The Art and Science of Transportation Research in the AI Era

Introduction to R for Data Analysis (Hands-on Workshop)

M.Sc. Vijay Palliyil

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

M. Sc. Vijay Palliyil
palliyil@verkehr.tu-darmstadt.de

Learning Goals

#1 Running R code

#2 Packages & data import

#3 Wrangling data

#4 Visualising insights

#5 How to keep learning R and advance

Lecture Structure

#1	Orientation
#2	Rstudio & R basics
#3	Import and inspect data
#4	Wrangling with dplyr
#5	Visualisation with ggplot2
#6	Exporting, debugging and way forward

#1 Orientation

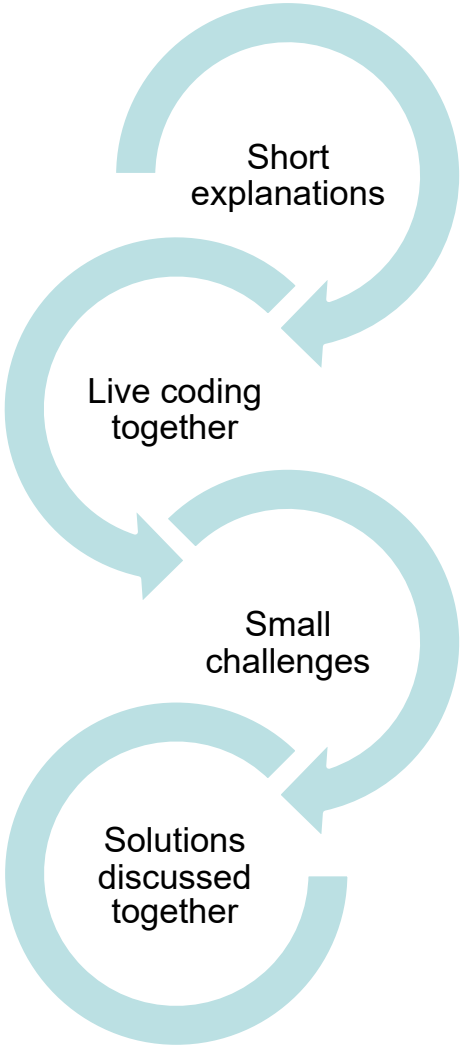
“R is a language for data analysis and graphics.”

- Ross Ihaka, co-creator of R

“Friends don’t let friends analyze data in Excel. They let them do it in R.”

- A wise data analyst

How this session will work:



Reasons to learn R

- Free and open source.

- Software for data science:
  - experiment/survey design
  - data retrieval
  - data wrangling
  - data analysis
  - reporting

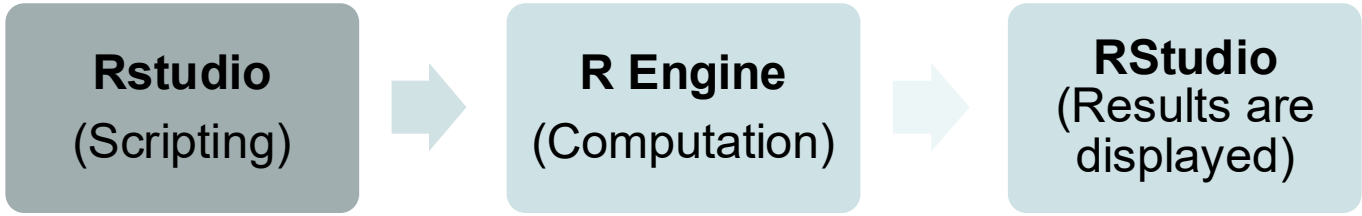
- A programming language, so we can use existing functions to code up our data science tasks or write new functions for customised/novel tasks

Tools we need installed:

R and **Rstudio**

What is R?

- A programming language
- A statistical computing engine
- Executes your code
- Produces results

