

# INTRODUCTION TO R AND POSITRON

This PDF version is not interactive. For interactivity, use the [html version](#) instead

Carsten Lange

[clange@cpp.edu](mailto:clange@cpp.edu)

Cal Poly, Pomona

# LEARNING OUTCOMES

## Part 1: Setup

- How to install R and *Positron*
- What is the windows layout of *Positron*
- How to use a (project) folder in *Positron*
- How to extend R's functionality with R-packages and which packages to install

# LEARNING OUTCOMES

## Part 2: How R Stores the Data (plus how it presents very big/small numbers)

- Data types
- Data objects in R
- How does R presents very big/small numbers

# LEARNING OUTCOMES

## Part 3: The **tidyverse** Package:

- The Structure of R commands
- About the **tidyverse** package for data frames
  - **select()** and rename columns (variables)
  - **filter()** rows (observations)
  - **mutate()** (define columns (variables); overwrite old or create new)
  - **arrange()** sort observations in a data frame.
  - piping (connecting commands) with **|>**.

# PART 1: INSTALL AND SETUP R AND POSITRON

A typical setup to work with R consists of two components:

- the **R Console** which executes R code and
- an integrated development environment (**IDE**) such as **RStudio** or **Positron**.

You can download R here: [Download R](#)

You can download *Positron* here: [Download Positron](#)

## Note

- Install R before *Positron*
- If an older R version exists uninstall it before installing the newer version

# RSTUDIO – INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

(DOES NOT HAVE NEWEST FEATURES OF A MODERN IDE)

The screenshot displays the RStudio interface with three main panels:

- Source Editor (left):** Contains R code for a Venn diagram. A red watermark "Write Code Window (text only)" is overlaid on this panel.
- Environment (top right):** Lists loaded objects in the Global Environment. A red watermark "Info: Variables & Objects" is overlaid on this panel.
- Viewer (bottom right):** Displays a Venn diagram titled "Churn = TRUE". The diagram shows the overlap of three variables: TenureLow (blue), ChargesHigh (yellow), and SeniorYes (green). A red watermark "View, Plot & Help" is overlaid on this panel.
- Console (bottom left):** Shows the execution of the R code. A red watermark "Run Code & See Results Window" is overlaid on this panel.

**Venn Diagram Data:**

Region	Count	Percentage
TenureLow only	464	24.8%
ChargesHigh only	261	14.0%
SeniorYes only	65	3.5%
TenureLow & ChargesHigh	603	32.3%
TenureLow & SeniorYes	86	4.6%
ChargesHigh & SeniorYes	132	7.1%
TenureLow, ChargesHigh & SeniorYes	241	12.9%
None	17	0.9%

RStudio Window

Video for *First Steps* to setup R and RStudio: [Click here](#)

(However, **it is recommended** to work with *Positron* rather than RStudio)

Textbook

# POSITRON – INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

The screenshot displays the Positron IDE interface with three main panels:

- Code Editor (Top Left):** Contains R code for a Venn diagram analysis. A red watermark "Write Code Window (text only)" is overlaid on this panel.
- Console (Bottom Left):** Shows the execution of the R code. A red watermark "Run Code & See Results Window" is overlaid on this panel.
- Viewer (Right):** Displays the results of the R code, including a Venn diagram plot. A red watermark "View, Plot & Help" is overlaid on this panel.

**Code Editor Content:**

```
VennDiagram > VennDiagram.qmd > What Will You Learn > (code cell)
24
25 ## What Will You Learn {.scrollable .smaller}
26
27
28
29 - Preparing Venn-Diagram Churn Analysis
30
31 ▶ Run Cell | Run Next Cell
32 {r}
33 library(rio); library(janitor); library(tidyml)
34 DataChurnOrg=import("https://ai.lange-analytic.com/data/aiCoData.csv") |>
35   clean_names("upper_camel") |>
36   select(Churn,Gender, SeniorCitizen, Tenure, MonthlyCharges) |>
37   mutate(Churn=ifelse(Churn=="Yes",TRUE,FALSE)) |>
38   mutate(SexMale=ifelse(Gender=="Male",TRUE,FALSE))|>
39   mutate(SeniorYes=ifelse(SeniorCitizen==1,TRUE,FALSE))|>
40   mutate(ChargesHigh=ifelse(MonthlyCharges>70.35,TRUE,FALSE))|>
41   mutate(TenureLow=ifelse(Tenure<29,TRUE,FALSE)) |>
```

**Console Content:**

```
c:/CalPoly/EC4990AINew/AIBookSlides
+ geom_venn()+
+ coord_fixed()+
+ theme_void() +
+ ggtitle("Churn = TRUE")
+ plot(PlotYes)
Warning message:
package 'ggvenn' was built under R version 4.5.2
> PlotNo=ggplot(aes(A=TenureLow, B=ChargesHigh, C=SeniorYes),data=DataChurnNo) +
+ geom_venn()+
+ coord_fixed()+
+ theme_void()+
+ ggtitle("Churn = FALSE")
+ plot(PlotNo)
>
```

**Viewer Content:**

**Info: Variables & Objects**

DATA	VALUES
DataChurnNo	[5174 rows x 9 columns] <data.frame>
DataChurnOrg	[7043 rows x 9 columns] <data.frame>
DataChurnYes	[1869 rows x 9 columns] <data.frame>
ModelLogit	[coefficients = -1.791773986058575 -0.00167... glm
PlotNo	?? ggplot2::ggplot
PlotYes	?? ggplot2::ggplot

**Plots:**

Churn = TRUE

Region	Count	Percentage
TenureLow only	1415	27.3%
ChargesHigh only	1332	25.7%
SeniorYes only	1286	24.9%
TenureLow & ChargesHigh	475	9.2%
TenureLow & SeniorYes	91	1.8%
ChargesHigh & SeniorYes	349	6.7%
All three	122	2.4%
<b>Total</b>	<b>1286</b>	<b>24.9%</b>

Positron Window

First steps to setup R and Posit can be found in this video: coming soon

Textbook

# SET WINDOW LAYOUT AND CHOOSE R INTERPRETER

The screenshot shows the RStudio interface with the 'Customize Layout' dialog box open. The dialog box is titled 'Customize Layout' and has a 'Layout Presets' section. The 'Stacked Layout' option is selected and highlighted with a red arrow. Other options include 'Side-By-Side Layout', 'Notebook Layout', and 'Assistant Layout'. There are also sections for 'Visibility', 'Primary Side Bar Position', and 'Panel Alignment' with various checkboxes and keyboard shortcuts.

