Introduction to Applied Data Science

Lecture 1: Introduction to Data Science

Bas Machielsen
Utrecht University
2024-04-22

Introduction

Introduction

- Introduction to Applied Data Science
  - The goal of this course is to give you a rapid overview of the main tools of data science:
  - A basic understanding of programming using the R language
  - Acquiring, importing and tidying data
  - Analyzing and working with a few modern data formats
  - Reporting and presenting your findings efficiently
- Overview of this class:
  - **This lecture**: Introduction to Data Science
  - Introduction to R & Programming
  - Getting Data: API's and Databases
  - Getting Data: Web Scraping
  - Transforming and Cleaning Data
  - Spatial & Network Data
  - Text as Data and Mining
  - Data Science Project (Tentative) - Vote at end of lecture

Course Content

- Introduction to R, in arguably the most user-friendly way of programming:
  - You will learn to **read and understand** code
  - And communicate effectively with computers using code rather than a User Interface
- Integrating data **collection** and data **cleaning**
  - How do you get data to test hypotheses?
- Introduction to **non-standard, more modern** data formats
- Writing **reports and presentations** of your analyses using RMarkdown
  - An alternative to Microsoft Word and Google Docs

Learning Goals

- On effective completion of the course, students should:
  - Understand the **basics** of R programming in a data science context
  - Be able to **independently acquire data** from a variety of sources
  - Understand and be able to analyze **non-standard formats of data** such as text and spatial data
  - Be able to **integrate code in reporting**, thereby writing reproducible code and analysis

Who am I?

- Bas Machielsen
- Assistant Professor Applied Economics
- Research: Economic History, Political Economy
- Background in Economics, Econometrics, Data Science, Coding experience in R and Python
- Contact
    - Email: a.h.machielsen@uu.nl
    - Office hours: **Tuesday, after the lecture** from 12:00-13:00 in ASH 1.12b

Course Structure

- The format of the course is simple: we'll have 8 Lectures and 8 Tutorials
- In 7 of the tutorials, we'll focus on an **in-depth explanation** of the content of the lecture
    - The 8th tutorial will be dedicated to the final exam and will feature a mock exam
- The course will have a *mid-term exam* and a *final exam*
- The mid-term exam counts for 40% of the grade, the final exam for the remaining 60%
- These exams will be conducted *in-person* using your *own device*
    - The exams will *not* be paper exams, but exams in which you have to answer questions, sometimes verbally, sometimes in code in a so-called *Rmarkdown document*
    - You will hand in (submit) the exam on Blackboard
    - Make sure to **charge your laptop beforehand**

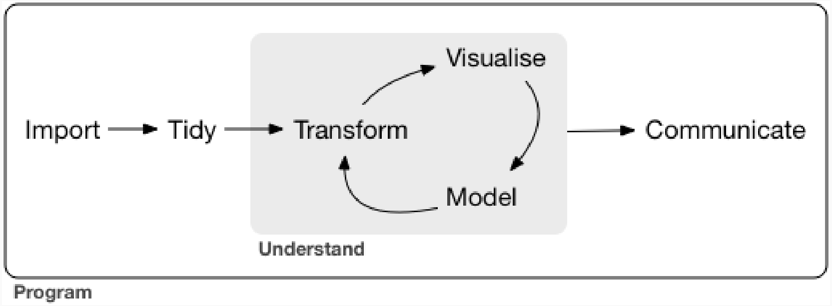
Why do we need data?

Why do we need data?