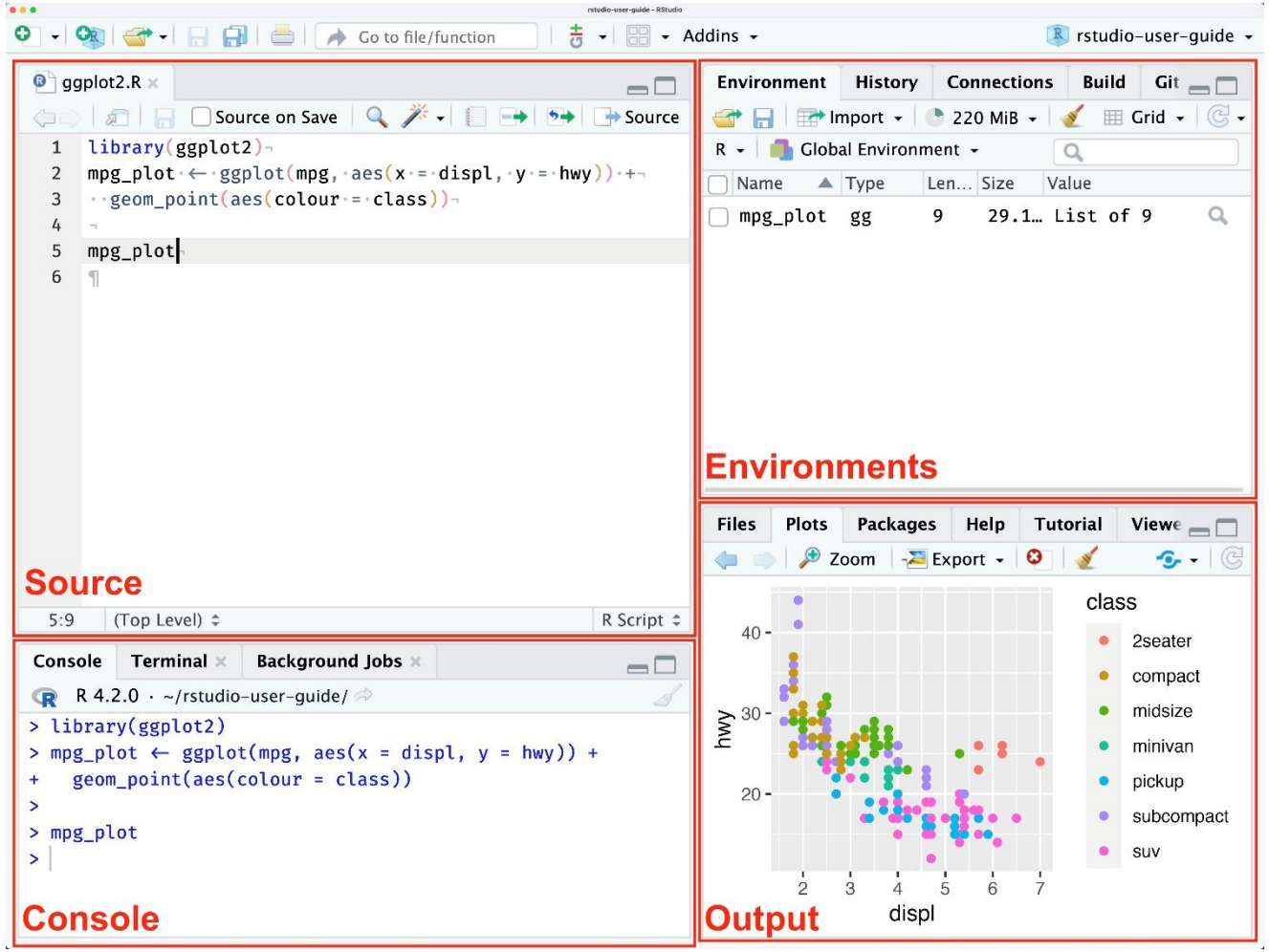


What is Rstudio?

- An IDE for R
- Makes R easier to use
- Editor, console, plots, help
- Does NOT replace R

RStudio IDE (Integrated Development Environment):

The **Source pane** is where you can edit and save R or Python scripts or author computational documents.



The **Environment pane** displays temporary R objects as created during that R session.

The **Console pane** is used to write short interactive R commands.

The **Output pane** displays the plots, tables, or HTML outputs of executed code along with files saved to disk.

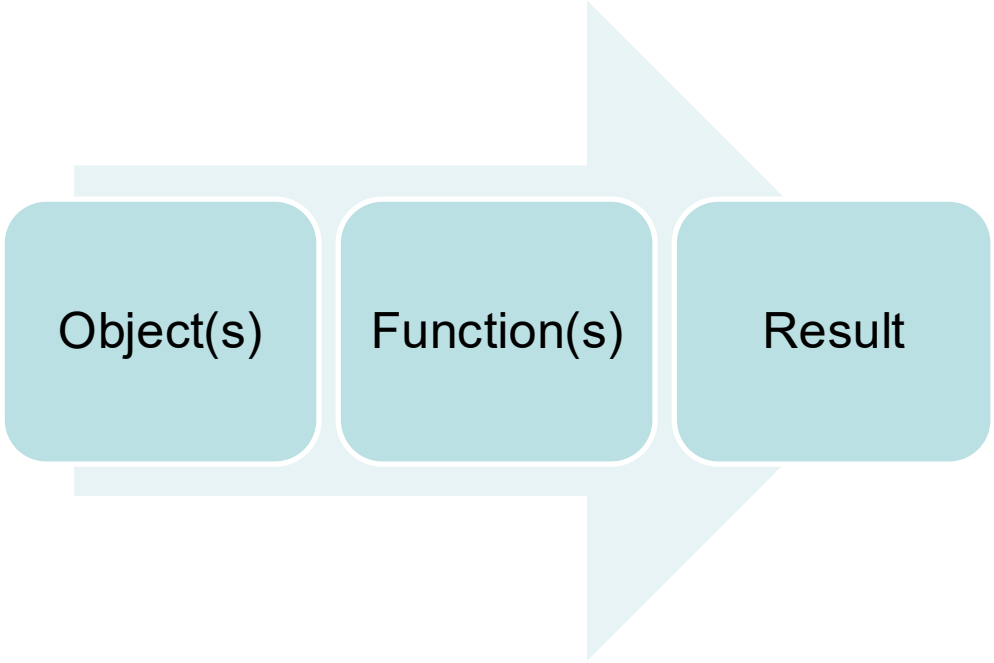
Image source: https://docs.posit.co/ide/user/ide/get-started/

How R works (conceptually):

R and **Rstudio**

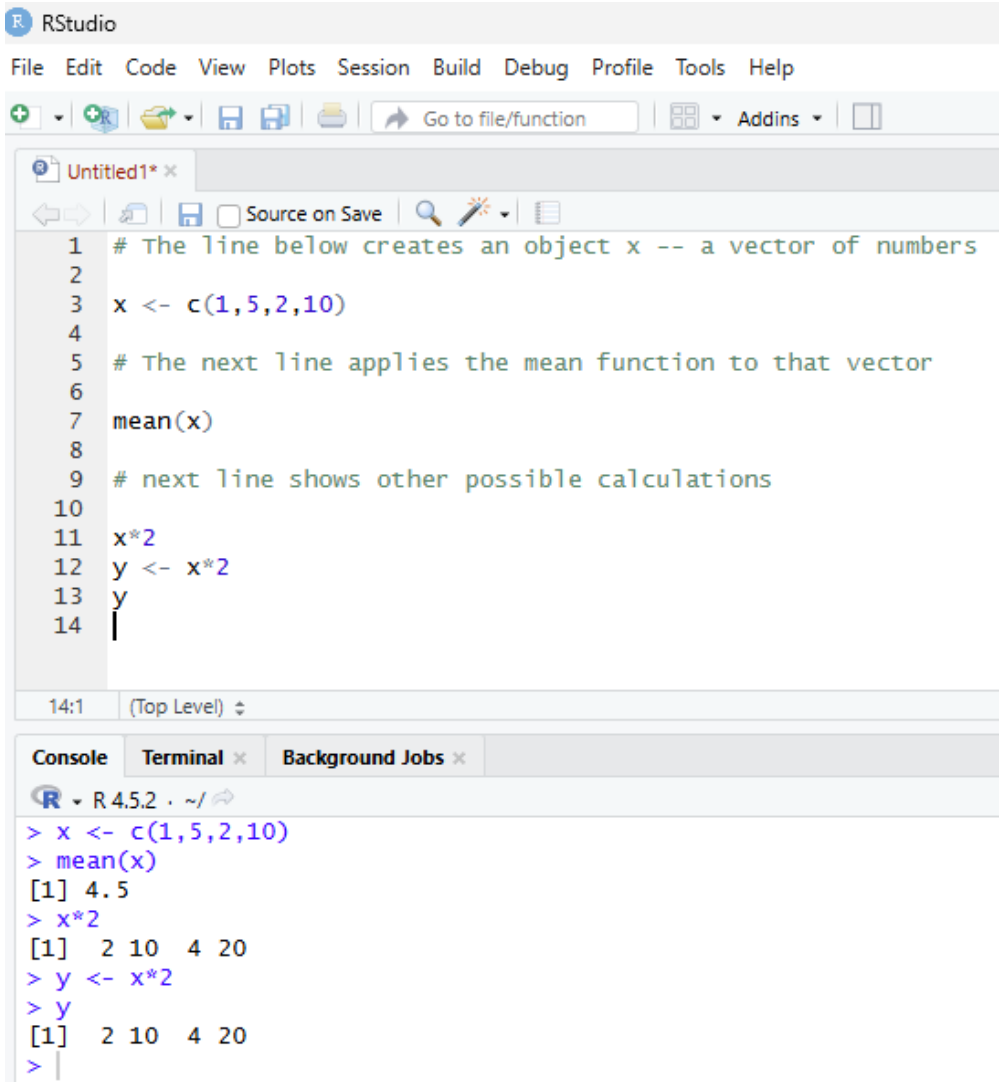
- You create **objects** (values, variables, tables, etc.)
- Objects have **names**
- **Functions** take objects as input
- Functions return new **objects** or **results**