In the background, the RStudio window shows the R 4.5.1 interpreter selected in the top right corner, also highlighted with a red arrow. The console output shows the R startup message:

```
R 4.5.1 started.  
  
R version 4.5.1 (2025-06-13 ucrt) -- "Great Square Root"  
Copyright (C) 2025 The R Foundation for Statistical Computing  
Platform: x86_64-w64-mingw32/x64  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
>
```



# RSTUDIO KEYBINDINGS (NEEDS TO DONE ONLY ONCE)

Click Gear Icon -> Choose: Settings -> Search for: Keybindings -> Toggle-On RStudio Keybindings

The screenshot shows the RStudio Settings window with the search bar containing 'keybindings'. The search results are filtered to show 6 settings. The 'Workbench > Keybindings: Rstudio Keybindings' setting is highlighted, and its checkbox is checked. A red circle highlights the 'keybindings' search term, and a red line connects it to the checked checkbox. Another red circle highlights the gear icon in the bottom-left corner of the RStudio interface, with a red line connecting it to the 'Settings' option in the dropdown menu. The 'Settings' dropdown menu is open, showing options like 'Command Palette...', 'Profiles', 'Settings', 'Extensions', 'Keyboard Shortcuts', 'Snippets', 'Tasks', 'Themes', and 'Check for Updates...'. The 'Settings' option is highlighted in blue. The main settings window shows the following settings:

- Workbench > Keybindings: Rstudio Keybindings**
  - Enable RStudio keybindings (requires restart)
- Workbench > Settings: Open Default Keybindings**
  - Controls whether opening keybinding settings also opens an editor showing all default keybindings.
- Accessibility > Verbosity: Keybindings Editor**
  - Provide information about how to change a keybinding in the keybindings editor when a row is focused.
- Terminal > Integrated: Send Keybindings To Shell**

The bottom of the screenshot shows the terminal window with the following text:

```
You are welcome to redistribute it under certain conditions.
distribution details.
g in an English locale
y contributors.
ation and
ckages in publications.
' for on-line help, or
terface to help.
```

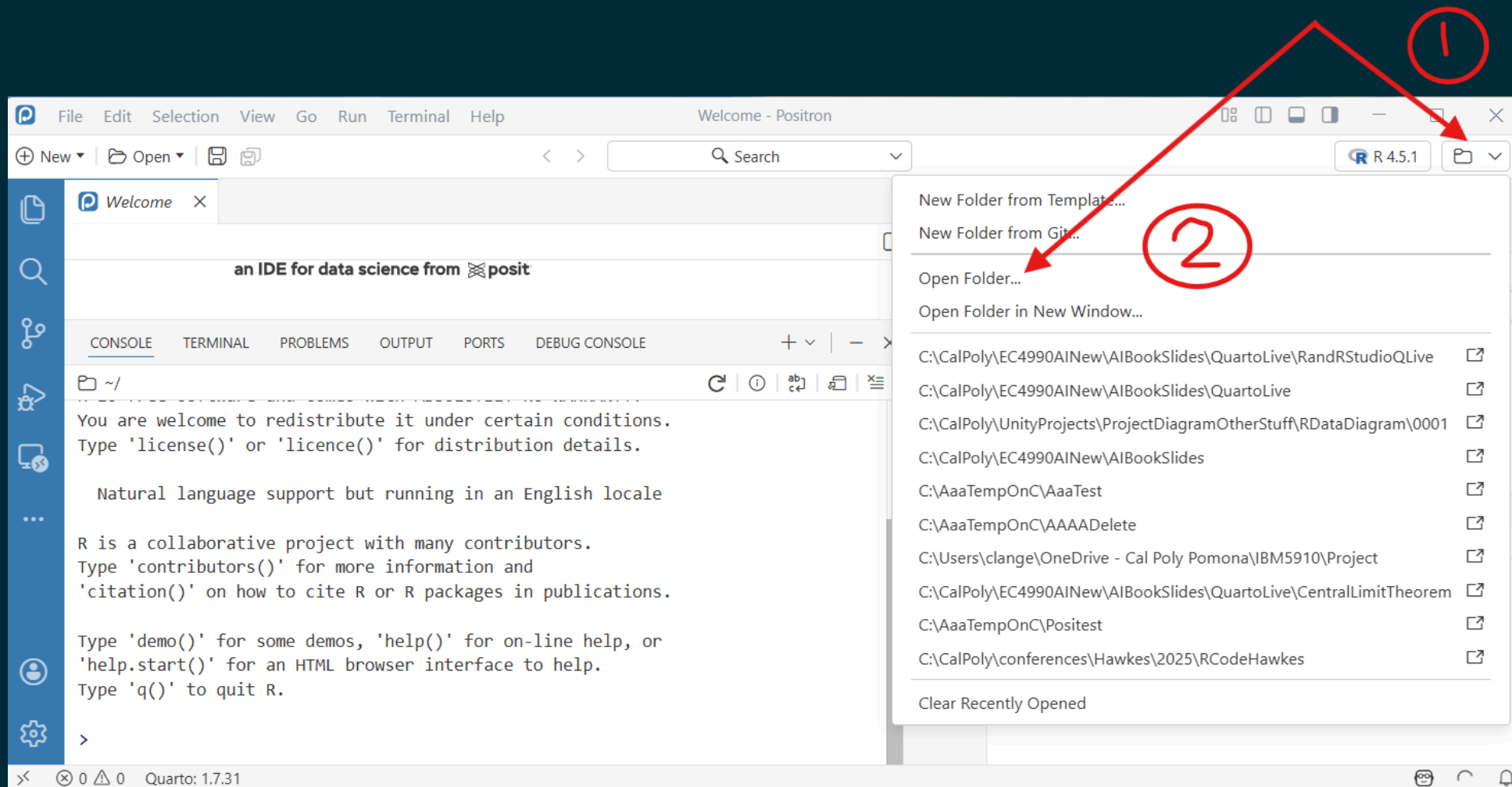
The bottom of the screenshot shows the system tray with the text 'Textbook' and 'Quarto: 1.7.31'.



# ALWAYS WORK WITH FOLDERS

## Always open or create a folder first

The folder is the one where all your R (`.r`), Quarto (`.qmd`), and data (e.g., `.csv`) files are stored



# USE COMMAND PALETTE TO CREATE A NEW R OR Quarto FILE

CTRL SHIFT P (Windows) or ⌘ SHIFT P (Mac) to open **Command Palette**:

1. Type either “New Quarto Document” or “New R Document” into search bar
2. Click and Create file
3. Save file right away

# OPEN A FILE FROM AN EXISTING POSITRON FOLDER

The screenshot displays the Positron IDE interface. The Explorer pane on the left shows a project structure under 'AIBOOKSLIDES'. The file 'RandRstudioQLive.qmd' is selected and highlighted in blue. A red arrow points to the Explorer icon in the top-left corner, and another red arrow points to the selected file. The main editor shows the content of 'RandRstudioQLive.qmd', which includes a title, author information, and execution settings. The terminal at the bottom shows the R prompt and the start of the R session.

```
1 ---
2 title: "Introduction to R and Positron "
3 author:
4   name: "Carsten Lange"
5   email: "clange@cpp.edu"
6   affiliation: "Cal Poly, Pomona"
7
8 execute:
9   message: false
10  warning: false
11
12 format:
13   live-revealjs:
14     theme: [moon, ../../CustomCL.scss]
15   warning: false
16   controls: true
17   chalkboard:
18     theme: whiteboard
```

CONSOLE

```
c:/CalPoly/EC4990AI/New/AIBookSlides
>
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

# POSITRON: FIRST STEPS WITH AN R FILE (.r)

- Remember, always open a folder first!

## AN R FILE SUCH AS `MyFile.r` CONTAINS ONLY R-CODE

1. Print “Hello world” using the `print()` command.
  2. Assign values 3 and 4 to the legs `a` and `b` of a right-angled triangle.
- calculate the hypotenuse `c`:

$$c^2 = a^2 + b^2 \iff c = \sqrt{a^2 + b^2}$$

- print the result using the `cat()` command
- Assign the number 2 to the variable (“R” objects) `a` and run the `cat()` command again.

# TRY POSITRON WITH A QUARTO FILE ( `.qmd` )

- Remember, always open a folder first!

## A QUARTO FILE SUCH AS `MyFile.qmd` CONTAINS TEXT AS WELL AS R-CODE

- Text is written in *Markdown*
- Code is surrounded by:

```
```${r}
```

```
MyCode goes here
```

```
```
```

# TRY POSITRON WITH A QUARTO FILE ( `.qmd` )

## Instructions:

1. Print “Hello world” using the `print()` command.
2. Assign values 3 and 4 to the legs `a` and `b` of an right-angled triangle.
  - calculate the hypotenuse `c`:

$$c^2 = a^2 + b^2 \iff c = \sqrt{a^2 + b^2}$$

- print the result using the `cat()` command

Note, now we want everything nicely commented!



# WASM: R RUNS IN A BROWSER INCLUDING LIBRARIES AND DATA

- Print “Hello world” using the `print()` command.
- Assign values 3 and 4 to the legs `a` and `b` of a right-angled triangle.
- Calculate the hypotenuse `c`

$$c^2 = a^2 + b^2 \iff c = \sqrt{a^2 + b^2}$$

- Print the result using the `cat()` command
- Assign the number 2 to the variable (“R” object) `a` and run the `cat()` command again.

R Code

↻ Start Over

▶ Run Code

```
1 print(...)  
2 a=3  
3 b=4  
4 c= ...  
5 cat(...)
```

# R PACKAGES

R Packages extend R's functionality. They have to be **installed** only once:

For example, to install the `tidyverse` package, type in the consol window:

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

Needs to be only done once!

After installation packages they need to be **loaded** in every new R script or Quarto file with:

```
library()
```

Packages frequently used in this course (**please install soon**):

- `tidyverse`: supports easy data processing .
- `rio`: allows loading various data resources with one `import()` command from the user's hard drive or the Internet.
- `janitor`: provides functionality to clean data and rename variable names to avoid spaces and special characters.

# VIDEOS FOR THE **rio** AND THE **tidyverse** PACKAGE

Example: How to install the tidyverse package: [Click here](#)

Video about the rio package: [Click here](#)

# USING THE `rio` AND THE `tidyverse` PACKAGE

Example: `rio` and `tidyverse` package (assuming they are installed already)

`import()` would not work if the `rio` package were not loaded.

`select()` would not work if the `tidyverse` package were not loaded.

R Code

↺ Start Over

▶ Run Code

```
1 library(tidyverse)
2 library(rio)
3 DataHousing =
4   import("Data/HousingData.csv") |>
5   select(Price=price, Sqft=sqft_living, Bedrooms=bedrooms, Waterfront=waterfront)
6 print(DataHousing)
```

# PART 2: DATA TYPES & DATA OBJECTS

- **Data Types:** Which type of values can R store?
  - numerical **num** (such as: 0.1, 2.3, 3.14157)
  - numerical **int** (such as: 1, 2, 7)
  - character **chr** (such as: “Hello”, “Hi”, “World”)
  - categorical **factor** (such as: “Female”, “Male” Or: “small”, “medium”, “large”)
  - boolean **logic** (True, False)
- **Data Objects:** What are the **containers** R uses to store data?
  - single value as: **single entry**
  - list of entries as: **vector**
  - table as: **dataframe** or **tibble**
  - *advanced objects* can hold: plots, models, prediction results

# ANALOGY: DATA TYPES & DATA OBJECTS EXAMPLE FOR THREE ALCOHOLIC BEVERAGES

- **Data Types:** Which type of fluids can we store?
  - beer
  - wine
  - whiskey
- **Data Objects:** What are the **containers** to store our liquids?
  - bottles
  - cartons (incl. six packs)
  - cargo containers

# DATA TYPES

Main

Character

Numerical

Categorical

Logic

Truth Table

---

**Numerical Data Type (num and int):** Numerical values (e.g., 1, 523, 7 or 3.45, 0.1, 8.0) are used for calculations. In contrast, ZIP-Codes are not numerical data type.

**Character Data Type (chr):** Storing sequence of characters, numbers, and/or symbols to form a word or even a sentence is called a **character** data type (e.g. first or last names, street addresses, or Zip-codes)

**Categorical Data Type (factor):** A **factor** is an R data type that stores *categorical* data in an effective way. **factor** data types are also required by many classification models in R.

**Logic Data Type (logic):** A data type that stores the logic states **TRUE** and **FALSE** is called a **logic** object (sometimes called Boolean)

# PRINT

Print `Hello world!` by using variable A:

R Code

↺ Start Over

▶ Run Code

```
1 A=@@@  
2 print(@@@)
```



# CALCULATE WITH VARIABLES AND OUTPUT WITH `cat()`

A rectangular lot has a width of 200 feet (`Width`) and a length (`Length`) of 300 feet. Calculate the area (`Area`) and create a full sentence output.

R Code

↻ Start Over

▶ Run Code

```
1 @@@ = @@@
2 @@@ = @@@
3 @@@ = @@@ * @@@
4 cat("The area of the lot is", @@@, "square feet.")
```

# EXERCISE: `cat()` COMMAND AND SINGLE VALUE OBJECTS WITH DIFFERENT DATA TYPES

Assign your own first and last name, your ZIP code, and your your age, to three character variables (first name, last name, Zip code) and one numerical variable (age). Use `var1`, `var2`, `var3`, `var4`. Afterward, use `Cat()` to output a sentence like `Carsten Lange is 55 years old and lives in ZIP code 92656` using the variables you had created.

R Code

↻ Start Over

▶ Run Code

```
1 Var1=@@@
2 Var2=@@@
3 Var3=@@@
4 Var4=@@@
5
6 cat(@@@, @@@, "is", @@@, "years old and lives in ZIP code", @@@)
```

# AGAIN: DATA TYPES & DATA OBJECTS

- **Data Types:** Which type of values can R store? ✓
  - Numerical
  - Character
  - Categorical / Factor
- **Data Objects:** What are the containers R uses to store data? ?

# DATA OBJECTS

- **Single Value Object**
- **Vector Object**
- **Data Frame (Tibble) Object**
- **List Object** (not covered in this course)
- **Advanced Object** such as plots, models, recipes

# SINGLE VALUE OBJECT

Objects just store a single value:

R Code

↻ Start Over

▶ Run Code

```
1 A=1.5
2 B="beers are too much to drive?"
3 C=TRUE
4 cat(A, B, "Answer:", C)
```

# VECTOR-OBJECTS

A vector object stores a list of values (numerical, character, factor, or logic; mixing of data types is not allowed)

Example: Weather during the last three days in Stattown:

R Code

↻ Start Over

▶ Run Code

```
1 VecDay=c(1,2,3)
2 VecTemp=c(70, 68, 55)
3 VecWindSpeed=c("low","low","high")
4 VecIsSunny=c(TRUE,TRUE,FALSE)
5 print("Four vectors successfully defined!")
```

Vector objects can be used as arguments for an R command to calculate statistics such as the `mean()` or the number of entries in the vector (`length()`):

R Code

↻ Start Over

▶ Run Code

```
1 MeanTemp=mean(VecTemp)
2 cat("The average forecasted temperature is", MeanTemp)
```

R Code

↻ Start Over

▶ Run Code

```
1 ForecDays=length(VecTemp)
2 cat("The forecast is for", ForecDays, "days.")
```

# DATA FRAMES (TIBBLES)

A data frame is similar to an Excel table. **A data frame stores the values of R Vectors as variables entries in its columns .**

**Note**, that the `c()` command **combines** values to a vector.

Below we show how the **values from the four vectors** `VecDay`, `VecTemp`, `VecWindSpeed`, and `VevIsSunny` are stored in the *data frame* `DataWeather`.

The columns hold the values from the four vectors and the rows (with the exception of the first row), hold the observations for the various days. The first row contains the variable names:

```
R Code ↺ Start Over ▶ Run Code  
1 DataWeather=data.frame(Day=c(1,2,3), Temp=c(70,68,55),  
2 WindSpeed=c("low","low","high"),  
3 IsSunny=c(TRUE,TRUE,FALSE))  
4 kable(DataWeather)
```



# DATA FRAME FROM TITANIC DATA

Most of the times, we do not build a *data frame* from its vectors (columns). Instead we load the *data frame* from a file (for example, a `csv` file).

Below we load the *Titanic* dataset. Note, only the first six observations are shown.

```
R Code ↻ Start Over ▶ Run Code  
1 library(rio)  
2 DataTitanic=import("Data/Titanic.csv")  
3 head(DataTitanic)
```

We can see the *structure* of the data frame by using the `str()` command. This includes the type of all variables/vectors:

```
R Code ↻ Start Over ▶ Run Code  
1 str(DataTitanic)
```

# EXTRACTING THE VECTORS AND PERFORMING CALCULATIONS (NUMERICAL VECTORS)

Since the columns of a data frame are made up of *vectors*, we can extract these vectors, and use the values for data analysis (remember: observations are in the rows, variables are in the columns).

We can use the notation `DataFrameName$VectorName` to extract the vectors:

R Code

↻ Start Over

▶ Run Code

```
1 VecFareInPounds=DataTitanic$FareInPounds
2 AvgFare=mean(VecFareInPounds)
3 cat("The average fare of Titanic passengers was:", AvgFare, "British Pounds")
```

# EXTRACTING THE VECTORS AND PERFORMING CALCULATIONS (LOGICAL VECTORS)

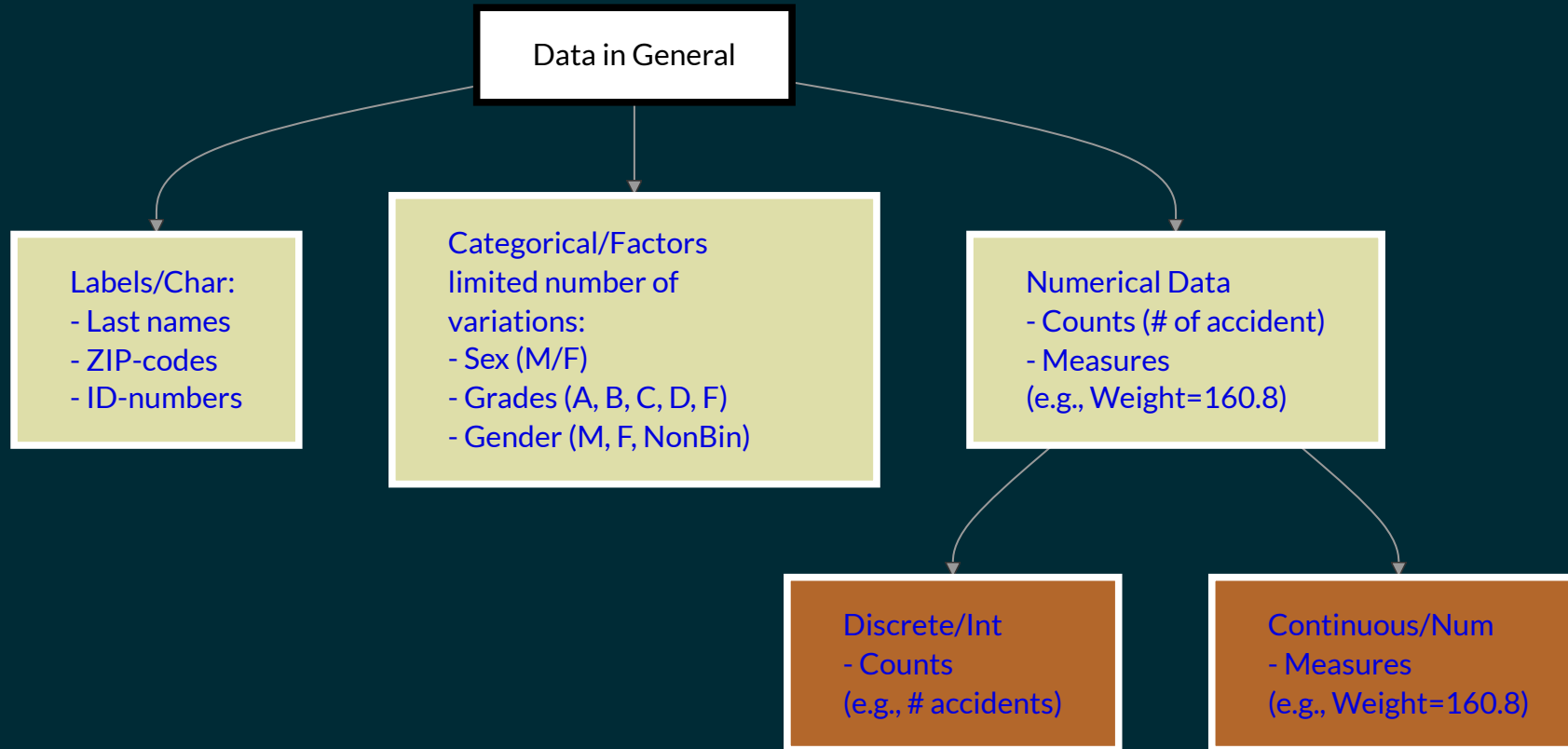
If we like, we can change a vector inside a data frame:

```
R Code ↺ Start Over ▶ Run Code  
1 DataTitanic$Survived=as.logical(DataTitanic$Survived)  
2 str(DataTitanic)
```

We can use the logical vector *Survived* (remember, **TRUE=1**, **FALSE=0**) to calculate the survival rate:

```
R Code ↺ Start Over ▶ Run Code  
1 SurvRate=mean(DataTitanic$Survived)  
2 cat("The average survival rate of Titanic passengers was:", SurvRate)
```

# SUMMARY DATA TYPES



# SUMMARY DATA TYPES AND OBJECTS

Data Type and Object Structure

# HOW ARE VERY BIG NUMBERS PRESENTED

The GDP for 2021 in the US was \$ 22,996,086,000,000 (rounded to millions)

$$GDP = 2.2996086 \cdot 10000000000000$$

$\Leftrightarrow$

$$GDP = 2.2996086 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10$$

$\Leftrightarrow$

$$GDP = 2.2996086 \cdot 10^{13}$$

Let us see what R does:

R Code

↻ Start Over

▶ Run Code

```
1 GDPUS=22996086000000
2 print(GDPUS)
```

# HOW ARE VERY SMALL NUMBERS PRESENTED

Main

Why?

The probability of getting struck by lightning in the US is about 0.000000000365 on any randomly chosen day.

$$ProbLight = \frac{3.65}{10000000000}$$

↔

$$ProbLight = \frac{3.65}{10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10}$$

↔

$$ProbLight = \frac{3.65}{10^{10}}$$

↔

$$ProbLight = 3.65 \cdot 10^{-10}$$

Let us see what R does:

R Code

↻ Start Over

Textbook

▶ Run Code

```
1 ProbStruck=0.000000000365
```

```
2 cat("Probability to get stuck by a lightning on an avg. day:", ProbStruck)
```



# PART 3: THE tidyverse AND PIPING

# BASICS OF R COMMANDS

R commands consists of the **command's name followed by a pair of parentheses**: `command()`

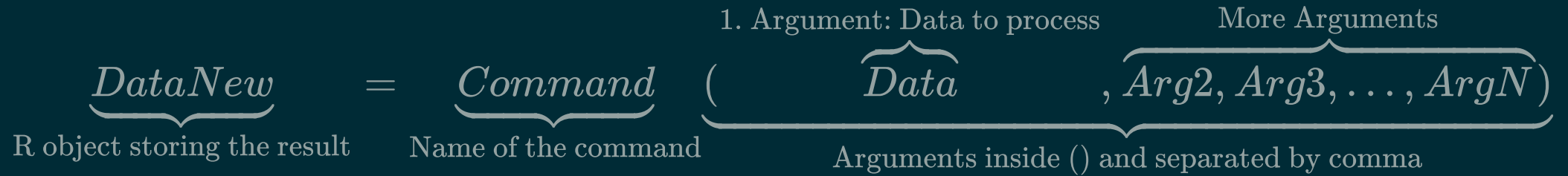
Inside the `()` we can define one or more **arguments** for the command.

```
R Code ↺ Start Over ▶ Run Code  
1 VecTest=c(1,2,3)  
2  
3 sum(x=VecTest)  
4 mean(VecTest)
```

- Arguments in a command usually have names such as `x=` or `data=`
- R does not require to use the argument's name, but **order matters**
- R commands have many arguments. Most have default values
- We can nest commands. However, nesting too deeply makes code difficult to read.»

# STRUCTURE OF R COMMANDS

Most R commands have the following structure:



Often the **data** argument is the first argument in a command. Usually named **data=** or **x=.**»

# USE A COMMAND WITH AND WITHOUT ARGUMENT NAMES

```
VecTest= c (1 ,2 ,3 )
```

R Code [↻ Start Over](#)

[▶ Run Code](#)

```
1 Result=mean(x=VecTest, trim=0, na.rm=FALSE)
2 cat("The mean of the values in vector VecTest is:", Result)
```

R Code [↻ Start Over](#)

[▶ Run Code](#)

```
1 Result=mean(VecTest, 0, FALSE)
2 cat("The mean of the values in vector VecTest is:", Result)
```

R Code [↻ Start Over](#)

[▶ Run Code](#)

```
1 Result=mean(VecTest)
2 cat("The mean of the values in vector VecTest is:", Result)
```

**All three examples are equivalent**

# GETTING HELP ABOUT A COMMAND (E.G., `mean`)

Use `?mean` or `help(mean)` in the *Positron* console to see the default values.

You can also *mark/highlight* and then press *F1*

Try it for the `mean()` command.

# IMPORTANT COMMANDS FROM `tidyverse/dplyr` PACKAGE

- `dplyr` package is part of the `tidyverse` (meta) package
- `library(tidyverse)` (loads the `tidyverse` and its packages)
- `select()` selects columns (variables) from a data frame
- `filter()` filters rows (observations) for specific criteria
- `mutate()` calculates new or overwrites existing columns (variables) based on other columns (just like Excel)
- `arrange()` sorts a data frame according to one or more columns in ascending order (use argument `desc()` for descending order)

# TITANIC DATASET

R Code

↻ Start Over

▶ Run Code

```
1 library(rio)
2 DataTitanic=import("Data/Titanic.csv")
3 head(DataTitanic)
```

# EXAMPLE: USING THE `tidyverse` FOR DATA ANALYSIS

**Goal:** Create a data frame with a few selected variables, that contains only female observations, and the fare in current U.S.-\$.



# THE `select()` COMMAND

- `select(DataMine, Var1, Var2)` selects columns (variables) `Var1` and `Var2` from a data frame `DataMine`. The first argument is the `data=` argument followed by the names of the selected variables.
- `select(DataMine, -Var1, -Var2)` selects all columns (variables) except `Var1` and `Var2` from a data frame `DataMine`.

Here is an example using the `DataTitanic` data frame with a few selected variables:

R Code

↻ Start Over

▶ Run Code

```
1 library(tidyverse)
2 DataTitanicSelVar=select(DataTitanic, Survived, PassClass=Pclass, Sex, Age, FareInPounds)
3 head(DataTitanicSelVar)
```

# THE `filter()` COMMAND

The `filter()` command filters rows (observations) of a data frame for specific criteria. The first argument is the `data=` argument followed by the filter criteria.

E.g., *filter* for female passengers:

We use `DataTitanicSelVar` that we created in the previous slide as a starting dataframe and save the result in `DataTitanicSelVarFem`.

**Note**, we have to use `==` instead of `=` for the criteria):

R Code

↻ Start Over

▶ Run Code

```
1 DataTitanicSelVarFem=filter(DataTitanicSelVar, Sex=="female")
2 head(DataTitanicSelVarFem)
```

# THE `mutate()` COMMAND

`mutate()` creates or overwrites columns (variables) based on other columns (just like Excel). The first argument is the `data=` argument followed by the instructions on how to create the new variable.

E.g., `mutate` calculates the `FareIn2023Dollars` by multiplying `FareInPounds` by 108.5. The command uses `DataTitanicSelVarFem` from the previous slide:\*

R Code

↻ Start Over

▶ Run Code

```
1 DataTitanicSelVarFemDolFare=mutate(DataTitanicSelVarFem,  
2     FareIn2023Dollars=108.5*FareInPounds,  
3     FareInPounds=NULL)  
4 head(DataTitanicSelVarFemDolFare)
```

# SUMMARY

We now have a data frame with only women and columns *Survived*, *PasClass*, *Sex*, *Age*, and *FareIn2023Dollars*.

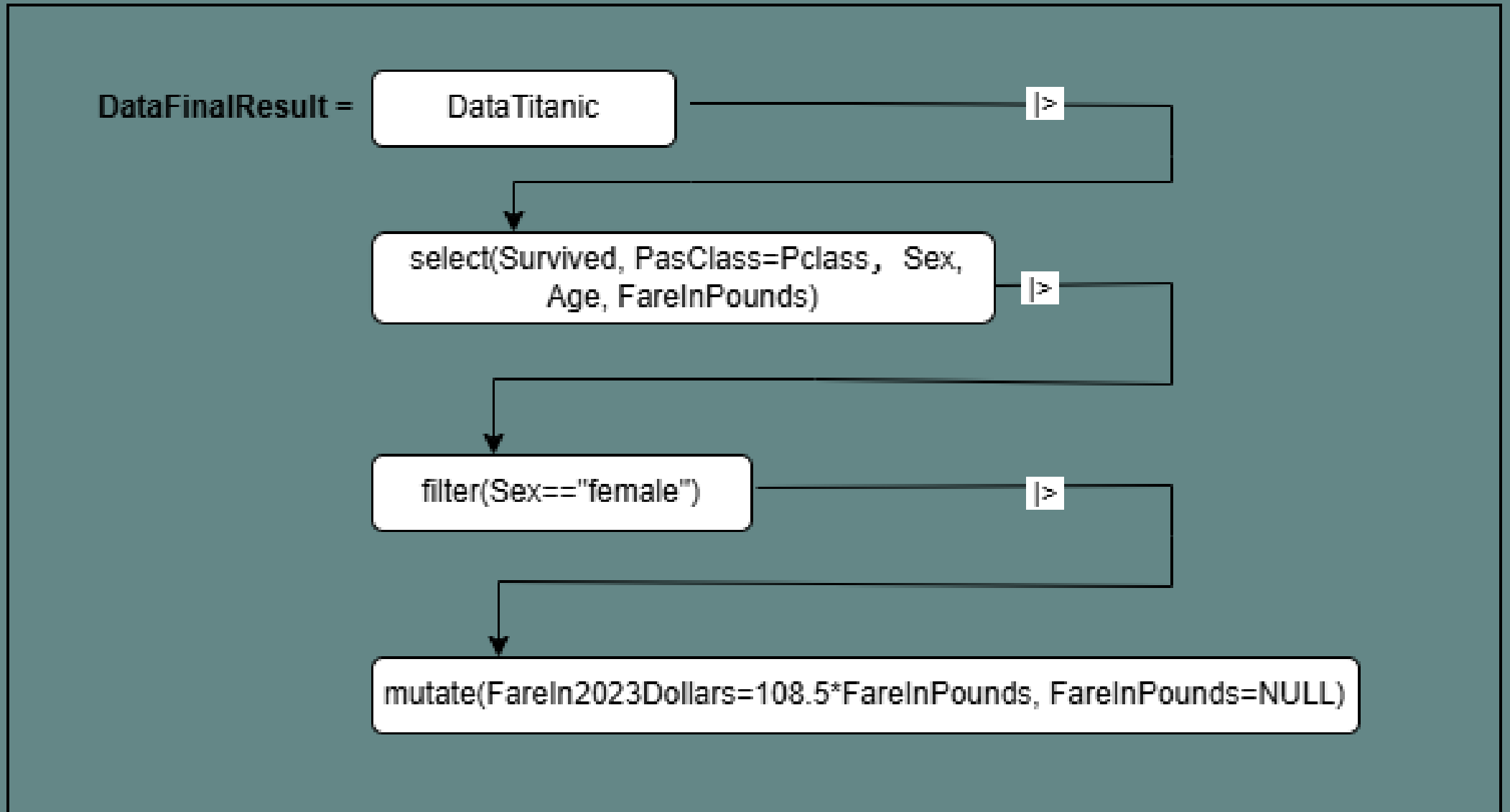
How did we get there:

1. We selected variables *Survived*, *PasClass*, *Sex*, *Age*, *FareInPounds* and saved in `DataTitanicSelVar`
2. We filtered for females and saved in `DataTitanicSelVarFem`
3. We mutated to calculate a new variable *FareIn2023Dollars* and saved finally in `DataTitanicSelVarFemDolFare`

Could this be done easier?

Note, overwriting data frames such as `DataTitanic` is usually a bad idea! Nesting the command is possible but very difficult to read.

# PIPING SCHEMA



Piping Schema

# ALTERNATIVE: PIPING

(WILL BE USED THROUGHOUT THE COURSE/BOOK)

**Shortcut** for `|>`: **CTRL SHIFT M** (Windows) or **⌘ SHIFT M** (Mac).

The pipe operator `|>` is for most practical purposes equivalent to `%>%`.

```
R Code ↻ Start Over ▶ Run Code
1 library(tidyverse)
2 DataTitanicFinal=DataTitanic |>
3   select(Survived, PasClass=Pclass , Sex, Age, FareInPounds) |>
4   filter(Sex=="female") |>
5   mutate(FareIn2023Dollars=108.5*FareInPounds, FareInPounds=NULL)
6 head(DataTitanicFinal)
```

# WHY R?

- Excel analytics is not reproducible
- SPSS focuses on surveys
- STATA and SAS are commercial products
  - not free
  - progress has to go through the corporate hierarchy and therefore is slower
  - limited support community

# R AND/OR PYTHON

- Analysis is always reproducible with little effort
- free
- extensive support
- R or Python
  - R is easier to understand for users with limited coding experience
  - Python is faster in incorporating cutting-edge algorithms
  - transfer from R to Python or vice versa is easy
  - Quarto supports both R and Python even simultaneously in the same project



# PYTHON VS. R – THE TASK

Let us compare code for the same tasks between *R* and *Python*:

- Download the Titanic dataset
- select the variables `Sex`, `FareInPounds`, `Survived` (renamed to: `Surv`)
- Calculate a new column `FareInDollars` by multiplying `FareInPounds` by 108.5
- Filter for `Sex` being *female*
- Calculate the mean of `FareInDollars`

# PYTHON VS. R – THE RESULTS (USING PANDAS)

```
1 library(tidyverse)
2 library(rio)
3 DataTitanicR = import("Data/Titanic.csv") |>
4   select(Sex, FareInPounds, Surv = Survived) |>
5   mutate(FareInDollars = FareInPounds * 108.5) |>
6   filter(Sex == "female")
7 MeanFareWomen = mean(DataTitanicR$FareInDollars)
8 print(MeanFareWomen)
```

[1] 4826.06

Python Code

↻ Start Over

▶ Run Code

```
1 import pandas as pd
2 DataTitanicPy=pd.read_csv("Data/Titanic.csv")
3 DataTitanicPy=DataTitanicPy[["Sex", "FareInPounds", "Survived"]] \
4     .rename(columns={"Survived": "Surv"}) \
5     .assign(FareInDollars = DataTitanicPy["FareInPounds"] * 108.5) \
6     .query("Sex=='female'")
7 MeanFareWomen=DataTitanicPy["FareInDollars"].mean()
8 print(MeanFareWomen)
```

# PYTHON VS. R – THE RESULTS (USING POLARS)

```
1 library(tidyverse)
2 library(rio)
3 DataTitanicR = import("Data/Titanic.csv") |>
4   select(Sex, FareInPounds, Surv = Survived) |>
5   mutate(FareInDollars = FareInPounds * 108.5) |>
6   filter(Sex == "female")
7 MeanFareWomen = mean(DataTitanicR$FareInDollars)
8 print(MeanFareWomen)
```

[1] 4826.06

Python Code

↻ Start Over

▶ Run Code

```
1 import polars as pl
2 DataTitanicPy=pl.read_csv("Data/Titanic.csv", infer_schema_length=None) \
3     .select(pl.col("Survived").alias("Surv"), pl.col("Sex"), pl.col("FareInPounds"),
4             (pl.col("FareInPounds")*108.5).alias("FareInDollars")) \
5     .filter(pl.col("Sex") == "female")
6 MeanFareWomen=DataTitanicPy["FareInDollars"].mean()
7 print(MeanFareWomen)
```

**Note,** `polars` is currently not supported in WASM

# WAS CHIVALRY DEAD IN 1912?

To answer the question, we develop a male and a female data frame and compare the survival rates.

In each data frame we would need only the variables `Sex` and `Survived` but we add also `PasClass` for additional analysis.

# EXERCISE: THE MALE DATA FRAME

We select `Sex`, `Survived`, and `PasClass=Pclass` and filter for `male`:

R Code

 Start Over

 Run Code

```
1 DataMale = DataTitanic |>
2   select(@@@) |>
3   filter(@@@)
4 head(DataMale)
```

# THE FEMALE DATA FRAME

We select `Sex`, `Survived`, and `PasClass=Pclass` and filter for `female`:

R Code

↻ Start Over

▶ Run Code

```
1 DataFemale = DataTitanic |>
2   select(@@@) |>
3   filter(@@@)
4 head(DataFemale)
```

# COMPARING THE SURVIVAL PROPORTION OF MALES TO FEMALES

**Hint:** You could either calculate the female proportion as `sum(DataFemale$Survived/nrow(DataFemale))` or `mean(DataFemale$Survived)`.

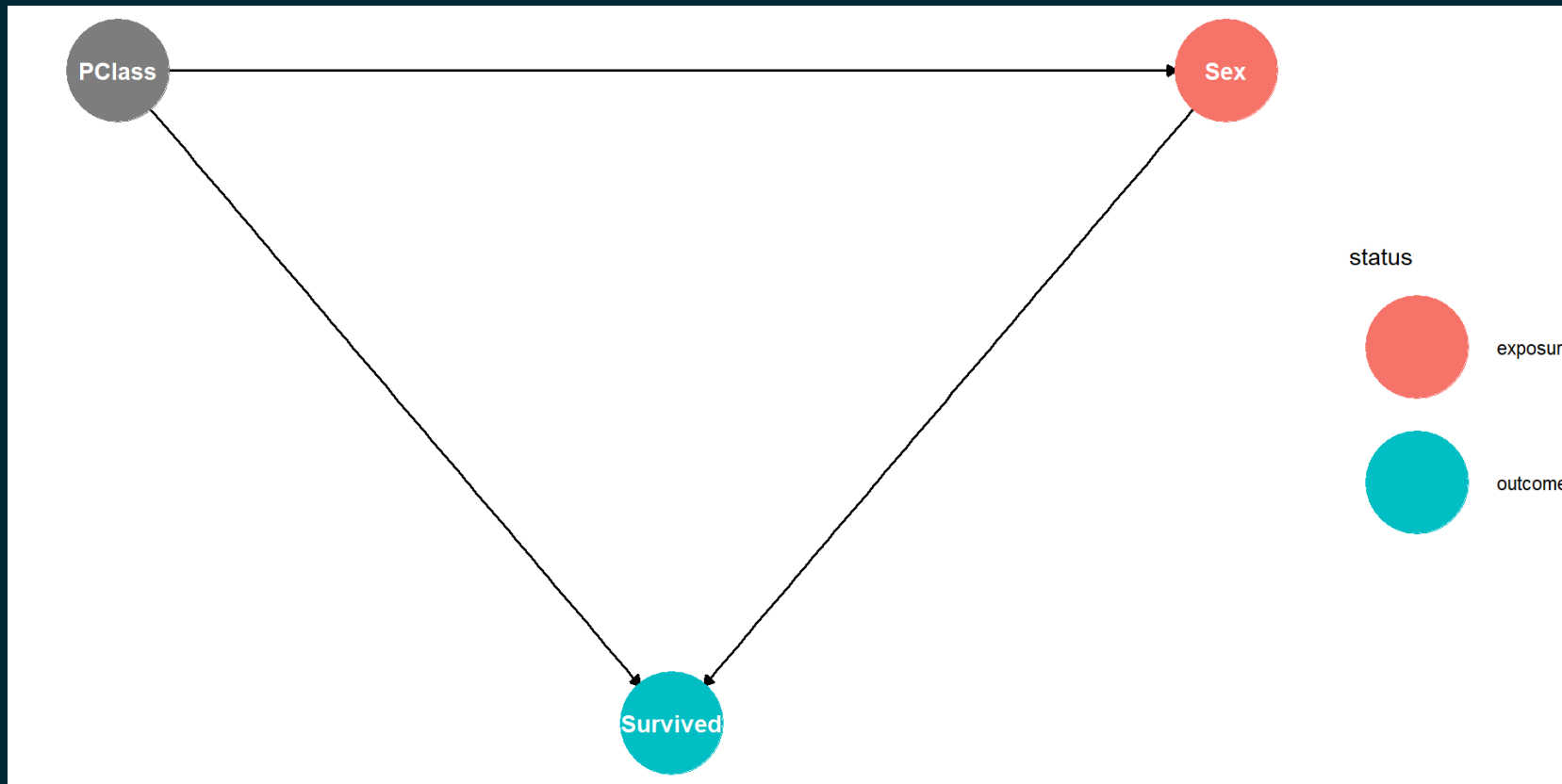
R Code

```
1 PropFemSurv = @@@
2 PropMaleSurv = @@@
3 cat(@@@, @@@)
4 cat(@@@, @@@)
```

# BE CRITICAL WITH YOUR OWN RESEARCH

## PasClass IS A CONFOUNDER

The third class was deep in the hull of the Titanic with low survival chances and more men were traveling in that class. This makes **PasClass** a confounder. Therefore we have to analyze male and female survival by class: We have to filter for **Sex** and **PasClass**.





# SURVIVAL RESEARCH FOR PASSENGER CLASS 1

- Select `Survived`, `Sex`, `PasClass=Pclass`
- Filter for `PasClass` and `Sex` (`female` and `male`)

```
R Code ↻ Start Over ▶ Run Code  
1 DataFemaleClass1=DataTitanic |>  
2   select(@@@, @@@, PasClass=Pclass) |>  
3   filter(@@@) |>  
4   filter(@@@)  
5 DataMaleClass1=DataTitanic |>  
6   select(@@@, @@@, PasClass=Pclass) |>  
7   filter(@@@) |>  
8   filter(@@@)  
9 cat(" Female Survival Proportion in Class 1:", mean(DataFemaleClass1$Survived), "\n",  
10    "Male Survival Proportion in Class 1:", mean(DataMaleClass1$Survived))
```

# SURVIVAL RESEARCH FOR PASSENGER CLASS 2

- Select `Survived`, `Sex`, `PasClass=Pclass`
- Filter for `PasClass` and `Sex` (`female` and `male`)

```
R Code ↻ Start Over ▶ Run Code  
1 DataFemaleClass2=DataTitanic |>  
2   select(@@@, @@@, PasClass=Pclass) |>  
3   filter(@@@) |>  
4   filter(@@@)  
5 DataMaleClass2=DataTitanic |>  
6   select(@@@, @@@, PasClass=Pclass) |>  
7   filter(@@@) |>  
8   filter(@@@)  
9 cat(" Female Survival Proportion in Class 2:", mean(DataFemaleClass2$Survived), "\n",  
10    "Male Survival Proportion in Class 2:", mean(DataMaleClass2$Survived))
```

# SURVIVAL RESEARCH FOR PASSENGER CLASS 3

- Select `Survived`, `Sex`, `PasClass=Pclass`
- Filter for `PasClass` and `Sex` (`female` and `male`)

```
R Code ↻ Start Over ▶ Run Code  
1 DataFemaleClass3=DataTitanic |>  
2   select(@@@, @@@, PasClass=Pclass) |>  
3   filter(@@@) |>  
4   filter(@@@)  
5 DataMaleClass3=DataTitanic |>  
6   select(@@@, @@@, PasClass=Pclass) |>  
7   filter(@@@) |>  
8   filter(@@@)  
9 cat(" Female Survival Proportion in Class 3:", mean(DataFemaleClass3$Survived), "\n",  
10    "Male Survival Proportion in Class 3:", mean(DataMaleClass3$Survived))
```