- Very fundamentally, why do we need data?
- Roughly, three purposes:
  - To *measure* a quantity of interest, such as a country's Gross Domestic Product (GDP)
  - To *predict* a quantity of interest, such as stock prices
  - To *explain* a quantity of interest, such as the effect of education on earnings
- Measuring and prediction can also lead to the formation of theory, which can then be *tested* empirically
- In economics, we are fundamentally interested in *explaining* things, rather than prediction or measurement
- However, there are also plenty of applications for these purposes

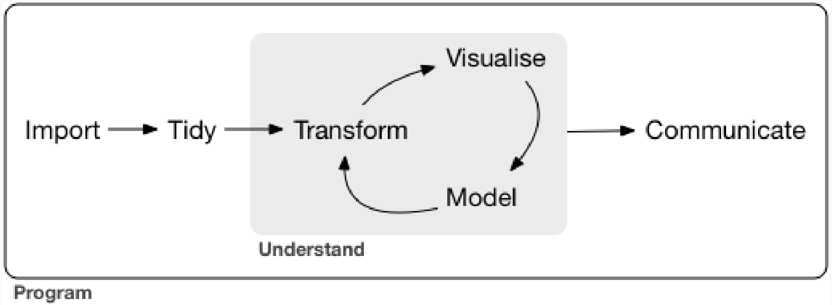
Why Data Science?

- **Contemporary economics** does a lot of **empirical** work (meaning, testing theories)
- The data used in economics research comes from a wide variety of sources
  - The analyses are getting **more and more diverse**
- Hence, more and more advanced coding skills and creativity in acquiring data are required.
  - Here **data science** comes into play
- Our model of the **tools** needed in a typical data science project looks something like this:



Why Data Science?

- Figuring out whether you want to **measure, predict or explain** something should always precede your analysis (model in the figure)
  - And should often also **precede** data collection



- In all of these cases, however, you need to run through these steps
- Tidy and *transform* the data to make it suitable for analysis
  - Recode, rearrange or regroup data to create *variables* in columns and *observations* in rows

Tidy Data

There are three **interrelated rules** which make a dataset tidy:

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

```
palmerpenguins::penguins |> head(8)
```

```
## # A tibble: 8 × 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <fct>   <fct>              <dbl>         <dbl>             <int>       <int> <fct>
## 1 Adelie  Torgersen           39.1          18.7               181        3750 male
## 2 Adelie  Torgersen           39.5          17.4               186        3800 femal
## 3 Adelie  Torgersen           40.3          18                 195        3250 femal
## 4 Adelie  Torgersen           NA            NA                  NA          NA <NA>
## 5 Adelie  Torgersen           36.7          19.3               193        3450 femal
## 6 Adelie  Torgersen           39.3          20.6               190        3650 male
## 7 Adelie  Torgersen           38.9          17.8               181        3625 femal
## 8 Adelie  Torgersen           39.2          19.6               195        4675 male
```

Untidy Data

- An example of untidy data:

```
untidy_data ▷ head(10)
```

```
## # A tibble: 10 × 5
##    species island       id var               value
##    <chr>   <chr>     <int> <chr>             <chr>
##  1 Adelie  Torgersen     1 bill_length_mm    39.1
##  2 Adelie  Torgersen     1 bill_depth_mm     18.7
##  3 Adelie  Torgersen     1 flipper_length_mm 181
##  4 Adelie  Torgersen     1 body_mass_g       3750
##  5 Adelie  Torgersen     1 sex               male
##  6 Adelie  Torgersen     1 year              2007
##  7 Adelie  Torgersen     2 bill_length_mm    39.5
##  8 Adelie  Torgersen     2 bill_depth_mm     17.4
##  9 Adelie  Torgersen     2 flipper_length_mm 186
## 10 Adelie  Torgersen     2 body_mass_g       3800
```

Transforming Data

- Once you have **tidy data**, a common first step is to transform it.
- Transformation includes:
  - **Narrowing in** on observations of interest (like all people in one city, or all data from the last year)
  - **Creating** new variables that are functions of existing variables (like computing speed from distance and time)
  - **Calculating** a set of summary statistics (like counts or means).
- Together, tidying and transforming are called **data wrangling**
- One of the most **underrated aspects of data science** and one of the most useful skills you might learn

Data Analysis

- Once data is cleaned and organized, you usually want to **analyze** it.
- Data analysis is done with one of the three purposes mentioned earlier in mind: measurement, prediction or explanation.
- It is important that we analyze the data using appropriate models.
  - These models might come from *econometrics*, inspired by *economic theory* or from *machine learning*
  - You have already learned a foundational model: the *linear regression model* in statistics:

$$Y_i = \alpha + \beta X_i + \epsilon_i$$

- If our purpose is *explanation*, we are usually interested in the β coefficient
- If our purpose is prediction, we are interested in the *accuracy* of the model

Communication

- The last step of data science is **communication**, an absolutely critical part of any data analysis project.
- It doesn’t matter how well your models and visualization have led you to understand the data unless you can also interpret and communicate your results to others
- Communication encompasses creating nice graphs and tables
- But also to interpret models in good, no-nonsense language

Programming and R

Why R?

- R is in the process of becoming the most important data science language
- Also in economics, where it is (likely) replacing **Stata** (a paid alternative)
- I think this is partially due to its **ease to learn** and **user friendliness**
  - Even though you might not agree when you are frustrated!
- In this class, the focus is on *understanding* the code
  - You don't have to write (much) code yourself
  - We'll be using a *user-friendly, intuitive* approach to R specifically fine-tuned towards data wrangling and cleaning

Why R?

- R is a programming language (like others such as Python, Matlab, C++)
- It provides a set of basic **functions** and **operations** which you can execute on .. data
    - A function is, like in mathematics, something that takes an input and transforms it into an output
- A few short examples:

```
mean(c(1,2,3))
```

```
## [1] 2
```

```
x ← c(1,2,3,4)
y ← c(2,4,1,2)

cor(x,y)
```

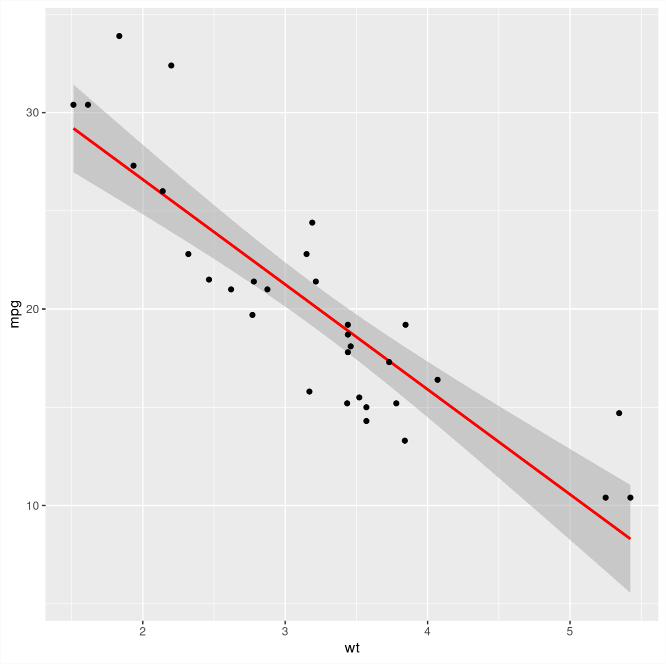
```
## [1] -0.3077935
```

Preview

- A short preview of what R can create:

```r
library(ggplot2)

ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_smooth(method = "lm", col = "red") +
  geom_point()
```



Preview

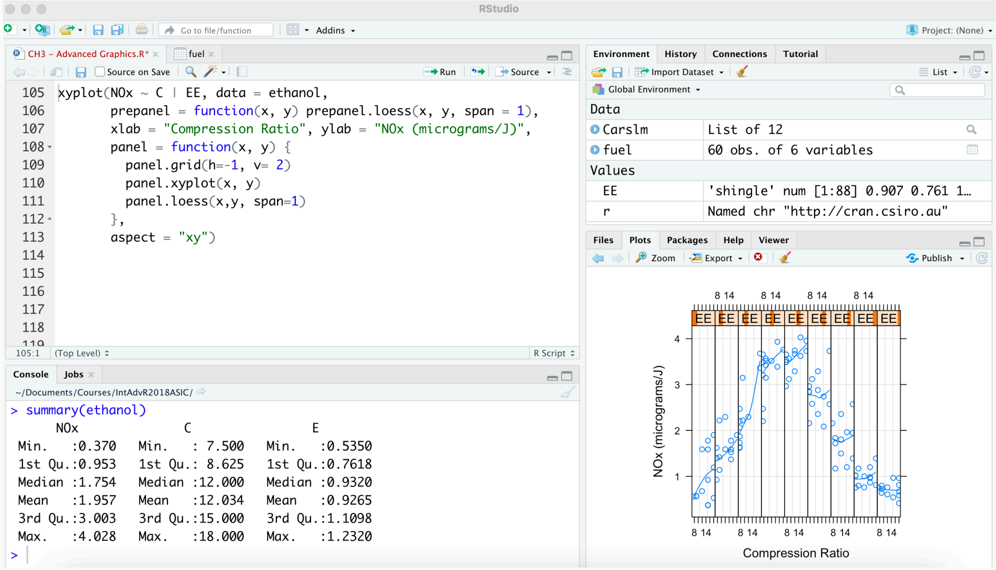
- A short preview of what R can create:

	female			male		
	mean	**sd**	**N**	**mean**	**sd**	**N**
bill_length_mm	42.10	4.90	165	45.85	5.37	168
bill_depth_mm	16.43	1.80	165	17.89	1.86	168
flipper_length_mm	197.36	12.50	165	204.51	14.55	168
body_mass_g	3862.27	666.17	165	4545.68	787.63	168
year	2008.04	0.81	165	2008.04	0.81	168

RStudio

RStudio

- RStudio is an **IDE** (Integrated Development Environment) focused on R
  - Rstudio is a **piece of software** used to make programming in R easier
  - Apart from allowing us to program, it'll help us do other things, like write up documents and interpret our code output, as we will see shortly.
- This is what RStudio looks like:



RStudio

- You can download R and RStudio from https://posit.co/download/rstudio-desktop/
- Make sure to first **download R**, the programming language
    - Click "Download & Install R"
- And then **download Rstudio**, the program which we use to write our code (in R) with
    - Scroll down and select your Operating System

RStudio: Layout

- RStudio contains four **subscreens** (three if you haven't opened an R script yet)
- The upper left screen is a script - usually a `.R` file or an `.Rmarkdown` file
  - You **write and save** your code up here
- The bottom left is the **console**
  - The console is asking you "What should I do next?": this is an open R session
  - You can use it to e.g. install packages, and try out stuff
- The upper right contains various tabs, the most important of which is the **environment**
  - This stores all the objects you have made into your (RAM) **memory**
- The bottom right contains a **File Explorer, and a Graphics Viewer**

Basics in RStudio: The main window

- In this course, we'll mainly focus on the upper left part (A `.R` or `.Rmd` file where you write your code in)

- If you have installed Rstudio, try creating a new Rmarkdown file file by File > New File > Rmarkdown

  - You can use the default options and click OK.

- The **upper left window** is the main window for writing and editing scripts

  - This is where you **write, edit, and save** your R scripts.
  - It's like a **digital notebook** where you write down your R code. Here, you can write lines of code to perform calculations, manipulate data, create visualizations, and much more. It's where you spend most of your time crafting your R programs

- We will learn a lot about Rmarkdown during the **first tutorial**

- You will also make your midterm and final exams in an Rmarkdown document

Basics in RStudio: The console

- The lower left window is the *console*: The console is like a command center where you can interact with R in real-time
  - You can think of it as a place where you give orders to R, and it immediately executes them and shows you the results
  - When you run your scripts (the code you've written) from the editing window, the results often appear in the console
  - It's also where **error messages** show up if something goes wrong with your code
  - The console is a **valuable tool** for testing small bits of code, experimenting with functions, and getting immediate feedback from R
- Try typing a calculation in the console, like `1+1`

Libraries or "Packages"

- Before proceeding to do anything else in RStudio, it is useful to download and install a couple of *packages* or *libraries*
- In R, packages are collections of **functions, datasets, and other resources** that extend the capabilities of the base R system.
- They are essentially **bundles of code that are created by other users or developers** to provide additional functionality for specific tasks, such as data manipulation, statistical analysis, visualization, machine learning, and more.
- Many students include things like `install.packages()` in their code, which prompts R to (re-)install the package every time the code is run.
    - Don't do this! As a rule, you should install packages in the *console*, not in your R script
- You can install packages using the `install.packages()` function, with the name of the package you want to install within brackets

Installing or Loading Packages

- Once a package is installed, you need to **load it** into your R session before you can use its functions and data.
- Compare this to installing a video game and then clicking it to be able to play
- In R, like with video games, you have to install packages, and only then, you can use them
- When you want to use them, you have to tell R that you want to use them
- You do this by means of the `library()` command, with the name of the package within brackets

The First Packages to Install

- The first package we will install is a **package manager**, which allows you to either load or install packages, depending on whether the package is already installed
    - This package manager is called `pacman`. To install it, use:

```
install.packages("pacman")
```

- This package manager gives us the opportunity to use *one command* to install and/or load packages, depending on whether the package is already installed
- The second package we install is called `tidyverse`. Accordingly, we can now use `p_load()` from the `pacman` package to to install `tidyverse`:

```
library(pacman)
p_load(tidyverse)
```

The Rtools library

- If you're using Windows, `Rtools` allows you to install other packages faster and more efficiently. To install `Rtools`, now use:

```
p_load(Rtools)
```

- We'll come across much more packages later. Sometimes, you might forget to load a package. In that case, you might get an Error like:

```
Error in function() : could not find function "function"
```

- Which is a clue that you should load the correct package.
- If you use `pacman`, make sure to load `pacman` by `library(pacman)` in each R session
- If you don't, you can just load libraries by e.g. `library(tidyverse)`

The Tinytex Library

- In the exams, we'll be making use of RMarkdown (later more) to generate `.pdf` documents from R code and text.
- In order to generate these `.pdf` files, you need a piece of software called Latex
- You can download a lightweight implementation of latex using R by:
    - Firstly installing the `tinytex` package (`p_load('tinytex')` or `install.packages(tinytex)`)
    - Secondly, using this package to install latex:

```
tinytex::install_tinytex()
```

RMarkdown, R Projects, and Working Directories

RMarkdown

- R Markdown provides an authoring framework for data science. You can use a single R Markdown file to:
  - Save and execute code
  - Generate nice reports that can be shared with an audience
  - Combine text and code in a nice and easy way
  - In RStudio: File > New File > R Markdown..
- Let's watch **this video** as a short introduction to Rmarkdown
- You'll be seeing a lot of Rmarkdown documents in this class
- These slides were also created with Rmarkdown

RMarkdown

- RMarkdown is slightly different from e.g. Microsoft Word, in the sense that you have to **compile** a document
  - This is a deviation from the "What you see is what you get" approach
- Compiling a document is called *knitting* (see the button "Knit" on top of your document)
- Technically, when you **knit** an RMarkdown document, RStudio launches a different, new R session
  - It will run the code from the RMarkdown file from front to back
  - But will not take into account the stuff that is *currently* in memory
  - Remember, you can see what is in memory in the top right window of RStudio
- So be careful, if you have implemented something in the console, but have not written it in RMarkdown
  - Then, RMarkdown doesn't know how to find that something

Working Directories

- In general, computers **organize files and directories like trees**, with a layered structure of folders
    - In Windows, they start with the `root` C (or your hard drive name):
    - E.g. `C://Users/yourname/Documents/R`
- In Mac/Linux, the `root` directory has no name, but can be accessed with a tilde (`~`) sign
    - E.g. `~/Users/yourname/Documents/intro_ads`
- By default, the R **console** takes a directory to be its *reference point*
- Depending on the system, this is usually your document directory, or your home (`~`) directory
- This directory is called your *working directory*

Working Directories

- You can see what **working directory** you are in by running the function `getwd()` without any arguments.
- The working directory is where everything you want to save in R will be saved unless you specify somewhere else
- R will also be able to **access all files** in the working directory (and sub-directories) easily without you needing to know the full file path.

