LinkedIn") {

  print("Showing LinkedIn information")

} else {

  print("Unknown medium")

}

* If-Elseif- Else statement:

if (num\_views > 15) {

  print("You're popular!")

} else if (num\_views <= 15 & num\_views > 10) {

  print("Your number of views is average")

} else {

  print("Try to be more visible!")

}

### Loops

#### While loop

* While loop:

speed <- 64

while (speed > 30) {

  print(paste("Your speed is",speed))

  if (speed > 48) {

    print('Slow down big time!')

    speed <- speed - 11

  } else {

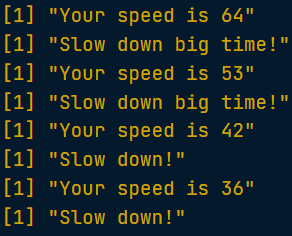
    print('Slow down!')

    speed <- speed - 6

  }

}

result:



* Break statement:

speed <- 88

while (speed > 30) {

  print(paste("Your speed is", speed))

  if (speed > 80) {

    break

  }

  if (speed > 48) {

    print("Slow down big time!")

    speed <- speed - 11

  } else {

    print("Slow down!")

    speed <- speed - 6

  }

}

result:



#### For loop

* For loop with vectors:

linkedin <- c(16, 9, 13, 5, 2, 17, 14)

for (i in linkedin) {

    print(i)

}

for (i in 1:length(linkedin)) {

    print(linkedin[i])

}

* For loop with lists:

nyc <- list(pop = 8405837,

            boroughs = c("Manhattan", "Bronx", "Brooklyn", "Queens", "Staten Island"),

            capital = FALSE)

for (i in nyc) {

    print(i)

}

for (i in 1:length(nyc)) {

    print(nyc[[i]])

}

* Func print with many inputs:

for (i in 1:nrow(ttt)) {

  for (j in 1:ncol(ttt)) {

    pp <- paste("On row", i, "and column", j, "the board contains", ttt[i, j])

    print(pp)

  }

}

* Break and next statements:

linkedin <- c(16, 9, 13, 5, 2, 17, 14)

for (li in linkedin) {

  if (li > 10) {

    print("You're popular!")

  } else {

    print("Be more visible!")

  }

  if (li > 16) {

    print("This is ridiculous, I'm outta here!")

    break

  } else if (li < 5) {

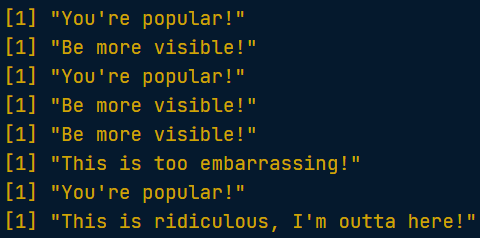
    print('This is too embarrassing!')

    next

  }

}

result:



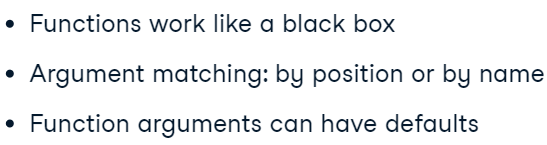
### Functions

#### Introduction to Functions

* Learn about arguments of a function:



* Overall idea:



#### Writing Functions

* Write your own function:

sum\_abs <- function(x, y) {

    return(abs(x) + abs(y))

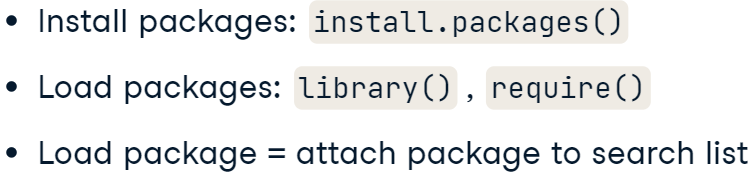
}

sum\_abs(-2, 3)

result:



* Working with packages:



* + Search() – to check all the libraries.

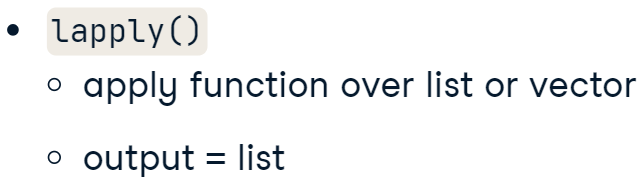
### The apply family

#### lapply

* How it works:

lapply takes a vector or list X, and applies the function FUN to each of its members. If FUN requires additional arguments, you pass them after you've specified X and FUN (...). The output of lapply() is a list, the same length as X, where each element is the result of applying FUN on the corresponding element of X.

* + lapply always returns a **list**. To solve this problem – use unlist() function.