- Functions in R are written to operate on whole vectors at once
- A **vector** in R is an ordered collection of values of the same type. A vector is R's way of storing many values under one name.
- Columns in data tables are vectors



First R code-along

- Create an object with the assign symbol **<-**
  - **<-** assigns
- Use functions with **()**
  - Example: **mean()**
- Run code line by line
  - Type into the script pane
  - **Ctrl + Enter** or **Cmd + Enter** to run the line
- R prints or returns results
- **R is case sensitive!**



Exercise

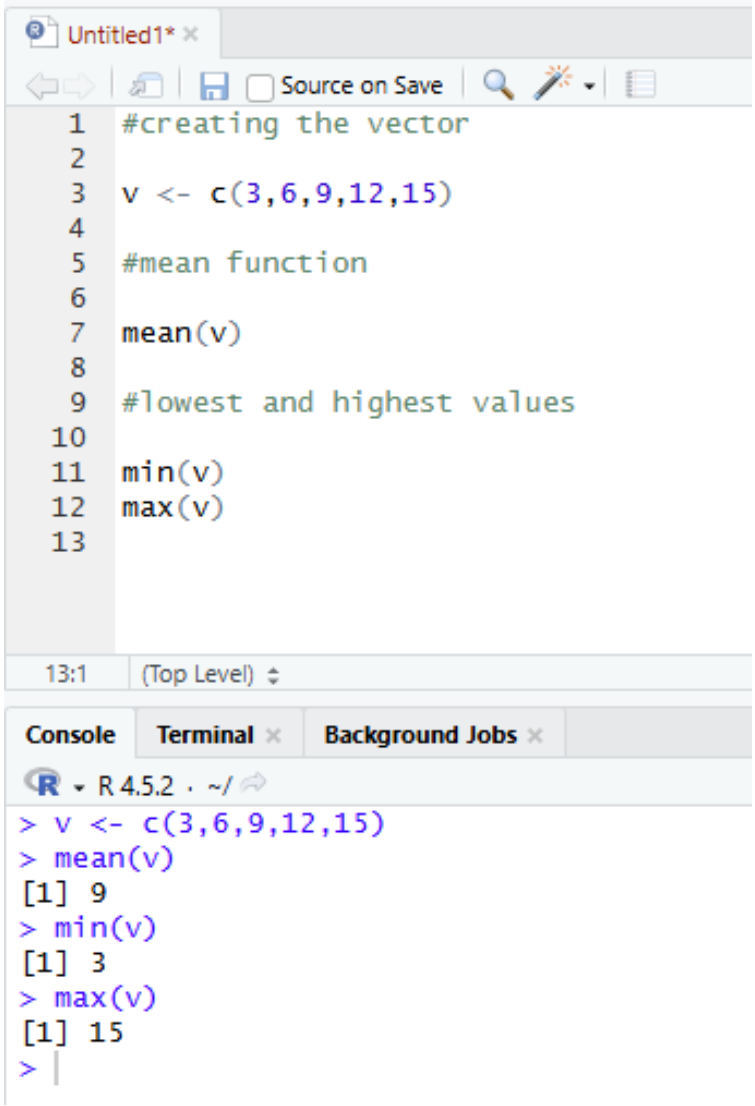
- Create a numeric vector with 5 values
- Compute mean, highest value, and lowest value
- Functions are **mean(), min(), max()**

Try this simple exercise

- Create a numeric vector with 5 values
- Compute mean, highest value, and lowest value
- Functions are **`mean(), min(), max()`**

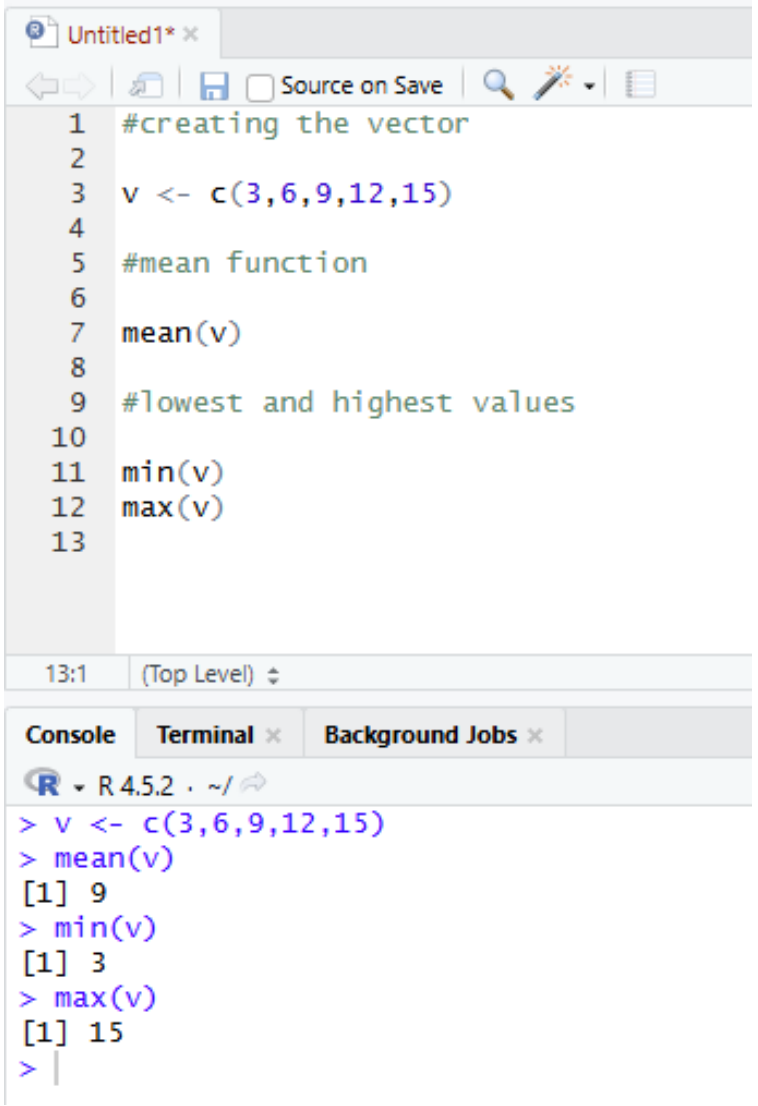
- **Solution:**

```
#creating the vector v

v <- c(3, 6, 9, 12, 15)

#using the mean, min and max functions

mean(v)
min(v)
max(v)
```



Scripts ensure Reproducibility of codes

- While working on code that you want to reuse -- always work from a script (source pane)
- **Save early, save often**
- **Comments should be used to label or describe the code**



What are packages?

- R comes with basic functionality
- R community develops packages
- Packages add new functions
- Package is a collection of functions
- Written for a specific goals
- We load packages when we need them

- Most packages come from CRAN (The Comprehensive R Archive Network)
- Some packages are also on GitHub
- To find the right packages, search the web with your objective..Eg:
  package for creating scatterplots

Installing packages

- Install = download it using R onto your PC
- Done once for a package
- Requires internet connection
- Packages usually come from CRAN
- Packages need to **loaded** for each session

- Install the packages now: dplyr and ggplot2
  - Install.packages(“dplyr”)
  - Install.packages(“ggplot2”)

- Install ≠ load...the function for loading is **library()**
  - Library(dplyr)
  - Library(ggplot2)

Working Directory (How R finds files)

- R is always working inside one assigned folder
- This folder is called the working directory

- To check it, use the function:
  - **`getwd()`**

- Function to list the files (R can access) in the folder:
  - **`list.files()`**

- If you open an Rstudio Project, the working directory is automatically the project folder

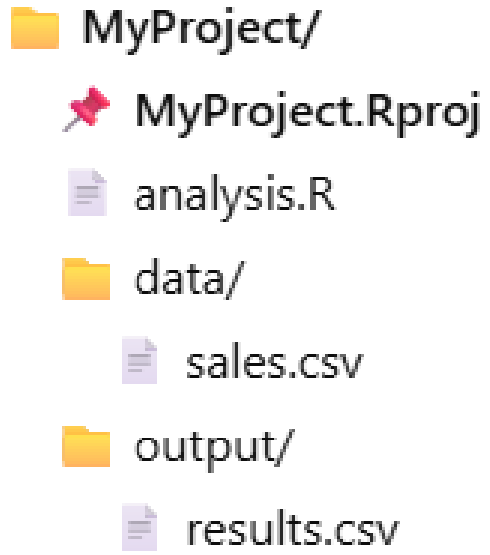
- Otherwise it is a default folder (documents, etc.)

RStudio Project (Solution to file path problems)

An Rstudio Project is a dedicated workspace folder.
When you use a project, Rstudio automatically:

- Sets the working directory
- Keeps scripts + data together
- Makes your work portable

Ideal Project Folder Structure



How data enters R

- Common source formats: CSV, Excel, Text files
- There are also built-in datasets (great for practice)

To import a CSV file we can use **read.csv()** function

Some functions to examine the imported data:

- **head()** function shows the first few rows of the dataset (quick preview)
- **names()** function shows the column names (what variables you have)
- **summary()** function gives a quick statistical summary of each column
- **str()** function shows the structure of the dataset – number of rows/columns + each columns **datatype** and example values

Code-along

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

- Lets use the built in dataset “mtcars” and try the functions to examine the data

```
data(mtcars)   # load dataset (optional)
head(mtcars)   # first rows
names(mtcars)  # column names
str(mtcars)    # structure + data types
summary(mtcars)# quick statistics
```

Output obtained using the function: head(mtcars)

```
> head(mtcars)
                   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
```

Your turn: Inspect the dataset

- Assign the dataset “iris” to a new object (hint: **df <- iris**)
- Use the data, head, names, str and summary function on this object

- How many rows and columns does the dataset have?
- What are the different variables (column names) in the dataset?

Solutions: Inspect the dataset

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

- Assign the dataset “iris” to a new object (hint: **df <- iris**)
- Use the head, names, str and summary function on this object

```
df <- iris

head(df)
names(df)
str(df)
summary(df)
dim(df)
```

- How many rows and columns does the dataset have? **150 rows, 5 columns**
- What are the different variables (column names) in the dataset? **Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species**

View the data

- The function view() opens a dataset in a table view (similar to excel)
- Useful for quickly checking rows and columns
- It does not change your data

- Try running these codes:

```
View(df)
View(iris)
View(mtcars)
```

- Additional tip: use ? To learn about a dataset or a function:

```
?mtcars
?iris
```

Missing values in datasets : (NA)

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

- **'NA'** means **'missing value'**
- When importing data, R converts blank values to NA automatically.

- **Try this in RStudio:**

```
x <- c(1, 2, NA, 4)

mean(x)
mean(x, na.rm = TRUE)
```

- **na.rm = TRUE** tells R to ignore missing values
- If even one value is missing, many calculations return NA unless you handle it. Eg: mean()
- But sometimes missing values are written differently (like N/A, ?, -) and then R does not recognise them.

What is data wrangling?

- **Data wrangling = preparing data for analysis**
- Typical tasks:
  - Selecting useful columns
  - Filtering rows
  - Creating new variables
  - Fixing missing values
  - Preparing data for plots and summaries

Pipes (%>%)

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

The pipe **%>%** passes the **output of one step** into the **next function**. So instead of writing nested functions, we can write it step by step.

So instead of nested functions like this:

```
select(filter(mtcars, cyl == 6), mpg, hp)
```

We use this:

```
mtcars %>%
  filter(cyl == 6) %>%
  select(mpg, hp)
```

Read it like: take mtcars, then filter, then select

💡 **Shortcut:**

in RStudio press **Ctrl + Shift + M** to insert **%>%**

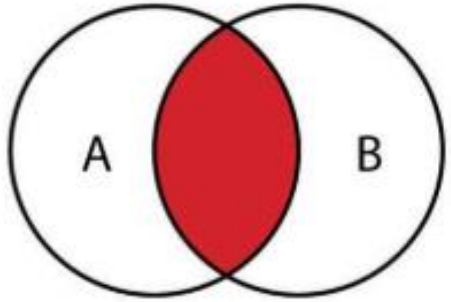
Filtering rows with `filter()` function

TECHNISCHE UNIVERSITÄT DARMSTADT

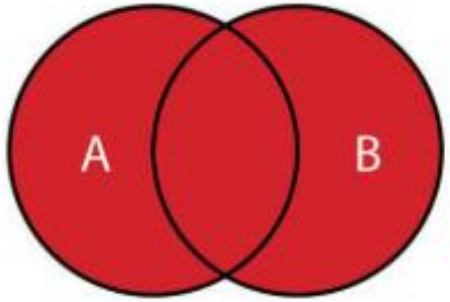
Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

filter() keeps only rows that match condition(s)

Try these examples in RStudio using the "iris" dataset on flowers:



AND



OR

- **One condition:** keep only flowers that are "setosa"

```
library(dplyr)
iris %>% filter(Species == "setosa")
```

- **AND (&) condition:** keep only flowers that are "setosa" & Sepal.Length > 5

```
iris %>% filter(Species == "setosa" &
Sepal.Length > 5)
```

- **OR (|) condition:** keep only flowers that are "setosa" OR have Sepal.Length > 7

```
iris %>% filter(Species == "setosa" |
Sepal.Length > 7)
```

Your turn: Filtering rows with `filter()` function

Using the built-in dataset “mtcars”, write R code to:

- **One condition:** Keep only cars with 6 cylinders (cyl == 6)
- **AND (&) condition:** Keep only cars with 8 cylinders (cyl == 8) AND fuel efficiency < 15 (mpg < 15)
- **OR (|) condition:** Keep cars with horsepower > 200 (hp > 200) OR weight > 4 (wt > 4)

💡 Hint: use filter() and ==, <, >

Solutions: Filtering rows with `filter()` function

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

- **One condition:** Keep only cars with 6 cylinders (cyl == 6)
- **AND (&) condition:** Keep only cars with 8 cylinders (cyl == 8) AND fuel efficiency < 15 (mpg < 15)
- **OR (|) condition:** Keep cars with horsepower > 200 (hp > 200) OR weight > 4 (wt > 4)

```
library(dplyr)

# 1) One condition: 6 cylinders
mtcars %>% filter(cyl == 6)

# 2) AND condition: 8 cylinders AND mpg < 15
mtcars %>% filter(cyl == 8 & mpg < 15)

# 3) OR condition: hp > 200 OR wt > 4
mtcars %>% filter(hp > 200 | wt > 4)
```

Selecting columns with `select()` function

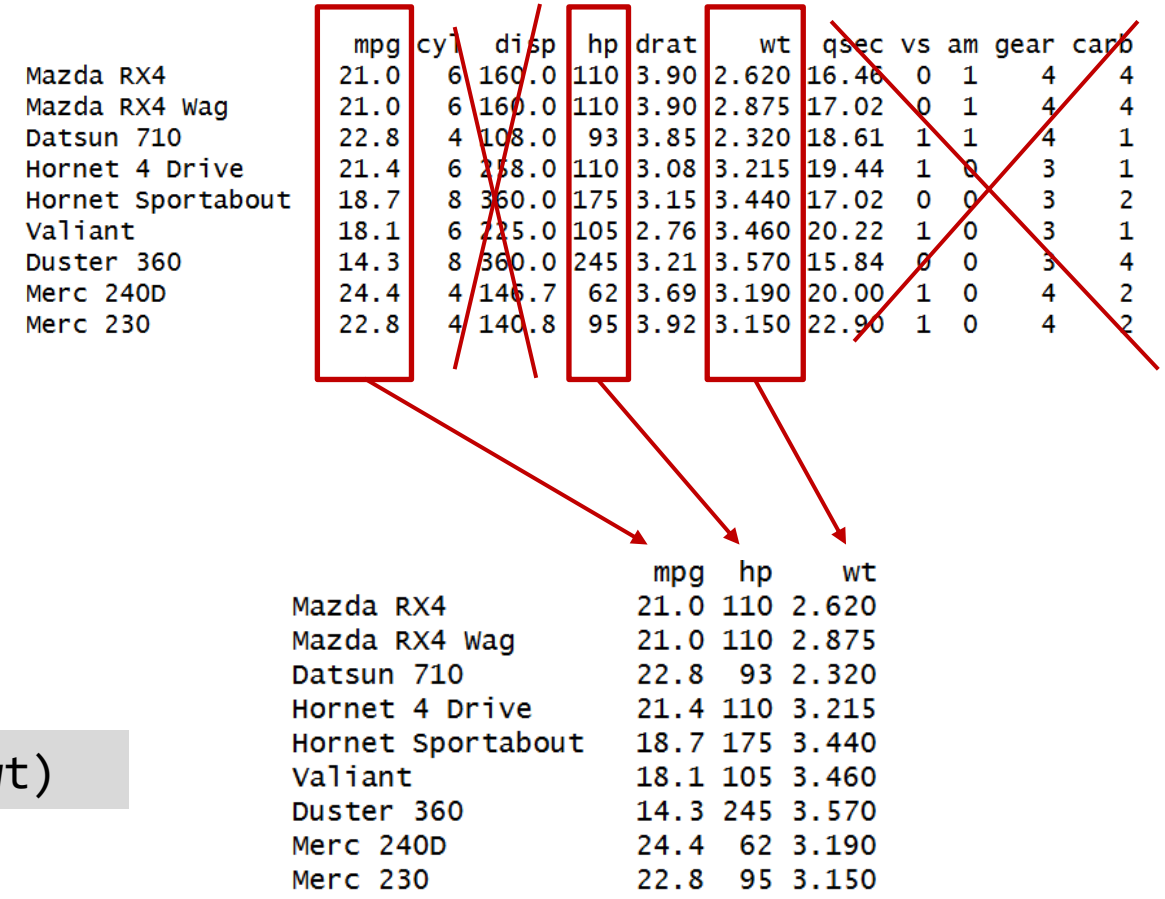
- select() keeps only the columns you choose (drops the rest)
- It returns a new data frame (table)

- *Try these example out:*

```
library(dplyr)

mtcars %>%
  select(mpg, hp, wt)
```

- *You may also save it as a new object:*

```
small_df <- mtcars %>% select(mpg, hp, wt)
```



Your turn: Selecting columns

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

- Use the dataset “iris”
- Select any 2 variables/columns out of the available ones (eg: Petal.Length, Petal.Width, Species, etc.)
- Save the result into a new object called iris_small

- *Hint:*

```
library(dplyr)

iris_small <- iris %>%
  select(...)
```

- *Then check your result:*

```
head(iris_small)
```

Creating new variables with `mutate()` function

- mutate() creates new columns (new variables) in a dataset
- Can also modify existing columns (by overwriting them)

- *Try this example out in RStudio:*

```
library(dplyr)

iris_new <- iris %>%
  mutate(petal_ratio = Petal.Length /
Petal.Width)
```

- *Check the result:*

```
head(iris_new)
```

Original dataset:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa

New dataset with an added column:

```
> iris_new
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	petal_ratio
1	5.1	3.5	1.4	0.2	setosa	7.000000
2	4.9	3.0	1.4	0.2	setosa	7.000000
3	4.7	3.2	1.3	0.2	setosa	6.500000
4	4.6	3.1	1.5	0.2	setosa	7.500000
5	5.0	3.6	1.4	0.2	setosa	7.000000
6	5.4	3.9	1.7	0.4	setosa	4.250000
7	4.6	3.4	1.4	0.3	setosa	4.666667
8	5.0	3.4	1.5	0.2	setosa	7.500000
9	4.4	2.9	1.4	0.2	setosa	7.000000

Your turn: Create a new variable with `mutate()` function

- Using the dataset iris, create a new dataset called iris_new2
- Add a new column called sepal_ratio
- sepal_ratio = Sepal.Length / Sepal.Width

- Hint:

```
Library(dplyr)

iris_new2 <- iris %>%
  mutate(sepal_ratio = ...)
```

- Then check your result

```
head(iris_new2)
```

Solution: Create a new variable with `mutate()` function

- A new column is added to the dataset
- Calculated row by row (for every row)

- *Solution:*

```
library(dplyr)

iris_new2 <- iris %>%
  mutate(sepal_ratio = Sepal.Length /
Sepal.Width)

head(iris_new2)
```

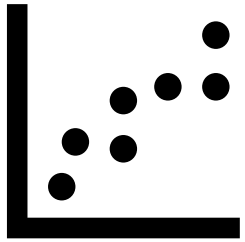
- *Optional quick check:*

```
summary(iris_new2$sepal_ratio)
```

The **$ operator** is used to access **one column (variable)** inside a data frame. So the code here means “Give me the sepal_ratio column from the dataset iris_new2”

Visualizations (using R)

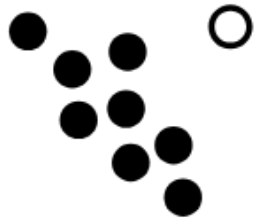
- Visualization helps us quickly see:



Patterns
(trends, relationships)

Comparisons
(groups, categories)

Insights
(what matters in the data)



Outliers
(unusual values)

- We're going to use **ggplot2**, the most popular plotting package in R.

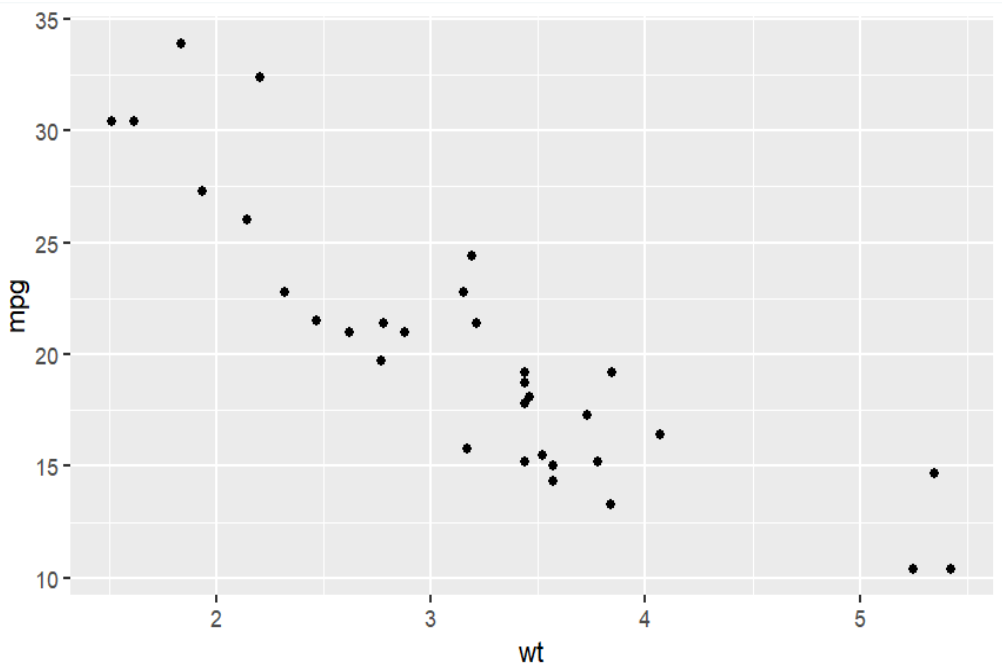
First plot in ggplot2: Scatter plot

- Goal: visualize the relationship between two numeric variables
- In a scatter plot each point is one observation
- In mtcars, each point would represent a car

Ggplot2 recipe:

1. Choose dataset
2. Map variables to axes -- `aes()`
3. Choose a geometry (plot type)

```
library(ggplot2)

ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()
```

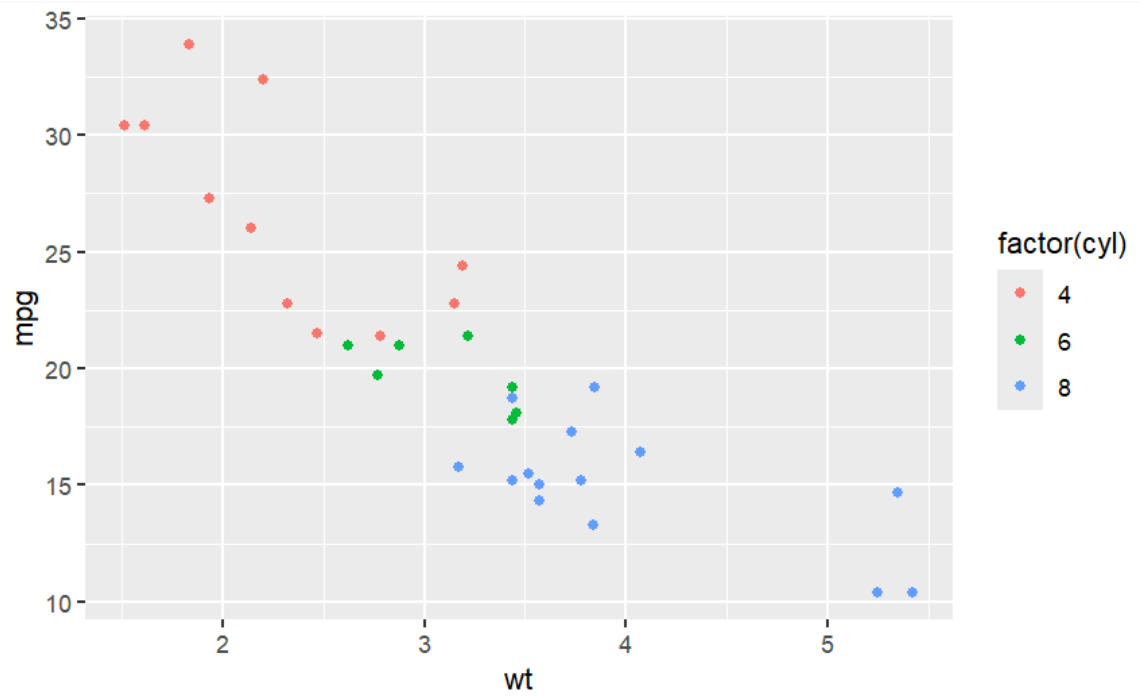


Color encodes information (not decoration)

- Color can represent a **variable** in plots
- This helps us **compare groups** in the same plot
- We usually colour by a **category** (eg: number of cylinders)
- In mtcars, cyl (cylinders) is numeric, but we treat it as a category

- Try this code in RStudio:

```
mtcars %>%

  ggplot(aes(x = wt, y = mpg, color =
factor(cyl))) +
  geom_point()
```



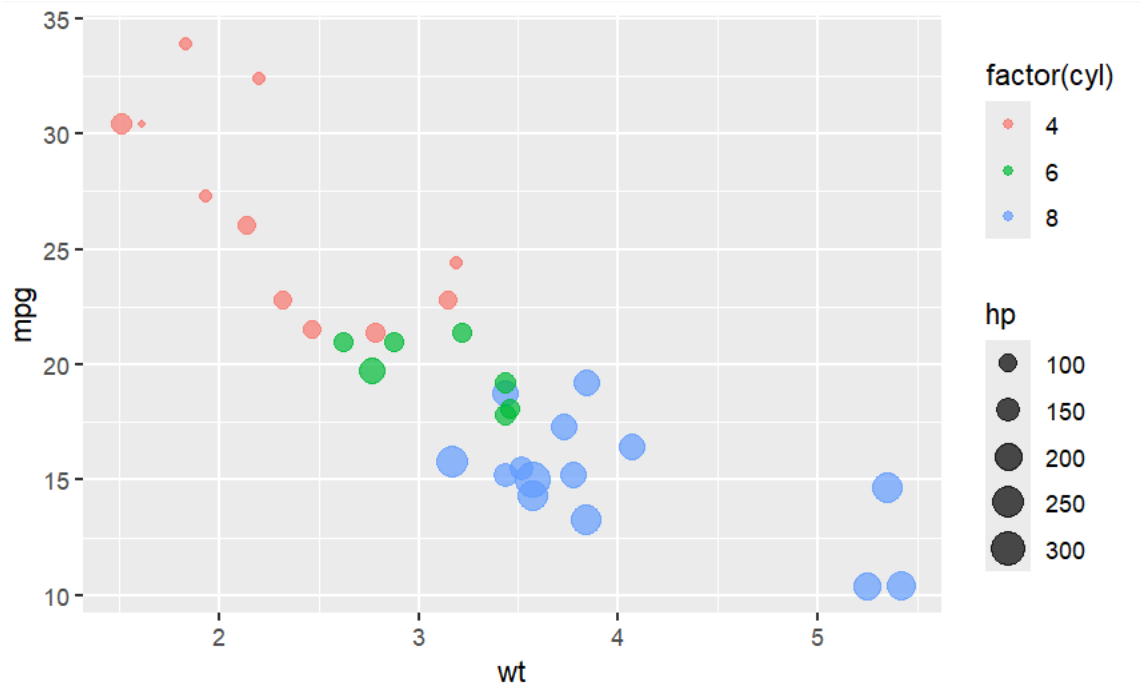
The plot now shows differences between **4, 6, and 8 cylinder cars**

More information on the same plot (colour + size)

- In addition to colour, we can use size to visualize another extra variable in a scatter plot
- Colour – categorical variable
- Size – numeric variable (eg: weight, horsepower, fuel efficiency)

- Try this code in RStudio:

```
mtcars %>%
  ggplot(aes(x = wt, y = mpg,
             color = factor(cyl),
             size = hp)) +
  geom_point(alpha = 0.7)
```



We added cylinders (colour) + horsepower (size)

Your turn: Scatter plot (colour + size)

- Use mtcars to create a scatter plot
  - x-axis: wt
  - y-axis: mpg
- Add 2 extra variables
  - One for colour (categorical)
  - One for size (numeric)

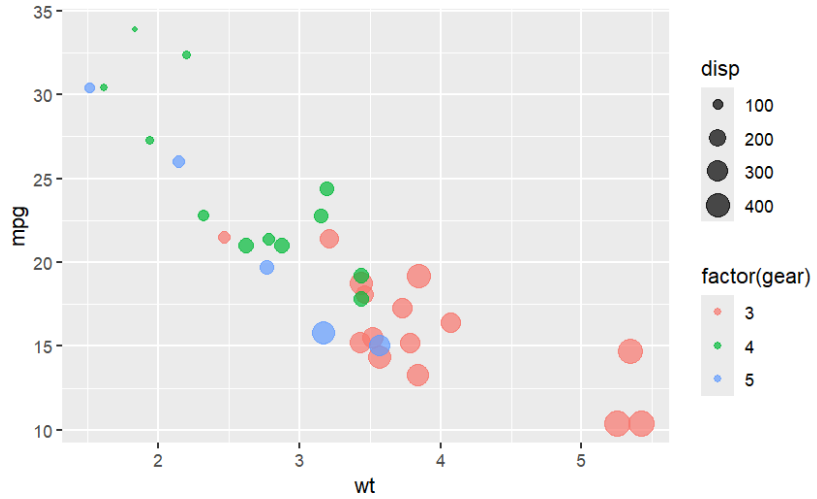
- Choose options you have not used yet. E.g.:
  - colour = factor(gear) (number of gears)
  - colour = factor(am) (0 = automatic, 1 = manual)
  - size = qsec (1/4 mile time)
  - size = disp (engine displacement)

```
mtcars %>%
  ggplot(aes(x = wt, y = mpg,
             color = factor(gear),
             size = disp)) +
  geom_point(alpha = 0.7)
```

Solution: Scatter plot (colour + size)

- There many correct answers
- Two example solutions:

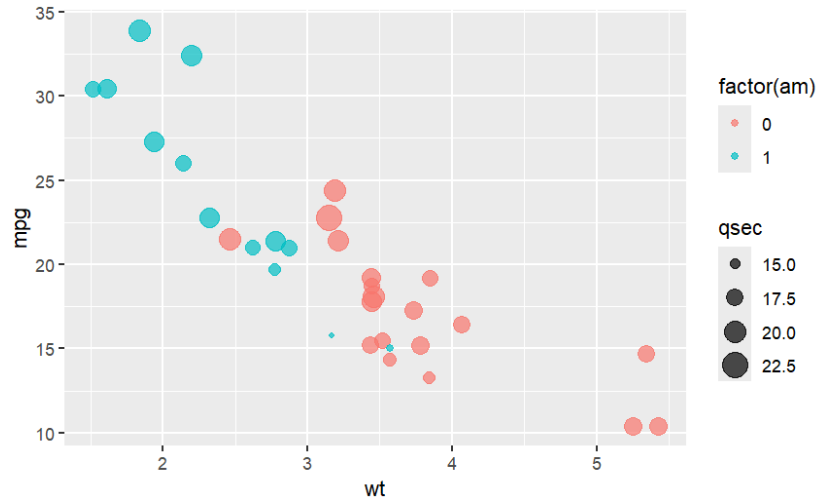
```
mtcars %>%
  ggplot(aes(x = wt, y = mpg,
             color = factor(gear),
             size = disp)) +
  geom_point(alpha = 0.7)
```



- You can swap the variables:
  - **Color:** gear or am
  - **Size:** disp or qsec

The "**+**" is important!!

```
mtcars %>%
  ggplot(aes(x = wt, y = mpg,
             color = factor(am),
             size = qsec)) +
  geom_point(alpha = 0.7)
```



Saving results

- We often want to save a table from R to a file
- The easiest format is CSV
- Use write.csv() to export a data frame

- Example (save the first 10 rows):

```
result <- mtcars %>%
  select(mpg, wt, hp) %>%
  head(10)

write.csv(result, "mtcars_sample.csv",
row.names = FALSE)
```

- The file will be saved in your working directory (check the Files pane)

Saving plots

- After creating a plot, we often want to save it as an image
- `ggsave()` saves the most recent plot
- Common formats:
  - PNG (best for slides + web)
  - PDF (best for reports)

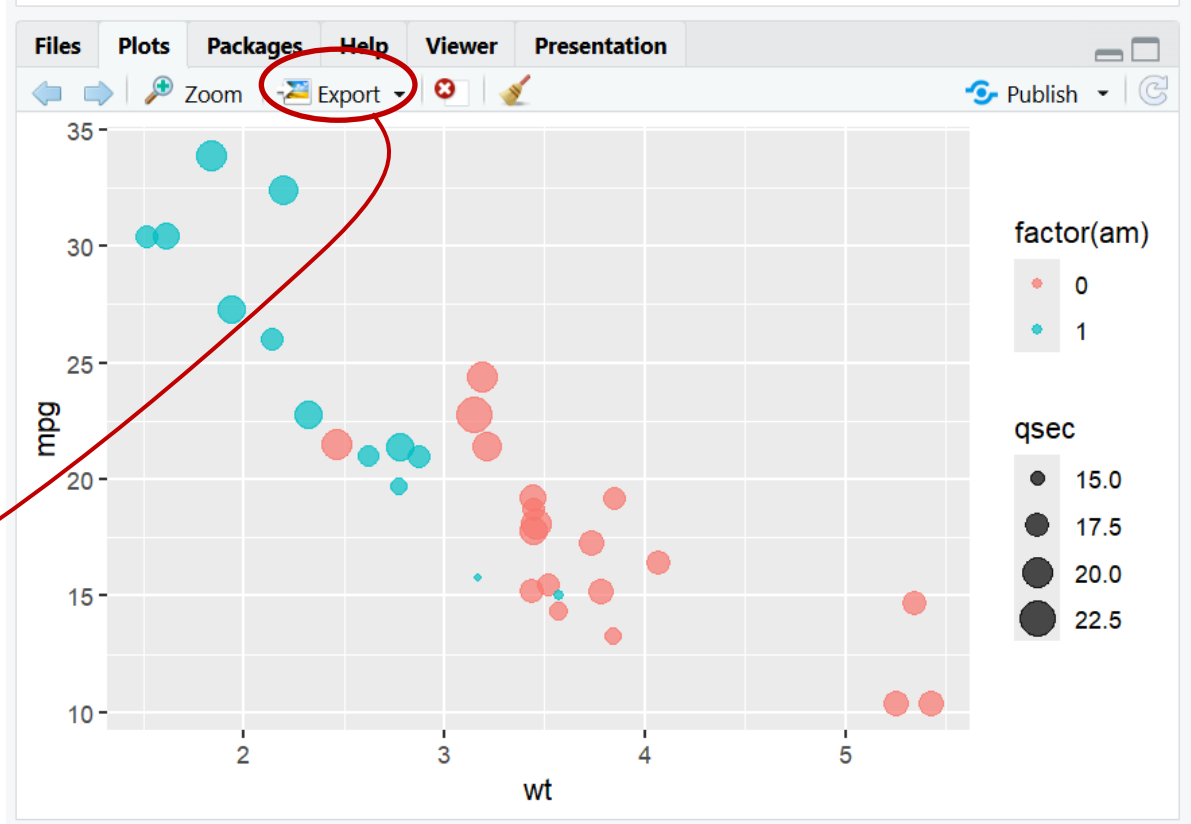
- Save the last plot you created using ggsave():

```
ggsave("my_plot.png", width = 6, height = 4)
```

- The file is saved in your working directory

OR (manual export):

- In the **Plots** pane:
- you can click **Export**
- Then **Save as Image / Save as PDF**



When things dont work: Debugging checklist

- Read the error message
- Check if you loaded the package in use:

```
library(dplyr)
library(ggplot2)
```

- Check if the object exists. This function lists the names of everything you created like vectors, data frames

```
ls()
```

- Inspect the data

```
head(df)
str(df)
names(df)
```

- Check the function help

```
?filter
?ggplot
```

- Search the exact search message online: copy and paste the full error message into Google

Resources for learning and help

- In **RStudio** (fastest help):
  - ?Function_name → e.g.: ?mean → opens documentation for a function
  - ??plot → searches help pages by keyword

- Cheat sheets (quick reference)
  - **Posit** “Data Transformation” (dplyr)
  - **Posit** “Data Visualization” (ggplot2)

- Learn by **examples and tutorials**
  - Search: “ggplot scatter plot examples”
  - Search: “dplyr filter select mutate examples”

- Community help (when you’re stuck)
  - **Stack Overflow**
  - **RStudio Community**

Final message

TECHNISCHE UNIVERSITÄT DARMSTADT
IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

You don't need to know everything. You need to know how to find and use what you need.

- **THANK YOU**
- **DANKE**