Navigating Directories

- From a particular working directory, you can move to folders up the tree by means of `../`, and down the tree by entering a folders name

- Suppose this is your file tree:

```
/
├── Applications/
│   └── iTunes.app/
├── Users/
│   ├── jones/
│   │   └── Desktop/
│   └── smith/
│       ├── Desktop/
│       │   ├── grades.xls
│       │   ├── manual.bak
│       │   ├── manual.doc
│       │   ├── myDoc.doc
│       │   └── readme.txt
│       ├── Documents/
│       │   ├── essay.doc
│       │   └── report.pdf
│       └── Movies/
├── Volumes/
├── bin/
├── etc/
└── usr/
    ├── lib/
    ├── local/
    │   ├── sbin/
    │   └── bin/
    │       └── prog1.py
    ├── sbin/
    └── share/
```

- E.g. if your working directory is `~/Users/jones/Desktop/`, you can move to applications by `../../../Applications`
- Or if your working directory is `~/bin`, you can move to `local` by `../usr/local`.

Navigating Working Directories

- In sum, there are two ways in which you can tell R (or any other programming language) where you want R to look for files:
    - Specifying the directory from the root (referred to by `C://..` or `~/`)
    - Specifying the directory from your current working directory (referred to by `./`)
- Try using this with the function `list.files` in R
    - `list.files('~')` will give the file in your root directory whereas `list.files('.')` will give the files in your *current* working directory
    - `list.files('./Downloads')` will give the files in the folder Downloads (if there is a folder Downloads in your current working directory)

R Projects

- This is where **R projects** come in, with the file extension `.Rproj`
- A project in RStudio is simply a file which keeps track of the environment and standardize a working directory for a project
  - You should create an R project for this course
  - We will do this in the tutorial
- Then, every time you open Rstudio, you should select File > Open Project
  - Or you could simply go to your File explorer and click the `.Rproj` file: this will launch RStudio
  - Your working directory is now automatically the directory in which the `.Rproj` file is located
- However, note that in `.Rmd` documents by default, the working directory for R code chunks is the directory that contains the Rmd document.
  - For example, if the path of an Rmd file is `~/Downloads/foo.Rmd`, the working directory under which R code chunks are evaluated is `~/Downloads/`.

Working Directories and R Markdown

- In more detail, this means when you **refer to external files** with relative paths *inside* Rmarkdown code chunks, you need to know that these paths are relative to the directory of the `Rmd` file.
- With the aforementioned `.Rmd` example file, `read.csv("data/iris.csv")` in a code chunk means reading the CSV file in `~/Downloads/data/iris.csv`.

- So you in fact have *two* working directories

    - One for in the console (which you can see while looking at your console) and one for the Rmarkdown document
    - In your RMarkdown code chunks, you refer to files in the same folder as e.g.: `./graph1.png`, or `.document3.Rmd`
    - Whereas in the console, your working directory is the directory in which the `.Rproj` file is:
    - So you would access them by e.g. `./assignment1/graph1.png` if your RMarkdown document is in the folder `assignment1`

- We will discuss this during the first tutorial

Recapitulation

Recapitulation

- We talked about various aspects of data science today
  - What is the **purpose** of collecting data?
  - Including importing, transforming (together called tidying), analyzing, and reporting your data
  - And its relationship to causal inference (to *explain* vs. to *predict*)
- We then got to **download R and Rstudio**, and got to know Rstudio
- And finally, we talked about RMarkdown, R projects and working directories
- Next lecture: we'll talk about R programming in more detail!
- Coming up: collecting data
- Finally: voting time for lecture 8

Voting Time

- **5 themes** for final lecture: Schooling on Income, Work Experience on Income, Democracy on Economic Activity, Development on Climate Change, Female Labor and Fertility Rates
- Go to https://www.menti.com/alvf896go6w3
- Or scan this QR Code: