# Week 2 Tutorial: R Basics

POP77001 Computer Programming for Social Scientists

# R and development environments

- There is some choice of integrated development environments (IDEs) for R (StatET, ESS, R Commander)
- However, over the last decade RStudio became the de factor standard IDE for working in R
- You can also find R extensions for your favourite text editor (Atom, Sublime Text, Visual Studio Code, Vim)
- For the purposes of consistency with Python part of the module, we will be using Jupyter with R.

# Running R in Jupyter

- In order to be able to run R kernel in Jupyter, you need to install package `IRkernel`:
  - Open R (in the terminal) or RStudio:
  - Run `install.packages("IRkernel")` to install the package
  - Wait until the package is installed
  - Run `IRkernel::installspec()` to initialize R kernel for Jupyter
  - Now you should be able to launch or edit a notebook with R kernel

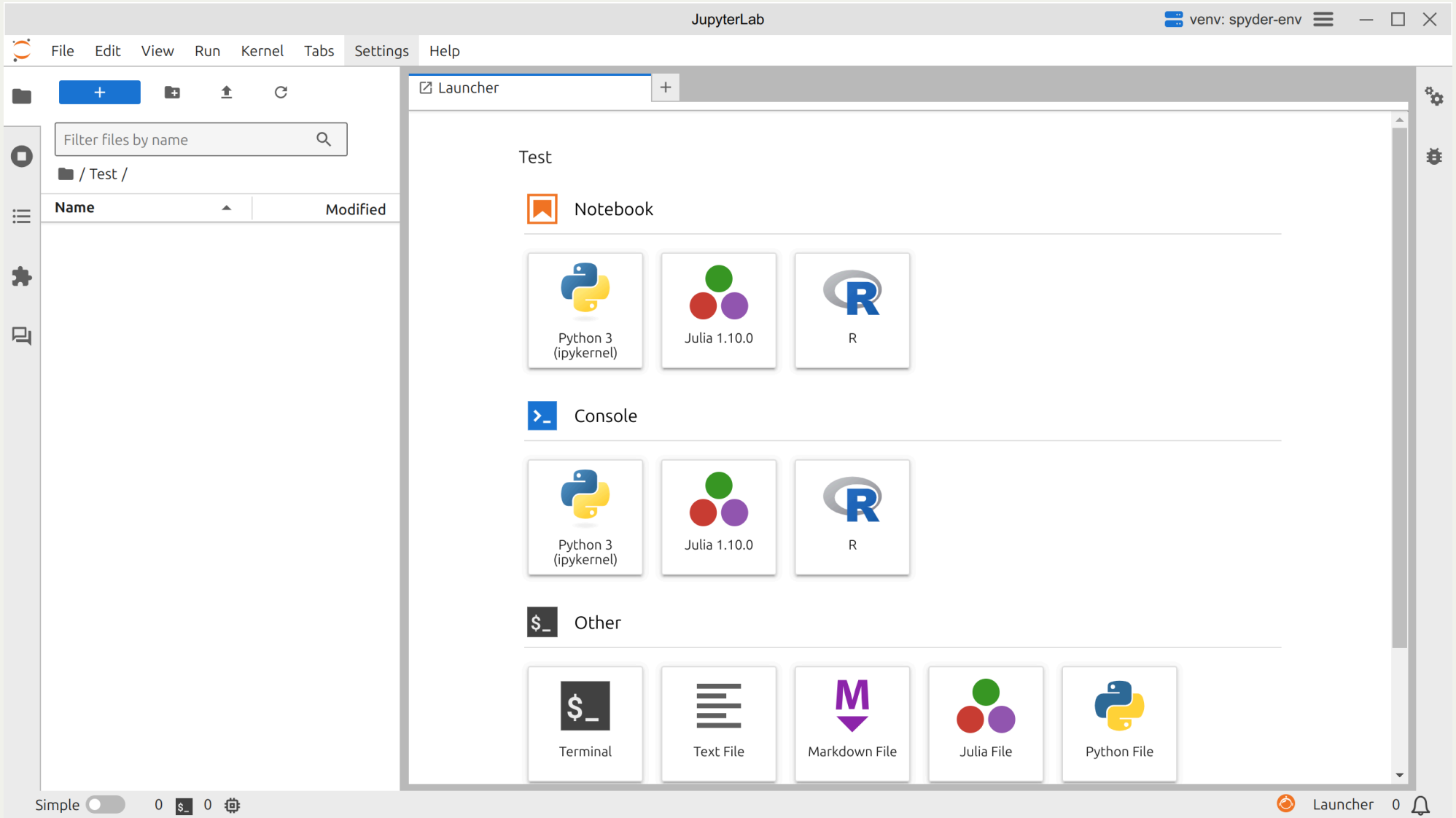
Tip: When starting working with R in Jupyter run `options(jupyter.rich_display = FALSE)` command to switch off pretty printing and get the output (albeit less neat) consistent with output in RStudio

# IRkernel

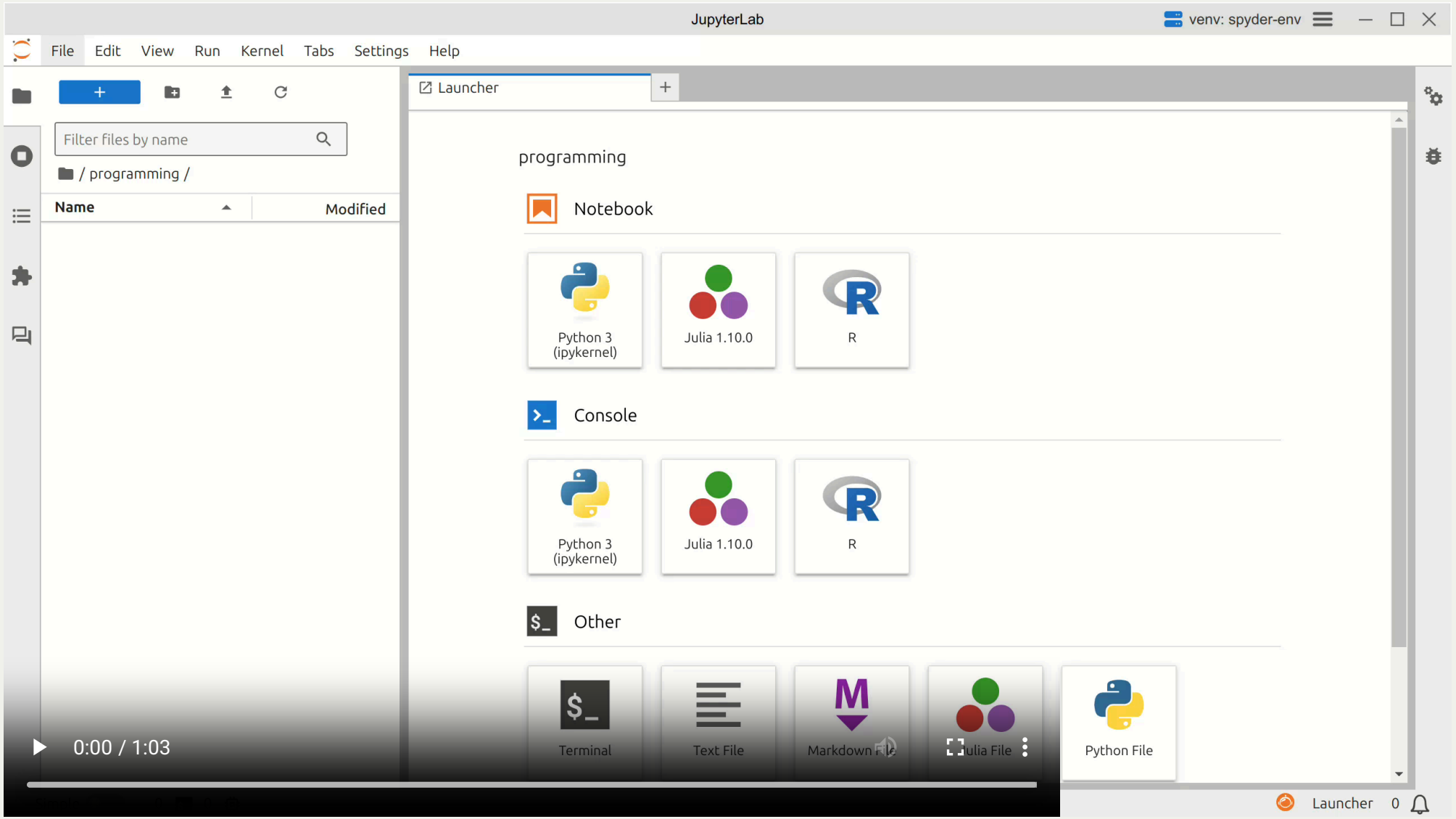
Package `IRkernel` is required to run R in Jupyter Notebook.

```
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.

> install.packages("IRkernel")
Installing package into ‘/home/tpaskhalis/R/x86_64-pc-linux-gnu-library/4.1’
(as ‘lib’ is unspecified)
trying URL 'https://cloud.r-project.org/src/contrib/IRkernel_1.2.tar.gz'
Content type 'application/x-gzip' length 62663 bytes (61 KB)
==================================================
downloaded 61 KB

* installing *source* package ‘IRkernel’ ...
** package ‘IRkernel’ successfully unpacked and MD5 sums checked
** using staged installation
** R
** inst
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (IRkernel)

The downloaded source packages are in
        ‘/tmp/RtmpnK5Pss/downloaded_packages’
> IRkernel::installspec()
[InstallKernelSpec] Removing existing kernelspec in /home/tpaskhalis/.local/share/jupyter/kernels/ir
[InstallKernelSpec] Installed kernelspec ir in /home/tpaskhalis/.local/share/jupyter/kernels/ir
>
```

# JupyterLab


JupyterLab
venv: spyder-env
File
Edit
View
Run
Kernel
Tabs
Settings
Help
Launcher
Filter files by name
/ Test /
Name
Modified
Test
Notebook
Python 3 (ipykernel)
Julia 1.10.0
R
Console
Python 3 (ipykernel)
Julia 1.10.0
R
Other
Terminal
Text File
Markdown File
Julia File
Python File
Simple
0
0
Launcher
0

# JupyterLab Demonstration



# Code Distribution

More often than not you want to record how analysis was performed.

There are 3 principal ways of distributing R code:

- R script (`.R` file)
- R Markdown/Quarto (`.Rmd`/`.qmd` file)
- Jupyter Notebook (`.ipynb` file)

# R Script

- The most straightforward way to keep track of your R code.
- Instead of writing your R commands in the interactive console,
- You put them in a script and run then together or one at a time.
- Can contain a mix of valid R commands and comments (lines starting with #).
- Easy to edit and integrate into larger projects.

# Markdown formatting basics

- Use `_` or `*` for emphasis (single - italic, double - bold, triple - bold and italic)
  - `*one*` becomes *one*, `__two__` - **two** and `***three***` - ***three***
- Headers or decreasing levels follow `#`, `##`, `###`, `####` and so on
- (Unordered) Lists follow marker `-`, `+` or `*`
  - Start at the left-most position for top-level
  - Indent four space and use another marker for nesting like here
- (Numbered) Lists use `1.` (counter is auto-incremented)
- Links have syntax of `[some text here](url_here)`
- Images similarly: `![alt text](url or path to image)`

# Markdown vs R Markdown

- Markdown:
  - Easy-to-read and easy-to-write plain text format;
  - Separates content from its appearance (rendition);
  - Widely used across industry sectors and academic fields;
  - `.md` file extension.
- R Markdown (Quarto):
  - Allows combining of R commands with regular text;
  - Compiles into PDF/DOC/HTML and other formats;
  - Can be converted into slide deck or even website!
  - `.Rmd` file extension (`.qmd` for Quarto).

**Extra**

Ch 27: R Markdown in Wickham & Grolemund 2017

# R Markdown

## R Markdown

````
### Title

Some text in *italic* and **bold**

Simple list:

- A
- B

Ordered list:

1. A
1. B

Example, where $Y_i = 5 + X_i + \epsilon$

```{r}
x_i <- 3
epsilon <- rnorm(1)
y_i <- 5 + x_i + epsilon
y_i
```
````

## Rendering

### Title

Some text in *italic* and **bold**

Simple list:

- A
- B

Ordered list:

1. A
2. B

Example, where $Y_i = 5 + X_i + \epsilon$

```r
x_i <- 3
epsilon <- rnorm(1)
y_i <- 5 + x_i + epsilon
y_i
```

```
[1] 9.685856
```

# Naming conventions

- Even while allowed in R, do not use `.` in variable names (it works as an object attribute in Python)
- Do not name give objects the names of existing functions and variables (e.g. `c`, `T`, `list`, `mean`)
- Use **UPPER_CASE_WITH_UNDERSCORE** for named constants (e.g. variables that remain fixed and unmodified)
- Use **lower_case_with_underscores** for function and variable names

**Extra**

The tidyverse style guide

# Code layout

- Limit all lines to a maximum of 79 characters.
- Break up longer lines

```
my_long_vector <- c(
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22,
  23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
  42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60
)

long_function_name <- function(a = "a long argument",
                               b = "another argument",
                               c = "another long argument") {
  # As usual code is indented by two spaces.
}
```

# Reserved words

There are ~14 reserved words in R that cannot be used as names assigned to objects.

| | |
|---|---|
| `break` | `NA` |
| `else` | `NaN` |
| `FALSE` | `next` |
| `for` | `NULL` |
| `function` | `repeat` |
| `if` | `TRUE` |
| `Inf` | `while` |

**Extra**

R reserved words

# Exercise 1: Vector subsetting

- Load built-in R object `letters` (lower-case letters of the Roman alphabet)
- Calculate its length
- Generate a vector of integers that starts from 1 and has the same length as `letters`
- Assign to each integer corresponding lower-case letter as its name
- Use these names to subset all vowels
- Now, repeat the subsetting, but using indices rather than names

Tip: You can use function `which()` for determining the indices of vowels

```
1 letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

# Tabulation & Crosstabulation

- R function `table()` provides an easy way of summarizing categorical variables
- Note that variables represented as character vectors are implicitly converted to factors

```r
# Top 10 most populous settlements on the island of Ireland
# https://en.wikipedia.org/wiki/List_of_settlements_on_the_island_of_Ireland_by_population
top_10_settlements <- c(
    "Dublin", "Belfast", "Cork", "Limerick", "Derry",
    "Galway", "Newtownabbey", "Bangor", "Waterford", "Lisburn"
)
```

```r
# Corresponding provinces
provinces <- c(
    "Leinster", "Ulster", "Munster", "Munster", "Ulster",
    "Connacht", "Ulster", "Ulster", "Munster", "Ulster"
)
```

# Tabulation & Crosstabulation

```r
# Given that each town appears only once, cross-tabulation might not be the most informative
table(top_10_settlements, provinces)
```

```
                  provinces
top_10_settlements Connacht Leinster Munster Ulster
      Bangor               0        0       0      1
      Belfast              0        0       0      1
      Cork                 0        0       1      0
      Derry                0        0       0      1
      Dublin               0        1       0      0
      Galway               1        0       0      0
      Limerick             0        0       1      0
      Lisburn              0        0       0      1
      Newtownabbey         0        0       0      1
      Waterford            0        0       1      0
```

```r
# Instead, we can just get tabulate the `provinces` vector
# and check the value counts for each province
table(provinces)
```

```
provinces
Connacht Leinster  Munster   Ulster
       1        1        3        5
```

# Exercise 2: Working with Factors

- As you note the output of `table(provinces)` is sorted alphabetically
- Change this to reflect the actual counts
- First, let's store the result of tabulation for later re-use
- Start from exploring the structure of this object with `str()`
- What are the 2 main parts of this object? How are they stored?
- Extract the relevant parts from the stored object
- Save them as a named vector with provinces as names and counts as values
- Use `sort()` function to sort the vector in a decreasing order (from largest to smallest)
- Convert the original `provinces` vector into a factor with the levels ordered accordingly
- Re-run `table(provinces)`

```
1 tab <- table(provinces)
```

# Week 2 Exercise (unassessed)

- Save a `letters` object under a different name
- Convert saved object into a matrix of 13 rows and 2 columns
- Subset letter ‘f’ using indices
- Concatenate 3 copies of `letters` object together in a single character vector
- Convert it into a 3-dimensional array, where each dimension appears as a matrix above
- Subset all letters ‘f’ across all 3 dimensions