* Code for lapply:

pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")

split <- strsplit(pioneers, split = ":")

split\_low <- lapply(split, tolower)

select\_first <- function(x) {

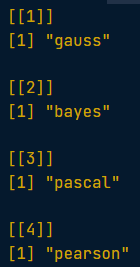
  x[1]

}

names <- lapply(split\_low, select\_first)

names

result:



* Anonymos function:

years <- lapply(split\_low, function(x) {x[2]})

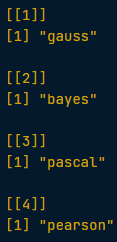
* Using multiple elements in function:

select\_el <- function(x, index) {

  x[index]}

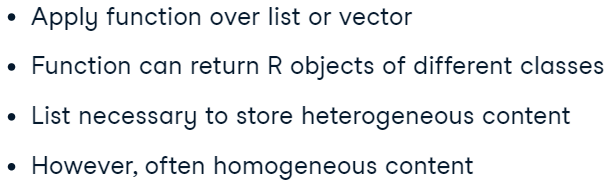
names <- lapply(split\_low, select\_el, index=1)

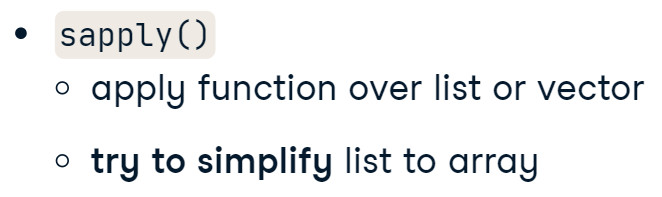
result:



#### sapply

* What does it to?





* Code (if it possible (if the length of the vectors the same) – returns simplified vector):

sapply(temp, max)

result:



* With own functions:

extremes\_avg <- function(x) {

  ( min(x) + max(x) ) / 2, 2

}

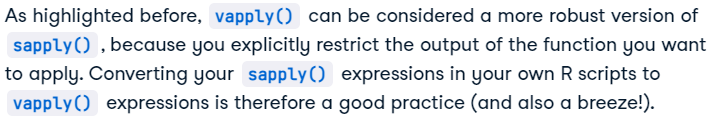
sapply(temp, extremes\_avg)

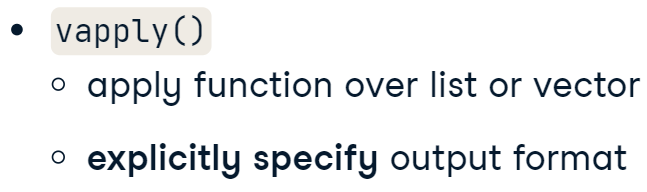
result:



#### vapply

* What is it?





* Code:



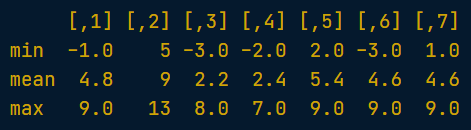
basics <- function(x) {

  c(min = min(x), mean = mean(x), max = max(x))

}

vapply(temp, basics, FUN.VALUE=numeric(3))

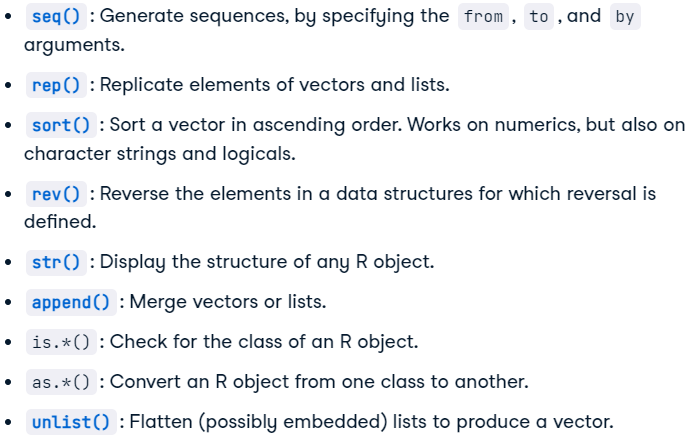
result:



### Utilities

#### Useful Functions

* Functions:



#### Times & Dates

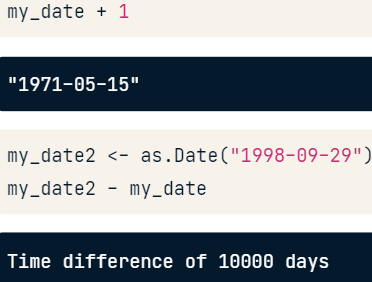
* Creating Date objects:



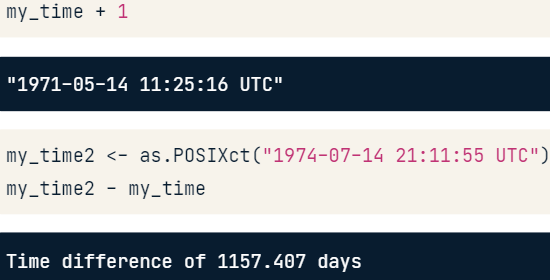
* Dates and times:



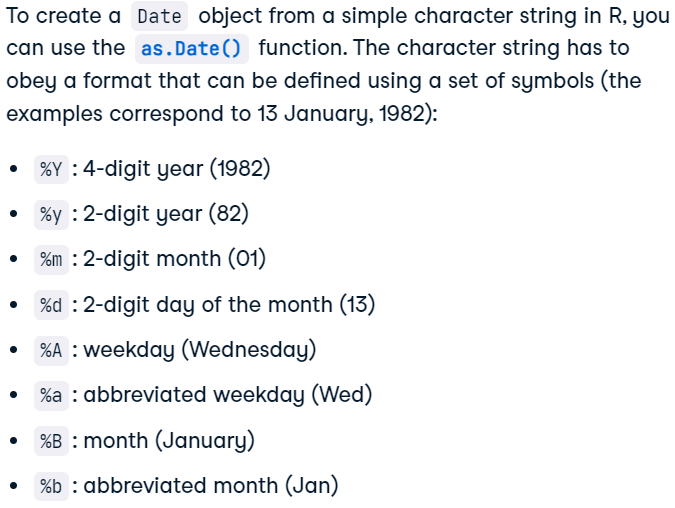
* Arithmetic with dates:



* Arithmetic with dates and times:



* Create and format dates:



* Formatting dates (as.Date – from string to Date; format – from Date to string):

str1 <- "May 23, '96"

str2 <- "2012-03-15"

str3 <- "30/January/2006"

date1 <- as.Date(str1, format = "%b %d, '%y")

date2 <- as.Date(str2, format = "%Y-%m-%d")

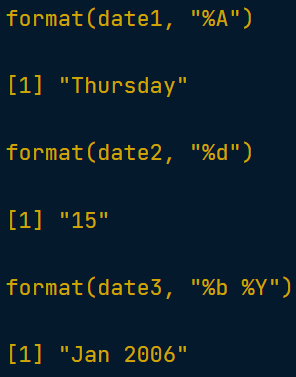
date3 <- as.Date(str3, format = "%d/%B/%Y")

format(date1, "%A")

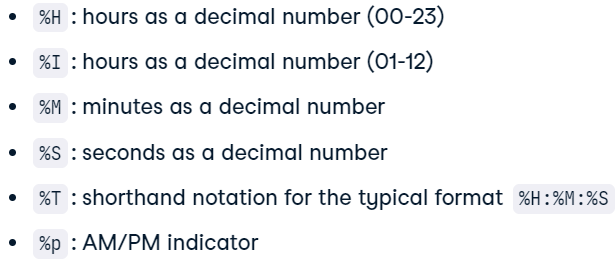
format(date2, "%d")

format(date3, "%b %Y")

result:



* Create and format times:



* Formatting datetimes:

str1 <- "May 23, '96 hours:23 minutes:01 seconds:45"

str2 <- "2012-3-12 14:23:08"

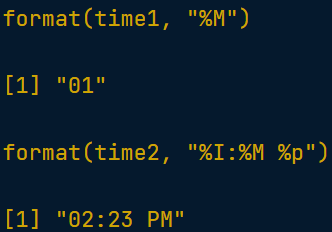
time1 <- as.POSIXct(str1, format="%B %d, '%y hours:%H minutes:%M seconds:%S")

time2 <- as.POSIXct(str2, format="%Y-%m-%d %T")

format(time1, "%M")

format(time2, "%I:%M %p")

result:



* Arithmentics:

day5 - day1

pizza <- c(day1, day2, day3, day4, day5)

day\_diff <- diff(pizza)

print(mean(day\_diff))

result:



## Introduction to the Tidyverse

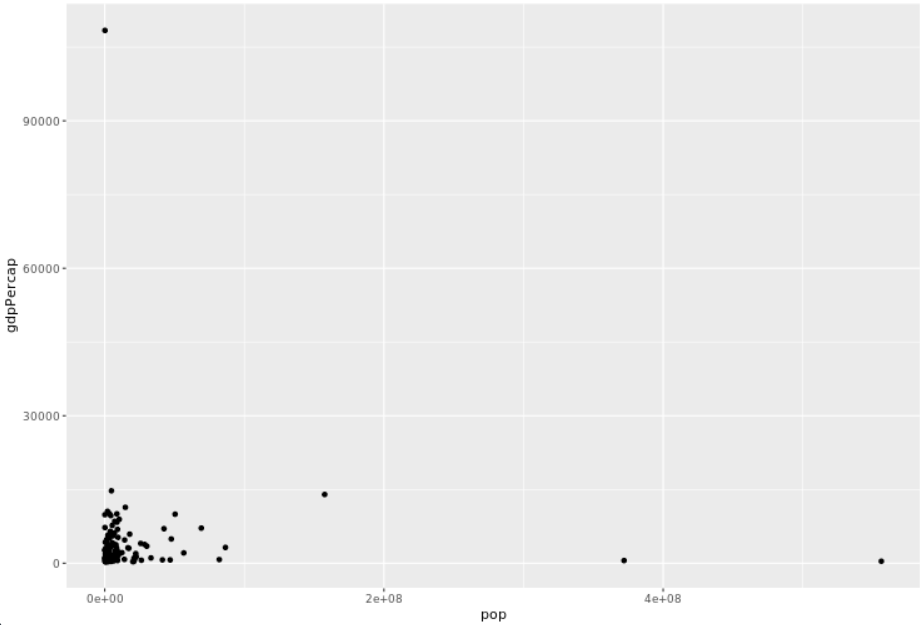
### Data visualization

#### Visualizing with ggplot2

* Scatterplot:

ggplot(gapminder\_1952, aes(x=pop, y=gdpPercap)) + geom\_point()

result:



#### Log scales

* Code:

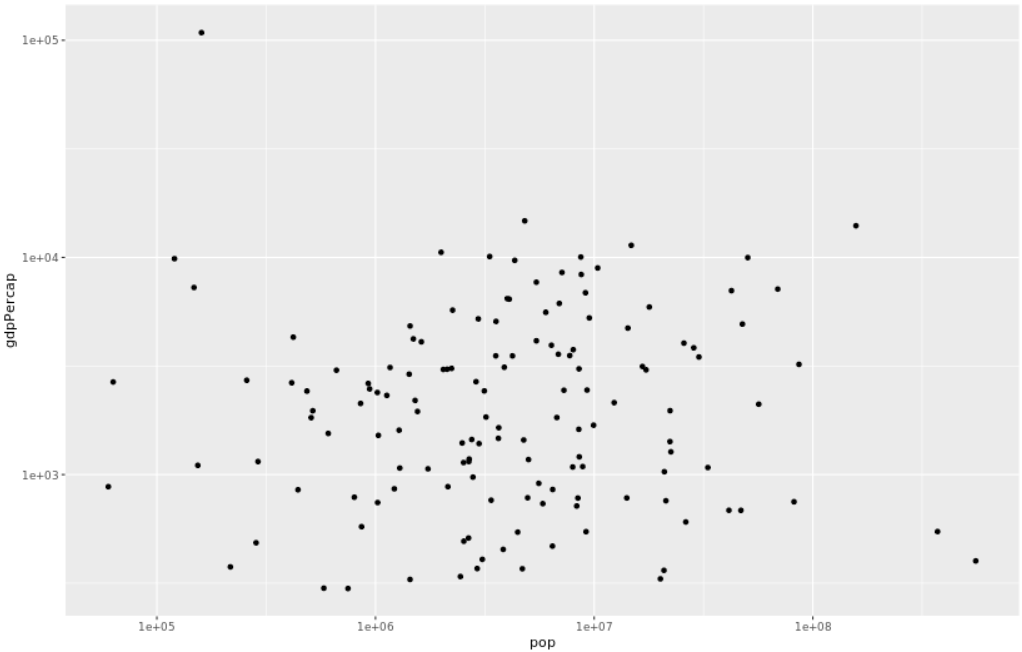
ggplot(gapminder\_1952, aes(x=pop, y=gdpPercap)) +

  geom\_point() +

  scale\_x\_log10() +

  scale\_y\_log10()

result:



* + Explonation:

Both population and GDP per-capita are better represented with log scales, since they vary over many orders of magnitude.

#### Additional aesthetics

* Sizing and coloring:

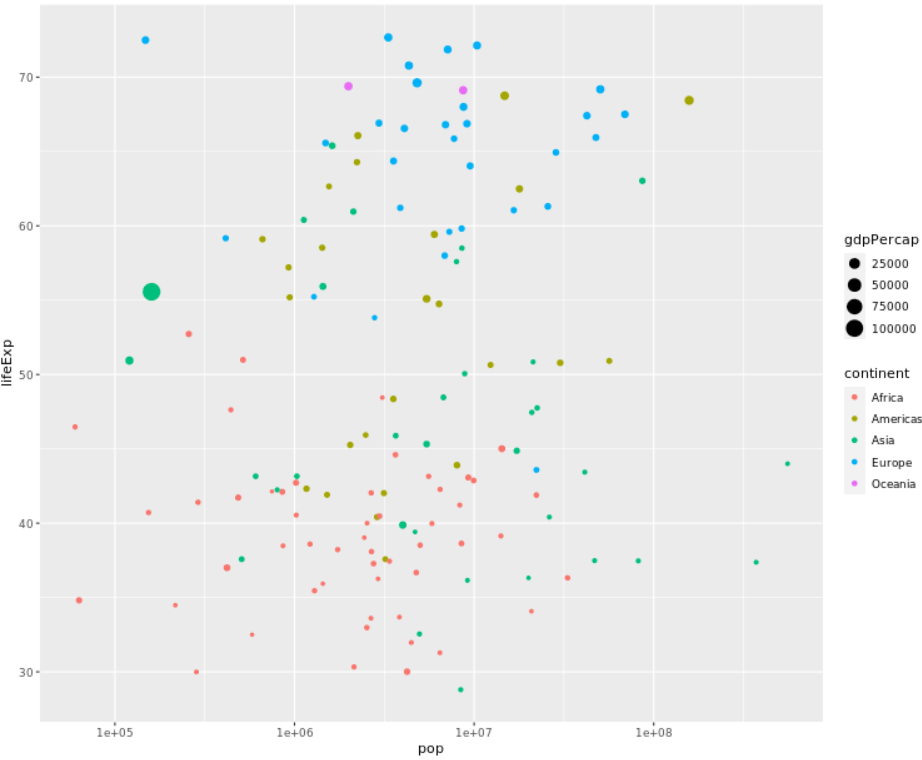
ggplot(gapminder\_1952, aes(x = pop, y = lifeExp,

                            color = continent, size = gdpPercap)) +

  geom\_point() +

  scale\_x\_log10()

result:



#### Faceting

* Code:

color=continent, size=pop)) +

    geom\_point() +

    facet\_wrap(~year) +

    scale\_x\_log10()

result:



### Grouping and summarizing

#### Visualizing summarized data

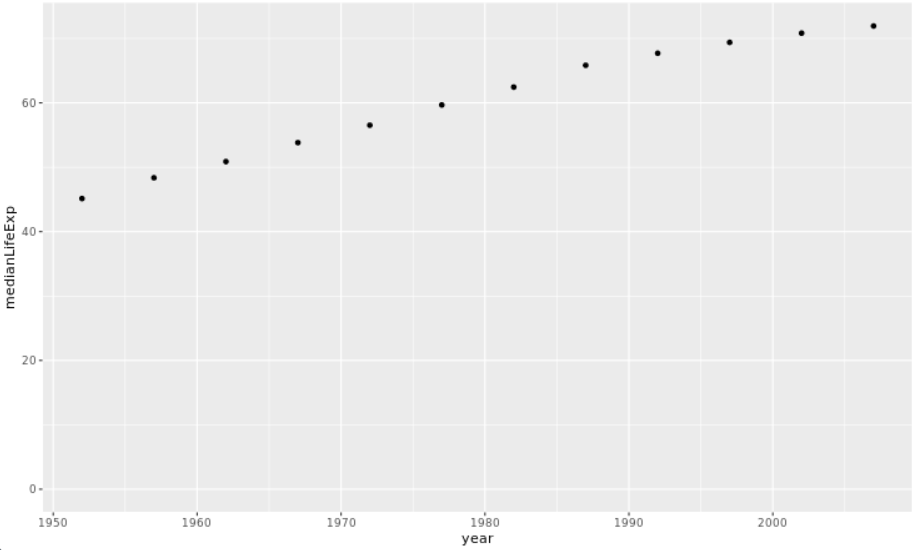
* Start your vis with the “0” y-axis:

ggplot(by\_year, aes(x=year, y=medianLifeExp)) +

  geom\_point() +

  expand\_limits(y=0)

result:



### Types of visualizations

#### Line plots

* Code:

by\_year\_continent <- gapminder %>%

                    group\_by(year, continent) %>%

                    summarize(medianGdpPercap = median(gdpPercap))

ggplot(by\_year\_continent, aes(x=year,

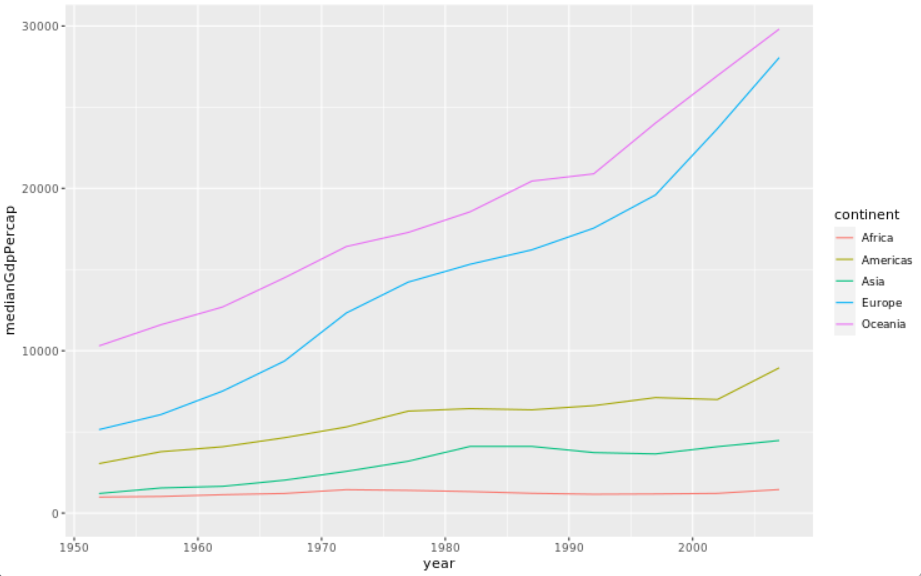
                                y=medianGdpPercap,

                                color=continent)) +

                            geom\_line() +

                            expand\_limits(y=0)

result:



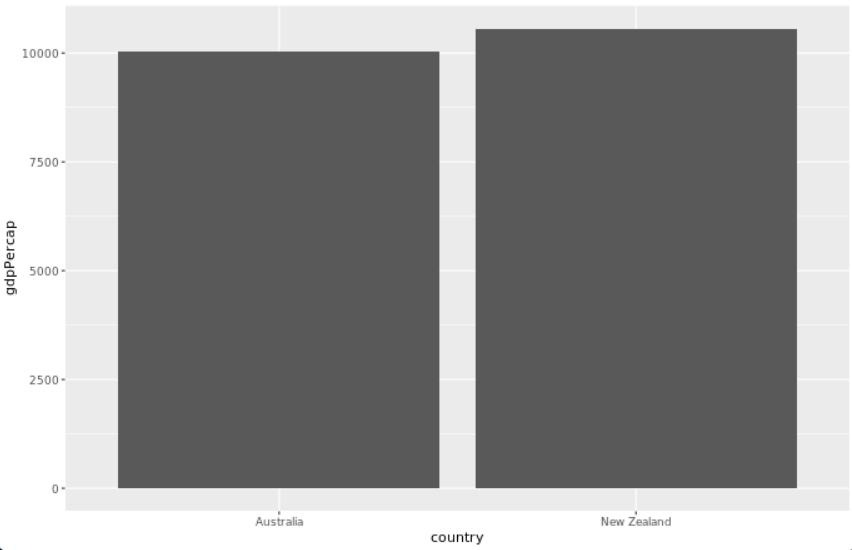
#### Bar plots

* Code:

ggplot(oceania\_1952, aes(x=country, y=gdpPercap)) +

    geom\_col()

result:



#### Histograms

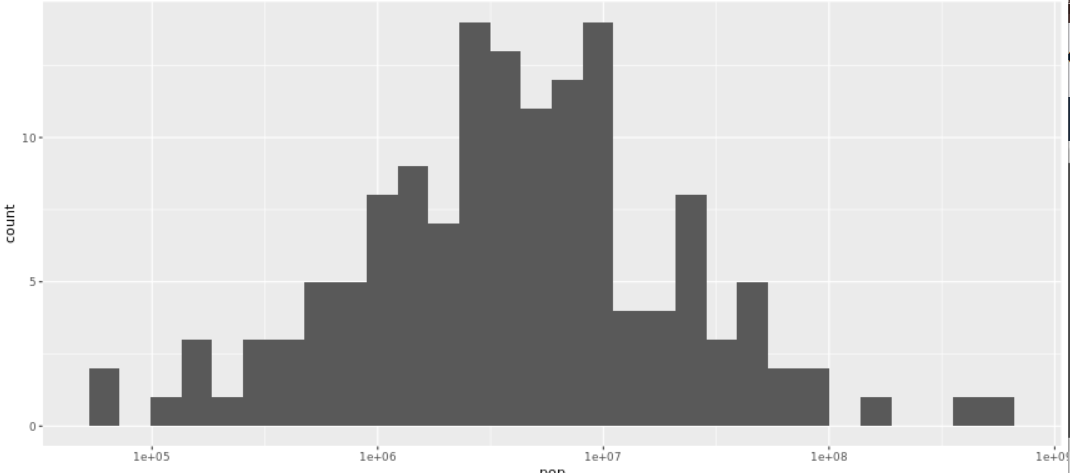
* Code:

ggplot(gapminder\_1952, aes(x=pop)) +

  geom\_histogram(bins=, binwidth) +

  scale\_x\_log10()

result:



#### Boxplots

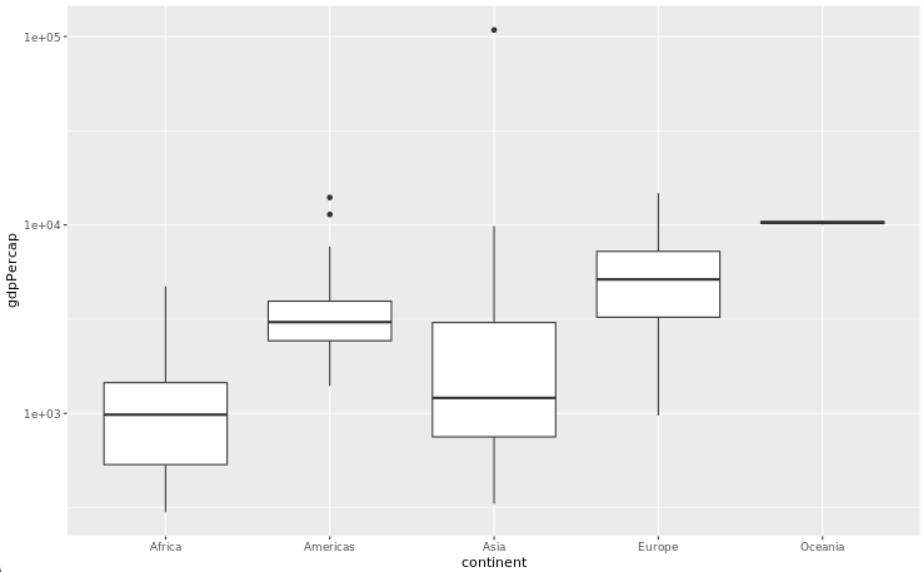
* Code:

ggplot(gapminder\_1952, aes(x=continent, y=gdpPercap)) +

  geom\_boxplot() +

  scale\_y\_log10()

result:



* Adding title:

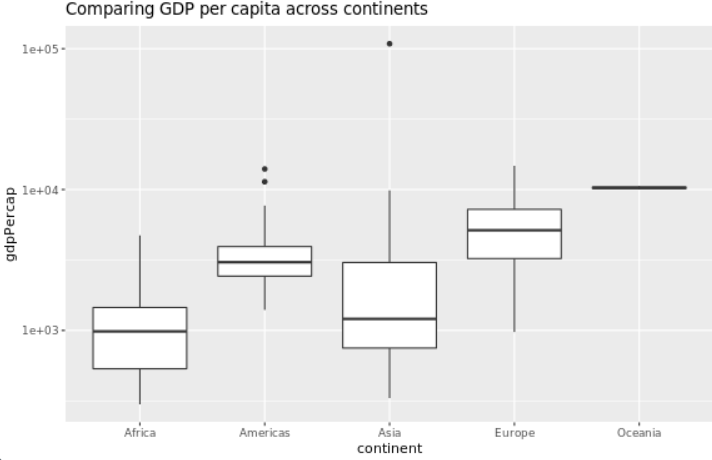
ggplot(gapminder\_1952, aes(x = continent, y = gdpPercap)) +

  geom\_boxplot() +

  scale\_y\_log10() +

  ggtitle("Comparing GDP per capita across continents")

result:



## Data Manipulation with dplyr

### Transforming Data with dplyr

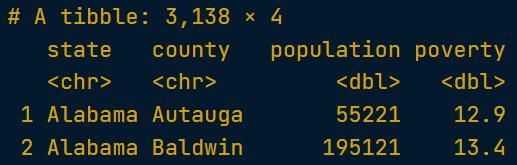
#### Exploring data with dplyr

* Select columns:

counties %>%

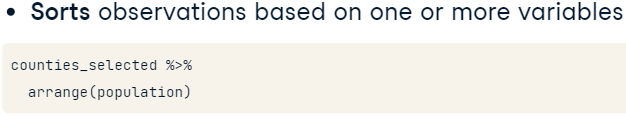
  select(state, county, population, poverty)

result:



#### The filter and arrange verbs

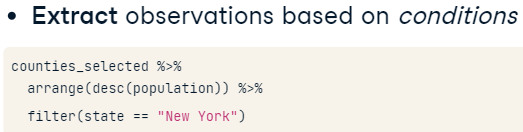
* Sorting by columns:
  + Ascending:



* + Descending:



* Filtering data:



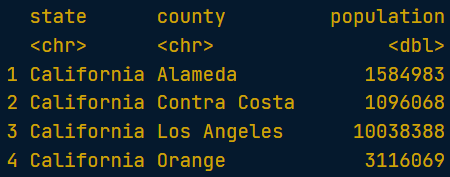
* + Filtering with multiple conditions:

counties\_selected %>%

  filter(population>1000000,

          state=="California")

result:



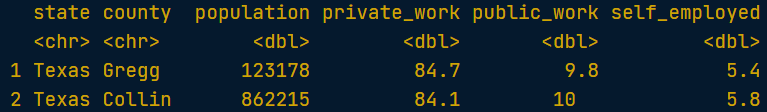
* All together:

counties\_selected %>%

  filter(state=="Texas", population>10000) %>%

  arrange(desc(private\_work))

result:



#### The mutate() verb

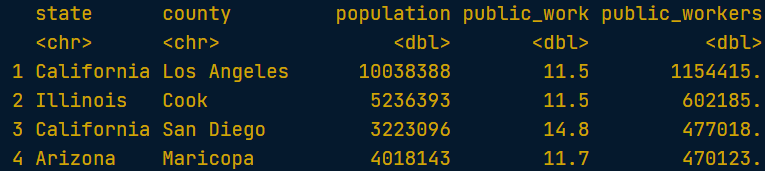
* Adding a new column to the DF:

counties\_selected %>%

  mutate(public\_workers = public\_work \* population / 100) %>%

  arrange(desc(public\_workers))

result:



* Select and transform:

counties %>%

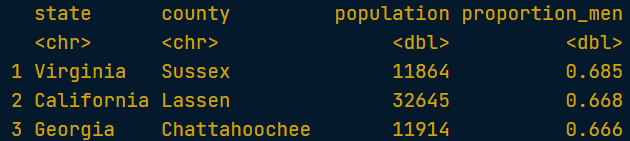
  mutate(state, county, population, proportion\_men=men/population, .keep="none") %>%

  filter(population>10000) %>%

  arrange(desc(proportion\_men))

* + .keep ­­– to include or exclude values.

result:



### Aggregating Data

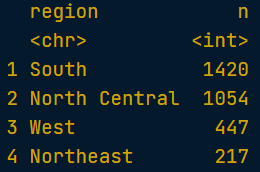
#### The count verb

* Code:

counties\_selected %>%

  count(region, sort=TRUE)

result:

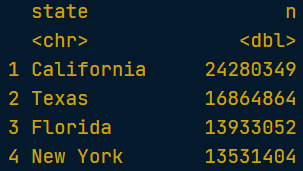


* Getting overall assessment:

counties\_selected %>%

  count(state, wt=citizens, sort=TRUE)

result:



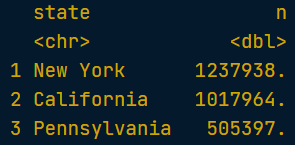
* All in:

counties\_selected %>%

  mutate(population\_work = walk \* population / 100) %>%

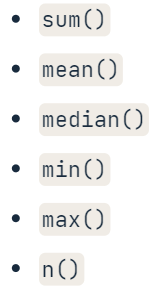
  count(state, wt=population\_work, sort=TRUE)

result:



#### The group\_by, summarize, and ungroup verbs

* Summarize:
  + Functions:



* + Code:

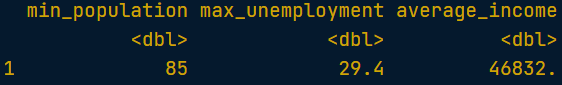
counties\_selected %>%

  summarize(min\_population=min(population),

            max\_unemployment=max(unemployment),

            average\_income=mean(income))

result:



* Groupping:

counties\_selected %>%

  group\_by(state) %>%

  summarize(total\_area = sum(land\_area),

            total\_population = sum(population)) %>%

  mutate(density=total\_population / total\_area) %>%

  arrange(desc(density))

* + Multiple groupping:

counties\_selected %>%

  group\_by(region, state) %>%

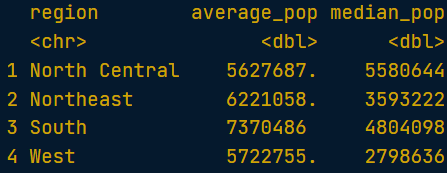
  summarize(total\_pop = sum(population)) %>%

  group\_by(region) %>%

  summarise(average\_pop = mean(total\_pop),

            median\_pop = median(total\_pop))

result:



#### The slice\_min and slice\_max verbs

* slice\_min():

counties\_selected %>%

  group\_by(region, state) %>%

  summarise(average\_income=mean(income)) %>%

  slice\_min(average\_income, n=1)

result:



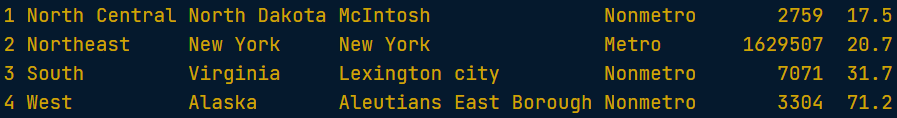
* slice\_max():

counties\_selected %>%

  group\_by(region) %>%

    slice\_max(walk, n=1)

result:



* All functions:

counties\_selected %>%

  group\_by(state, metro) %>%

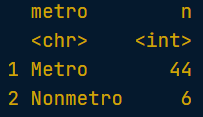
  summarize(total\_pop = sum(population)) %>%

  slice\_max(total\_pop, n = 1) %>%

  ungroup() %>%

  count(metro)

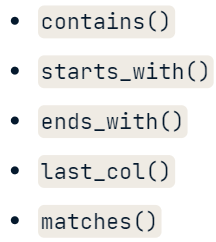
result:



### Selecting and Transforming Data

#### Selecting

* Select helpers:



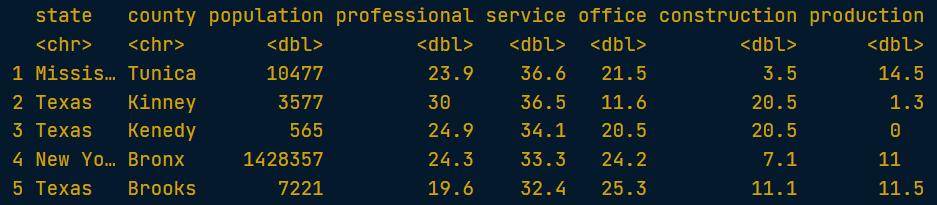
* Selecting several columns:

counties %>%

  select(state, county, population, professional:production) %>%

  arrange(desc(service))

result:



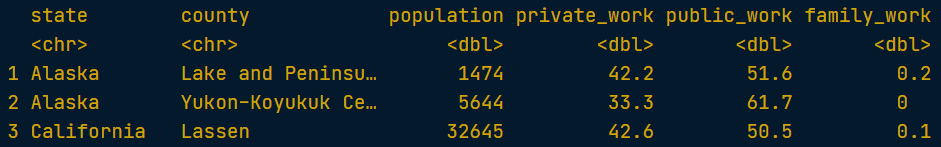
* Ends with:

counties %>%

  select(state, county, population, ends\_with("work")) %>%

  filter(public\_work>=50)

result:



* Eliminating columns:

counties %>%

  select(-state)

#### The rename verb

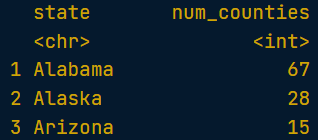
* Renaming:

counties %>%

  count(state) %>%

  rename(num\_counties=n)

result:

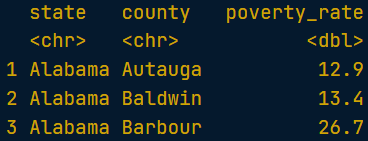


/\

counties %>%

  select(state, county, poverty\_rate=poverty)

result:



#### The relocate verb

* Types:







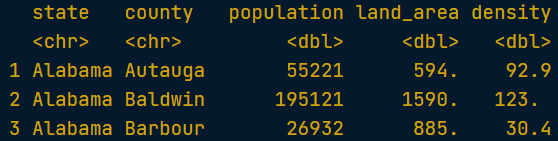
* Relocating:

counties\_selected %>%

  relocate(density, .after=last\_col()) %>%

  relocate(population, .before=land\_area)

result:



### Case Study

* Filter for multipple valies in the same column:

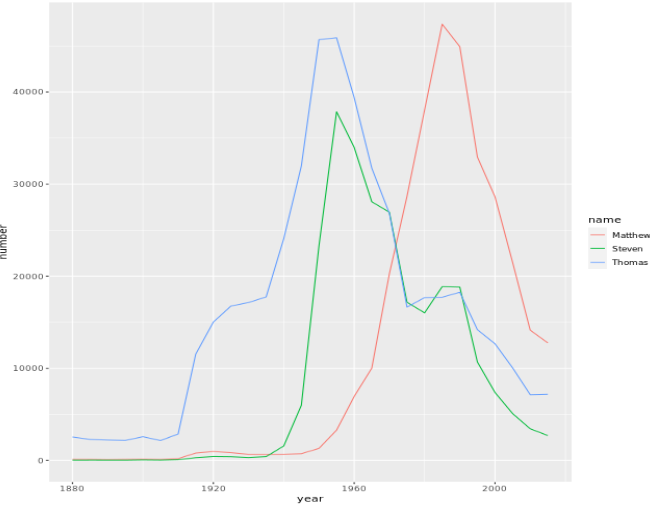
selected\_names <- babynames %>%

  filter(name %in% c("Steven", "Thomas", "Matthew"))

* Plotting:

ggplot(selected\_names, aes(x=year, y=number, color=name)) + geom\_line()

result:



#### Grouped mutates

* Code:

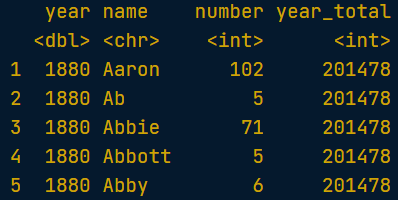
babynames %>%

  group\_by(year) %>%

  mutate(year\_total = sum(number)) %>%

  ungroup()

result:



#### Window functions

* Code (lag):

babynames\_fraction %>%

  arrange(name, year) %>%

  group\_by(name) %>%

  mutate(ratio = fraction / lag(fraction))

result:

