Lecture 8: Data Manipulation and Visualization in R

Chiheb Ben Hammouda

Mathematical Institute, Utrecht University

Python and R Course (WISB153)

June 10, 2025

Course Progress Overview

1. **Week 1**:
   - Thinking like a Computer
   - Introduction to Python (variables, expressions, functions, basic syntax)
2. **Week 2**:
   - Time/space complexity; Big-O definition; Worst/best/average complexity;
   - Data structures (Array); Conditions (if), Python modules, exceptions
3. **Week 3**:
   - Data structures continued (List, queue, stack, dictionary)
   - Iterative algorithms examples (Prime sieve, root) and complexity
   - Loops/repetition (for, while)
4. **Week 4**:
   - Recursive algorithms/examples, general form of recursive algorithm, and the master theorem for time complexity
   - Applications of the Master Thm;
5. **Week 5**:
   - Iterative vs Recursive- Memoization
   - Object-Oriented Programming
6. **Week 6**:
   - Algorithm Design Techniques: Knapsack Problem as Case Study

     ; Version control; Unit tests
7. **Week 7**:
   - Getting Started with R and Its Data Types

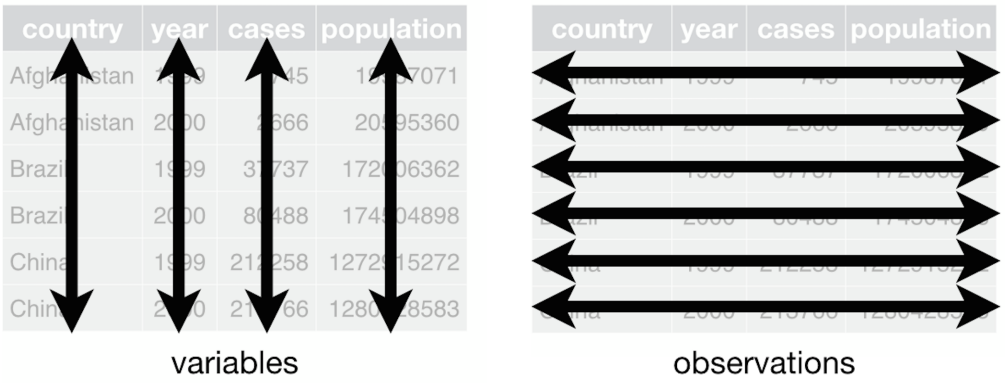
Plan of Lecture 8

1. Data Manipulation in R
   - Introduction and Motivation: The Tidyverse Packages in R
   - The tidyr Package
   - The dplyr Package and the Pipe Operator

2. Data Visualization in R
   - Base R plots
   - ggplot2

Data

Figure: Each row represents an observation (or case), detailing data for a single instance. Each column represents a variable, capturing a specific attribute/characteristics measured across all observations.



Example: The `mtcars` dataset

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Datsun710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet4Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Data manipulation

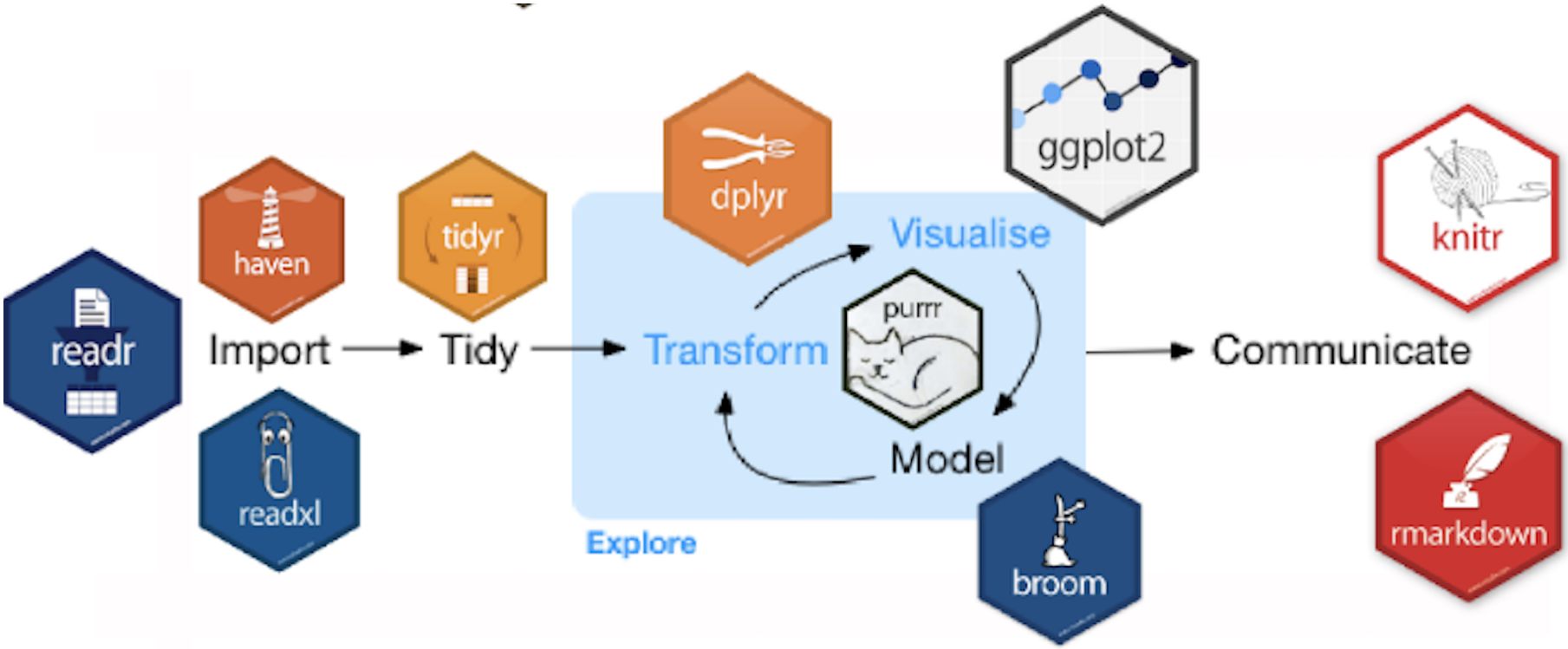
Figure: Each row represents an observation (or case), detailing data for a single instance. Each column represents a variable, capturing a specific attribute/characteristics measured across all observations.

Data analysis often involves data manipulations like:

- Cleaning and organizing data
- Missing value imputation
- Summary statistics for grouped data (e.g., mean, standard deviation)
- Selecting, grouping and (re)arranging observations
- Selecting variables
- Computing new variables, etc.

Tidyverse packages and data processing workflow

- The Tidyverse is a collection of R packages (`tidyr`, `dplyr`, `ggplot2`,...) that adhere to the tidy data principles of data analysis and graphing.
- Purpose: make working with data more efficient.



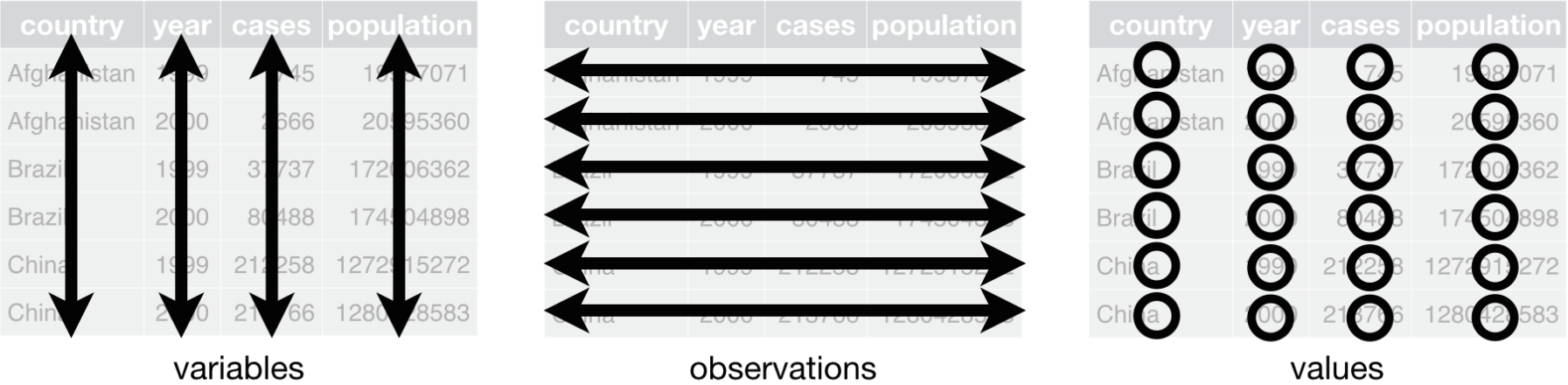
Working with Tidyverse packages

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

- `library(tidyverse)` will load the core tidyverse packages:
  - `tidyr`, for data tidying.
  - `dplyr`, for data manipulation.
  - `ggplot2`, for data visualisation.
  - `readr`, for data import.
  - `purrr`, for functional programming.
  - `tibble`, for tibbles, a modern re-imagining of data frames.
  - `stringr`, for strings.
  - `forcats`, for factors.
  - `lubridate`, for date/times.
- More on tidyverse:
  - https://tidyverse.tidyverse.org
  - https://github.com/tidyverse/tidyverse

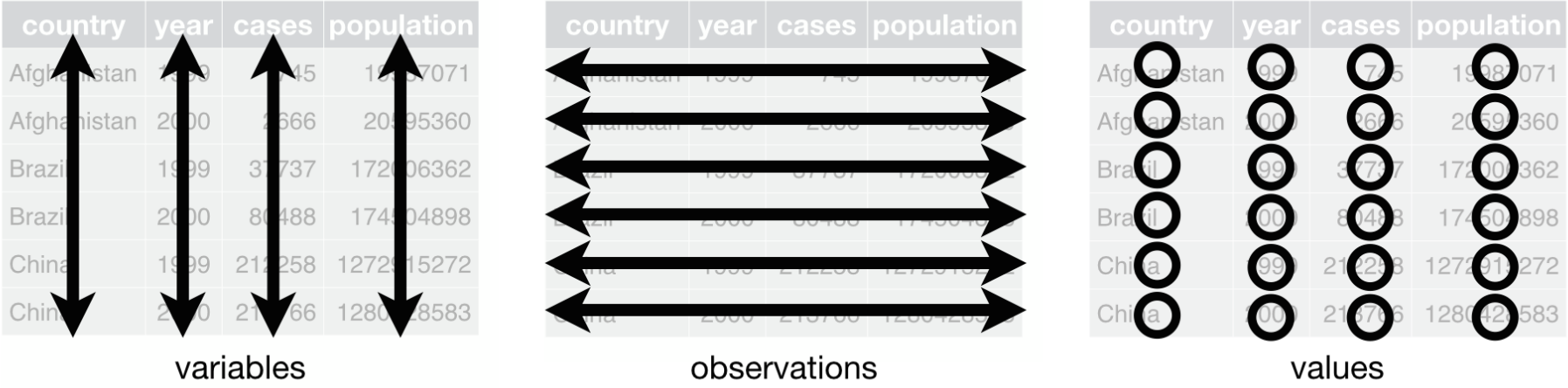
Tidy data

- The `tidyverse` **packages operate on tidy data**.
- What exactly are tidy data?:
  1. Every variable is stored in its own column.
  2. Every observation is stored in its own row—that is, every row corresponds to a single case.
  3. Each value of a variable is stored in a cell of the table.



Tidy data

- The `tidyverse` **packages operate on tidy data**.
- What exactly are tidy data?:
  1. Every variable is stored in its own column.
  2. Every observation is stored in its own row—that is, every row corresponds to a single case.
  3. Each value of a variable is stored in a cell of the table.



The tidyr package in R: Tools to help create tidy data.

Tibbles

Tidyverse package stores data frames as tibbles (enhanced data frames)

- tibbles do not have row names (No automatic row names)
- only first ten rows displayed (Cleaner printing)
- additional information about dimension and storage model variables

```
# Convert the mtcars data frame to a tibble
dplyr::as_tibble(mtcars)
# A tibble: 32 x 11
   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21       6   160   110   3.9  2.62  16.5     0     1     4     4
2   21       6   160   110   3.9  2.88  17.0     0     1     4     4
3  22.8      4   108    93   3.85 2.32  18.6     1     1     4     1
4  21.4      6   258   110   3.08 3.22  19.4     1     0     3     1
5  18.7      8   360   175   3.15 3.44  17.0     0     0     3     2
6  18.1      6   225   105   2.76 3.46  20.2     1     0     3     1
7  14.3      8   360   245   3.21 3.57  15.8     0     0     3     4
8  24.4      4   147    62   3.69 3.19  20.0     1     0     4     2
9  22.8      4   141    95   3.92 3.15  22.9     1     0     4     2
10 19.2      6   168   123   3.92 3.44  18.3     1     0     4     4
# # i 22 more rows

# To print all the data set

print(dplyr::as_tibble(mtcars), width = Inf, n= Inf)
```

Data tidying with `tidyr`

- **The package `tidyr`** (part of the collection of `tidyverse` packages) is designed for reshaping and cleaning data.
- The main `tidyr` functions for tidying data are:
  - `pivot_longer()`: transforms wide data to long format
  - `pivot_wider()`: transforms long data to wide format
  - `separate()`: splits a single column into multiple columns
  - `unite()`: combines multiple columns into a single column
  - `drop_na()`: removes rows with missing values
  - `fill()`: fills missing values with the next or previous value

Data tidying with tidyr: Example with pivot_longer () and pivot_wider()

Reshape Data - Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

```
pivot_longer(table4a, cols = 2:3, names_to ="year",
  values_to = "cases")
```

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name", values_from = "value"**)**

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

```
pivot_wider(table2, names_from = type,
  values_from = count)
```

Figure: From Data tidying with tidyr : : CHEATSHEET

R Example for pivot_longer()

```r
library(tidyr)      # Load necessary library

# Example data for pivot_longer
table4a <- data.frame(
country = c("A", "B", "C"),
`1999` = c(700, 37000, 212000),
`2000` = c(2000, 80000, 213000)
)

# Rename columns back to "1999" and "2000"
colnames(table4a)[2:3] <- c("1999", "2000")
print(table4a)    # Print the original data

# Use pivot_longer to reshape data
long_data <- pivot_longer(table4a, cols = 2:3,
                  names_to = "year", values_to = "cases")

# Print the reshaped data
print(long_data)
```

R Example for pivot_wider()

```
# Example data for pivot_wider
table2 <- data.frame(
country = c("A", "A", "B", "B", "C", "C"),
year = c(1999, 1999, 2000, 2000, 1999, 1999),
type = c("cases", "pop", "cases", "pop", "cases", "pop"),
count = c(0.7, 19, 37, 172, 212, 1)
                                )

print(table2)     # Print the original data

# Use pivot_wider to reshape data
wide_data <- pivot_wider(table2, names_from = type,
                                 values_from = count)

# Print the reshaped data
print(wide_data)
```

tidyr: More functions, examples and documentation

- Data tidying with tidyr : : CHEATSHEET
- Tidyr documentation
- Tidyr examples

Data manipulation with `dplyr`

- The package `dplyr` (part of the collection of `tidyverse` packages) is specifically designed for data manipulation.
- The main `dplyr` functions for data manipulation are: .
  - `summarize()`: summary statistics for grouped data
  - `select()`: selecting variables
  - `filter()`: filtering observations
  - `arrange()`: (re)arranging observations
  - `mutate()`: create, modify, and delete variables
  - `group_by()`: operations done on groups defined by variables

Recall mtcars dataset

The `mtcars` dataset contains data extracted from the 1974 Motor Trend US magazine and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

```
# Display the first 6 rows of the mtcars data frame
head(mtcars,6)
```

```
                   mpg   cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0    6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0    6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8    4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4    6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7    8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1    6  225 105 2.76 3.460 20.22  1  0    3    1
```

mpg: Miles/(US) gallon; **cyl**: Number of cylinders; **disp**: Displacement (cu.in.); **hp**: Gross horsepower; **drat**: Rear axle ratio; **wt**: Weight (1000 lbs); **qsec**: 1/4 mile time; **vs**: Engine (0 = V-shaped, 1 = straight); **am**: Transmission (0 = automatic, 1 = manual); **gear**: Number of forward gears; **carb**: Number of carburetors

Why dplyr?

- **More intuitive and easier than base R functions**
  - no need for working with square brackets

- Example: **Order rows of the** `mtcars` **data set on ascending values of** `mpg`

  The base R function `order()` **needs** `[]`**:**

```
mtcars[order(mtcars$mpg), ]
```

  The `dplyr` **function** `arrange()` **is more intuitive:**

```
arrange(mtcars, mpg)
```

The pipe operator in R: **Shortcut key:** `Ctrl/Cmd + M`

- Operations performed by the `dplyr` functions can be combined with the pipe operator %>%, resulting in efficient code that is more intuitive and easier to understand than code written with the corresponding base R functions.
- `dplyr` **in combination with the pipe operator:**
  - performs a series of operations step-by-step
  - no need to write code inside-out (nested code)
  - no need to make intermediate objects

Pipe operator %>%

- **Basic principle**: Passes (transformed) data on to the next operation

```
data %>%
perform operation A and pass on transformed data %>%
perform operation B and pass on transformed data %>%
etc.
```

- **Pipes work as follows**: the object preceding the pipe is by default **the first argument** of the function after the pipe. Hence

f(x)	becomes	x %>% f()
f(g(x))	becomes	x %>% g() %>% f()

Pipe operator %>%: Example

1. Sort the rows of `mtcars` on ascending values of `mpg`
2. Select only cars with 4 cylinders
3. Display the variables `disp` and `qsec`

Base R code

```
subset(mtcars[order(mtcars$mpg), ], subset = cyl == 4,
                                    select = c(disp, qsec))
```

`dplyr` **with pipe operator**

```
mtcars %>%
filter(cyl == 4) %>%
arrange(mpg) %>%
select(disp, qsec)
```

Datasets & pre-processing

Question: Why bother using pipes?

Answer: Better readable and less memory used

If multiple steps in pre-processing are needed, one might get something like

```
apps <- arrange(
        select(
          filter(
             read_csv('data/googleplaystore.csv'),
               'Content Rating' == "Teen")
        ,Rating, Reviews, Type)
        ,-Rating)
```

or

```
apps <- read_csv('data/googleplaystore.csv')
apps1 <- filter(apps, 'Content Rating' == "Teen")
apps2 <- select(apps1, Rating, Reviews, Type)
apps3 <- arrange(apps2, -Rating)
```

With piping, we have

```
apps <- read_csv('data/googleplaystore.csv') %>%
filter('Content Rating' == "Teen") %>%
select(Rating, Reviews, Type) %>%
arrange(-Rating)
```

For more details on pipe: see Pipes style guide

Summary statistics

The `dplyr` function for summarizing data

```
data %>% summarise(...)
```

- Replace the ellipses with appropriate summary function(s), e.g.
  - `mean()`, `median()`, `sd()`, `var()`, `sum()` etc. for numeric variables
  - `n()`, `n_distinct()` for counts
  - many others (see help page)

Example (with and without name for summary statistic)

```
mtcars %>% summarise('mean of mpg' = mean(mpg), sd(mpg))
```

```
      mean of mpg  sd(mpg)
1       20.09062    6.026948
```

Summaries for groups

The `dplyr` function for grouping rows of a data frame

```
data %>%
group_by(...) %>% # replace ... with grouping variables
summarise(...)  # replace ... with summary function(s), e.g. mean()
```

In combination with `summarise()`:

```
mtcars %>%
group_by(cyl) %>%
summarise(mean(mpg), sd(mpg))
```

```
# A tibble: 3 x 3
      cyl `mean(mpg)` `sd(mpg)`
    <dbl>       <dbl>     <dbl>
1     4          26.7      4.51
2     6          19.7      1.45
3     8          15.1      2.56
```

For more details on group_by(...): see help page

Exercise

Display the maximum of `mpg` for each level of `cyl`.

Contingency tables

Two different ways to build contingency tables

```
data %>%
group_by(data, ...) %>%
summarise(n())
```

or

```
data %>% count(...)
```

Example: **Contingency table of `vs` and `am`:**

```
mtcars %>% count(am, vs)
```

```
# A tibble: 4 x 3
      am    vs     n
   <dbl> <dbl> <int>
1      0     0    12
2      0     1     7
3      1     0     6
4      1     1     7
```

Exercise

Display the contingency table of `gear` **and** `cyl` **using** `summarise()`**.**

Filtering cases

Select cases that satisfy logical condition(s) ...

```
data %>% filter(...)
```

Example

```
mtcars %>% filter(mpg > 25 & gear == 4)
```

```
# A tibble: 4 x 11
   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  32.4     4  78.7    66  4.08  2.20  19.5     1     1     4     1
2  30.4     4  75.7    52  4.93  1.61  18.5     1     1     4     2
3  33.9     4  71.1    65  4.22  1.84  19.9     1     1     4     1
4  27.3     4  79.0    66  4.08  1.93  18.9     1     1     4     1
```

For more details on filter(...): see help page

Exercise

Display

- the frequency table of `gear` and `cyl`
- with only frequencies greater than 1 and
- the frequencies displayed in ascending order

select()

Selecting and deselecting (i.e., excluding) variables

```
data %>% select(var1, var2)     # select variables var1 and var2
data %>% select(-var1, -var2)   # deselect variables var1 and var2
```

Deselecting the first 5 columns

```
mtcars %>% select(-(1:5))
```

```
                    wt   qsec    vs  am  gear  carb
Mazda RX4          2.620 16.46   0   1     4     4
Mazda RX4 Wag      2.875 17.02   0   1     4     4
Datsun 710         2.320 18.61   1   1     4     1
Hornet 4 Drive     3.215 19.44   1   0     3     1
Hornet Sportabout  3.440 17.02   0   0     3     2
Valiant            3.460 20.22   1   0     3     1
Duster 360         3.570 15.84   0   0     3     4
Merc 240D          3.190 20.00   1   0     4     2
Merc 230           3.150 22.90   1   0     4     2
Merc 280           3.440 18.30   1   0     4     4
```

For more details on select(...): see help page

Exercise

Check the help page of `select()` and display

- the first two variables and
- all variables starting with the letter "d".

arrange()

Sort rows according to ascending or descending values of specified columns

```
data %>% arrange(var1, var2)   # ascending on var1 and var2
data %>% arrange(var1, -var2) # ascending on var1 and descending on var2
```

Sort rows of `mtcars` ascending on `cyl` and descending on `mpg`

```
mtcars %>% arrange(cyl, -mpg)
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla   33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128         32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic      30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa     30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Fiat X1-9        27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2    26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Merc 240D        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Merc 230         22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280         19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

For more details on arrange(...): see help page

Exercise

Display the means of `mpg` for `gear` from low to high.

Computing new variables

```
data %>% mutate(..., .keep = c("all", ...), .before = NULL, .after = NULL)
```

- adds the new variable(s) to the data frame
- `.keep` specifies which variables to return, "all", "used", "unused", "none"
- `.before` or `.after` determine where the new variables are inserted

The effect of `.keep = "used"`

```
mtcars %>% mutate(disp_cyl = disp/cyl, .keep = "used")
```

```
                  cyl  disp disp_cyl
Mazda RX4           6 160.0  26.66667
Mazda RX4 Wag       6 160.0  26.66667
Datsun 710          4 108.0  27.00000
Hornet 4 Drive      6 258.0  43.00000
Hornet Sportabout   8 360.0  45.00000
Valiant             6 225.0  37.50000
Duster 360          8 360.0  45.00000
Merc 240D           4 146.7  36.67500
Merc 230            4 140.8  35.20000
Merc 280            6 167.6  27.93333
```

For more details on mutate(...): see help page

Exercise

Display the variable names when you rerun the previous command with the arguments `.keep = "none"` **and** `.keep = "unused"`

dplyr: More functions, examples and documentation

- Data manipulation with dplyr : : CHEATSHEET
- dplyr documentation
- dplyr examples

Main Plot Functions

High-level plot functions (create plot)

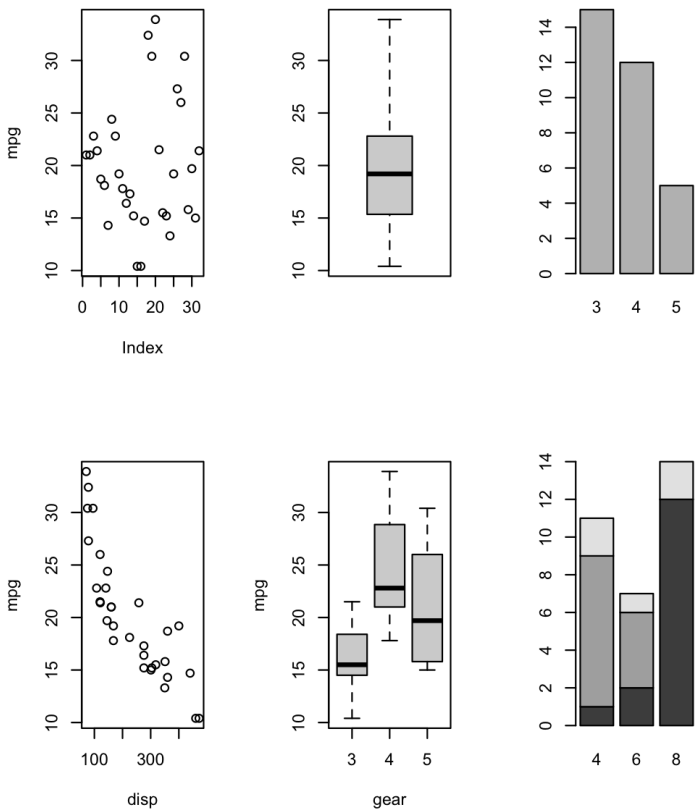
```
plot(x, y = NULL)          # Basic plot; e.g, scatterplot
hist(x, ...)               # Histogram of x
boxplot(formula, data)     # Boxplot of data according to formula
barplot(formula, data)     # Barplot of data according to formula
```

Low-level plot functions (add something to existing plot)

```
points(x, y)               # Add points to an existing plot
lines(x, y)                # Add lines to an existing plot
abline(a, b, h, v)         # Add straight lines (horizontal, vertical, ...)
text(x, y, labels)         # Add text labels to points in an existing plot
legend(x, y, legend)       # Add a legend to the plot
```

Example Plots with mtcars

```
par(mfrow = c(2, 3))
with(mtcars, plot(mpg))
with(mtcars, boxplot(mpg))
with(mtcars, plot(factor(gear)))
with(mtcars, plot(disp, mpg))
with(mtcars, boxplot(mpg ~ gear))
with(mtcars, barplot(xtabs(~ gear + cyl)))
```



The data Argument

```
ggplot(data = <data>)

# or simply

ggplot(<data>)
```

The `data` argument tells `ggplot()` where to look for the variables:

- `ggplot()` initializes a plot object of class `ggplot`.
- `data` specifies the data set.

Example:
Load the package `ggplot2` **and run:**

```
ggplot(data = mtcars)
```

Aesthetics

The `aes` arguments tell `ggplot()` where to map the variables:

```
ggplot(data = <data>, mapping = aes(x = <var1>, y = <var2>)

# or simply

ggplot(<data>, aes(<var1>, <var2>))
```

- `var1` is mapped to the x-axis, `var2` to the y-axis.

Example: Display a plot that maps:

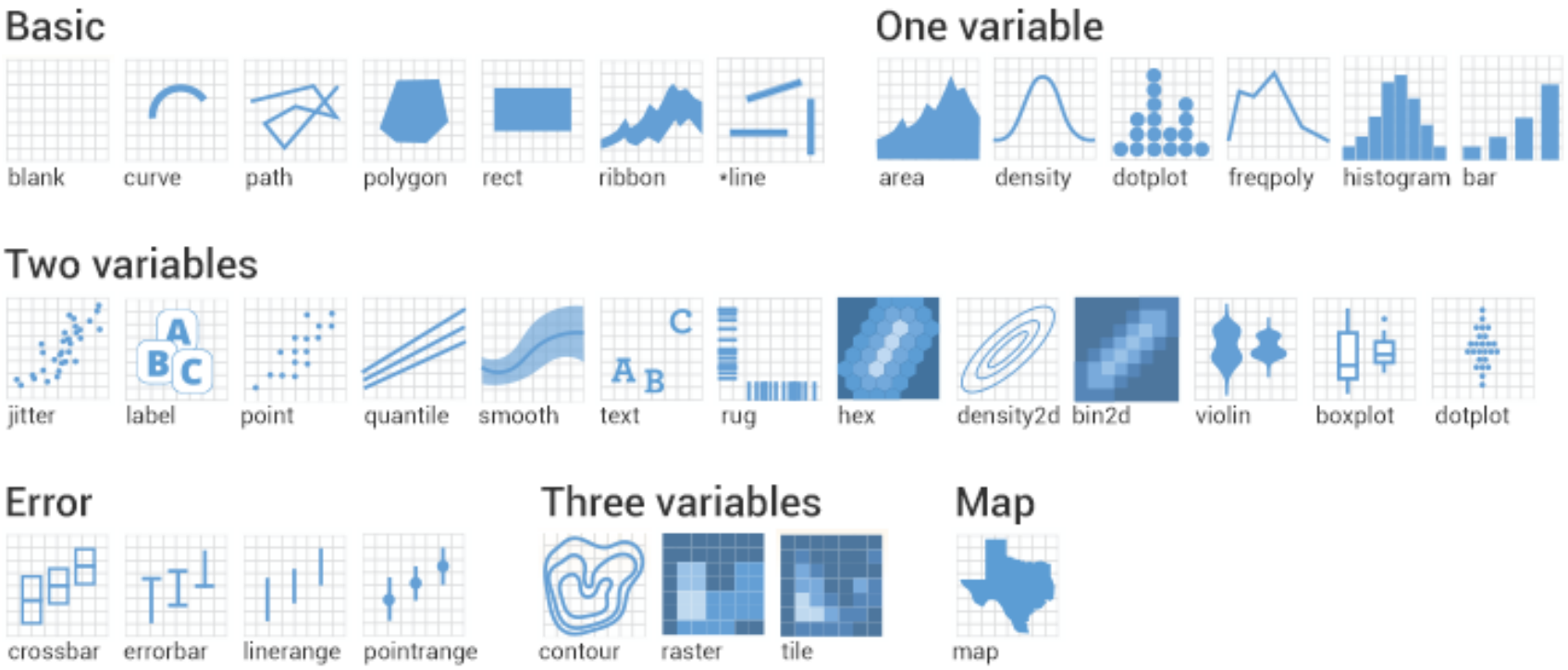
- `disp` to the x-axis
- `mpg` to the y-axis

```
ggplot(mtcars, aes(x = disp, y = mpg))
```

Geometries

The `geoms` tell `ggplot()` what shapes to display, e.g.,

```
ggplot(<data>, aes(x = <var>, y = <var>)) +
geom_xxx()
```

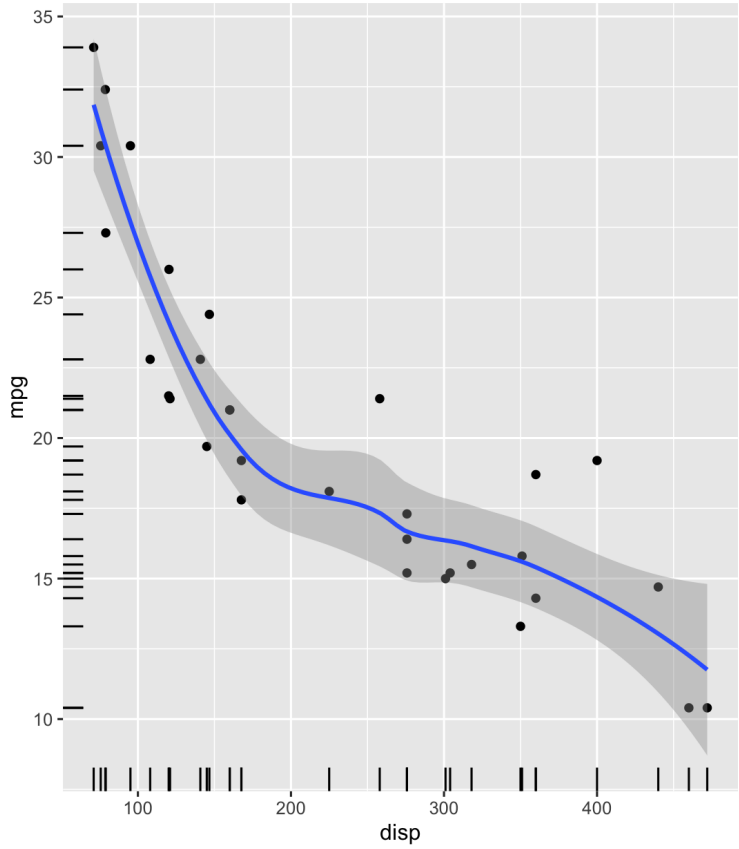


More on the different geometries:
https://ggplot2.tidyverse.org/reference/

Adding Other Geometries

Adding a regression line and a rug

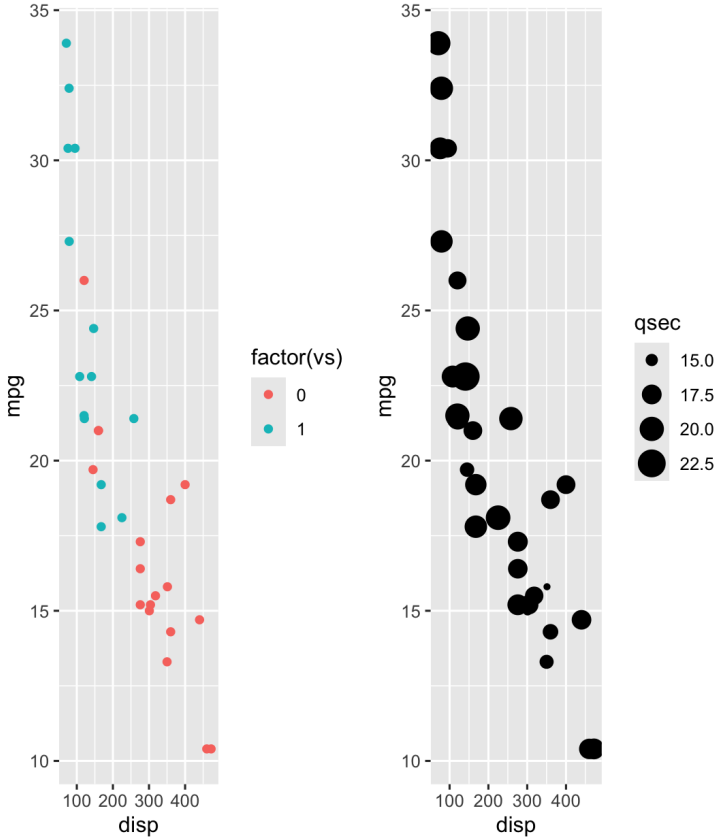
```
ggplot(mtcars, aes(disp, mpg)) +
geom_point() +
geom_smooth() +
geom_rug()
```



Other Aesthetics

Change the color, size, and/or shape of the points by a third variable

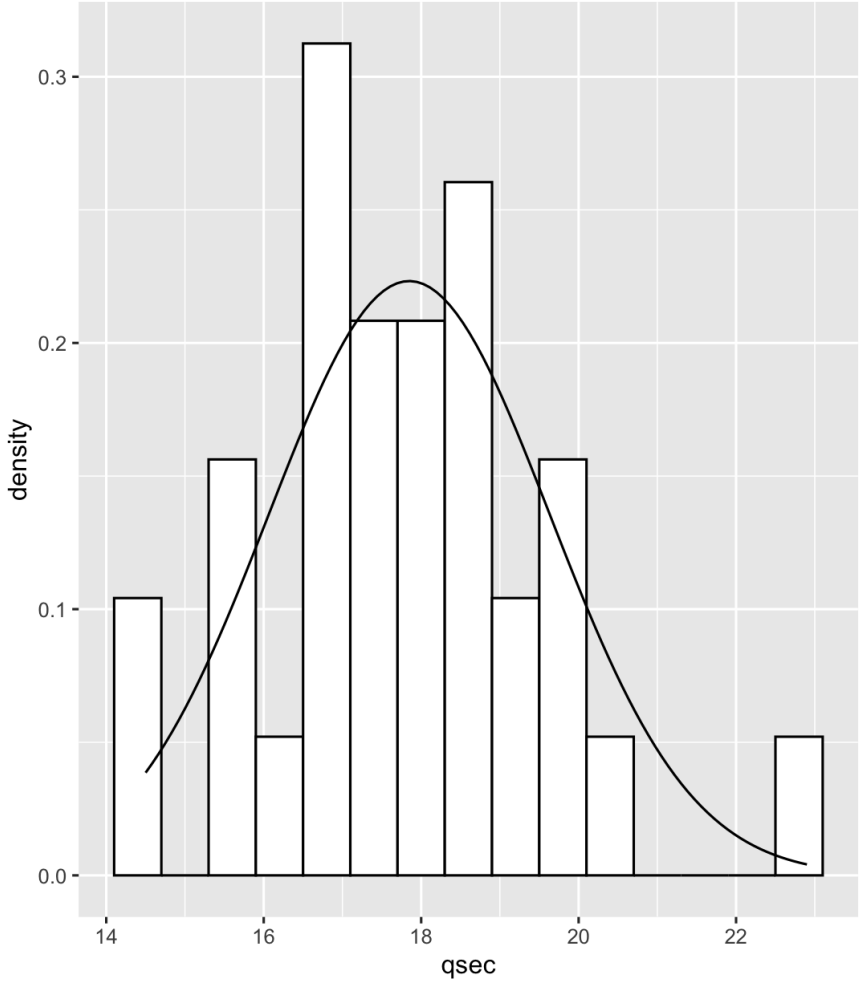
```
install.packages("gridExtra")
library(gridExtra)
grid.arrange(
ggplot(mtcars, aes(x = disp, y = mpg, col = factor(vs))) +
geom_point(),
ggplot(mtcars, aes(x = disp, y = mpg, size = qsec)) +
geom_point(),
nrow = 1)
```



Histogram with Normal Curve

Change histogram from counts to density and add normal curve

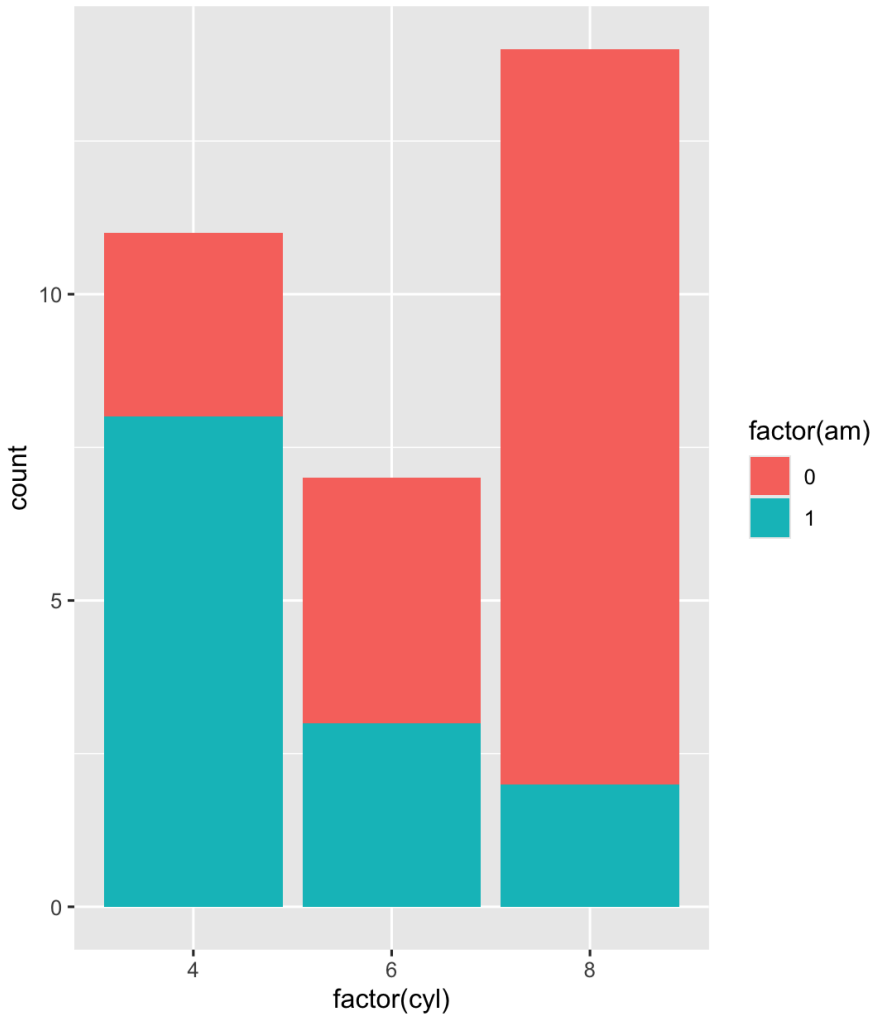
```
ggplot(mtcars, aes(qsec)) +
geom_histogram(aes(y = after_stat(density)), bins = 15, fill = "white", col= "black") +
stat_function(fun = dnorm, args = list(mean = mean(mtcars$qsec), sd = sd(mtcars$qsec)))
```



Barplots

The default barplot for two variables uses the aesthetic `fill` and position "stack"

```
ggplot(mtcars, aes(x = factor(cyl), fill = factor(am))) +
geom_bar()
```



ggplot2: More functions, examples and documentation

- Data visulaization with ggplot2 : : CHEATSHEET
- ggplot2 documentation
- ggplot2 functions documentation

Practicalities

- Deadlines:
  - R assignment part1 exercises: Friday June 13, 3 pm.
  - R assignment part2 exercises: Friday June 20, 3 pm.
  - **Quiz 2**: June 17, during Tuesday's Tutorial at 3:30 pm, duration: 30 minutes,
- Related complementary material:
  - Book chapters: Ch5, Ch6 of Introduction to R - tidyverse book.
  - Book chapters: Ch1-2 of ggplot2: Elegant Graphics for Data Analysis book
  - Introduction to dplyr
  - Video tutorials: Introduction to Tidyverse in R - dplyr
  - Video tutorials: data manipulation and visualization R
  - The tidyverse style guide