Lecture 2: Introduction to Data - Part 1

LSE ME314: Introduction to Data Science and Machine Learning (https://github.com/me314-lse)

2025-07-15

Daniel de Kadt

Slides last updated: 14 July 2025

What is Data?

Data vs. Information

Data: Abstractions, representations, symbols, values describing qualities or quantities. $\rightarrow$ E.g. “All” “the” “world’s” “a” “stage”

Data vs. Information

Data: Abstractions, representations, symbols, values describing qualities or quantities. $\rightarrow$ E.g. "All" "the" "world's" "a" "stage"

Information: Signals that contain something systematic (not just noise) $\rightarrow$ E.g. "All the world's a stage"

Data vs. Information

Data: Abstractions, representations, symbols, values describing qualities or quantities. $\rightarrow$ E.g. "All" "the" "world's" "a" "stage"

Information: Signals that contain something systematic (not just noise) $\rightarrow$ E.g. "All the world's a stage"

Data is a representation of underlying information.

Raw data may or may not contain interpretable information. Invariably, we need to *do things to and with data* to extract signal and information.

Different Types of Data

Numerical vs. Quantitative vs. Qualitative

Pretty much everything we do in this course will relate to **numerical** data: Data represented as numbers.

But this does not mean that everything is **quantitative**!

Numerical vs. Quantitative vs. Qualitative

Pretty much everything we do in this course will relate to **numerical** data: Data represented as numbers.

But this does not mean that everything is **quantitative**!

Quantitative: Capturing a *quantity* as a continuous or discrete variable

Numerical vs. Quantitative vs. Qualitative

Pretty much everything we do in this course will relate to **numerical** data: Data represented as numbers.

But this does not mean that everything is **quantitative**!

Quantitative: Capturing a *quantity* as a continuous or discrete variable

Qualitative: Capturing a *quality* as a categorical variable

Numerical vs. Quantitative vs. Qualitative

Pretty much everything we do in this course will relate to **numerical** data: Data represented as numbers.

But this does not mean that everything is **quantitative**!

Quantitative: Capturing a *quantity* as a continuous or discrete variable

Qualitative: Capturing a *quality* as a categorical variable

For our purposes, almost all data will need to be converted into **numerical** data

Peaches

Source: The Today Show, Peach Benefits

Quantitative Data as Numerical Data: Example

```r
# Quantitative data about peach varieties:
peach_counts = data.frame(
  variety = c("Donut", "Redhaven", "Nectarine", "White"),
  count = c(150, 200, 100, 120),
  gbp_per_lb = c(2, 3, 3.5, 2.5)
)

# Easy to calculate the total revenue from each variety:
peach_counts %>%
  mutate(revenue = count * gbp_per_lb) %>%
  select(variety, revenue) %>%
  arrange(desc(revenue))
```

```
    variety revenue
1  Redhaven     600
2 Nectarine     350
3     Donut     300
4     White     300
```

```r
# Save our data as a .csv:
write.csv(peach_counts, "../data/peach_counts.csv", row.names = FALSE)
```

Qualitative Data as Numerical Data: Example

```r
# Qualitative data about peach varieties:
peach_features = data.frame(
  variety = c("Donut", "Redhaven", "Nectarine", "White"),
  color = c("Yellow", "Red", "Red", "White"),
  size = c("Small", "Large", "Medium", "Large"),
  taste = c("Tangy", "Sweet", "Sweet", "Sweet"),
  fuzziness = c("Fuzzy", "Fuzzy", "Smooth", "Fuzzy")
)

peach_features
```

```
    variety  color   size taste fuzziness
1     Donut Yellow  Small Tangy     Fuzzy
2  Redhaven    Red  Large Sweet     Fuzzy
3 Nectarine    Red Medium Sweet    Smooth
4     White  White  Large Sweet     Fuzzy
```

Qualitative Data as Numerical Data: Example

```r
# Convert the categorical features into numerical data:
# column 1 is the variety, then a series of dummies for features:
peach_features %>%
  mutate(
    color_Yellow = ifelse(color == "Yellow", 1, 0),
    color_Red = ifelse(color == "Red", 1, 0),
    color_White = ifelse(color == "White", 1, 0),
    size_Medium = ifelse(size == "Medium", 1, 0),
    size_Large = ifelse(size == "Large", 1, 0),
    size_Small = ifelse(size == "Small", 1, 0),
    taste_Sweet = ifelse(taste == "Sweet", 1, 0),
    taste_Tangy = ifelse(taste == "Tangy", 1, 0),
    fuzziness_Fuzzy = ifelse(fuzziness == "Fuzzy", 1, 0),
    fuzziness_Smooth = ifelse(fuzziness == "Smooth", 1, 0)
  ) %>%
  select(-color, -size, -taste, -fuzziness) # remove original categorical columns
```

```
    variety color_Yellow color_Red color_White size_Medium size_Large
1     Donut            1         0           0           0          0
2  Redhaven            0         1           0           0          1
3 Nectarine            0         1           0           1          0
4     White            0         0           1           0          1
  size_Small taste_Sweet taste_Tangy fuzziness_Fuzzy fuzziness_Smooth
1          1           0           1               1                0
2          0           1           0               1                0
3          0           1           0               0                1
4          0           1           0               1                0
```

Unstructured Data as Numerical Data: Example

Benefits of peaches

Nutrients that can aid in heart health, gut health and immune function are all found in peaches.

For starters, the fiber in peaches — including both soluble and insoluble fiber — provides health benefits. "Soluble fiber can stabilize blood sugars and keeps cholesterol level in check," Zumpano says, "and then insoluble fiber more aids in digestion and helps prevent constipation."

Most of us in the U.S. get far below the recommended 25 to 40 grams of fiber per day, Derocha adds, and peaches can be a delicious way to get a fiber boost.

Peaches have "a good amount of potassium," Derocha says, which is "a mineral that helps regulate blood pressure and helps with muscle and nerve function." Animal studies have also shown that peach extract may help lower cholesterol and blood pressure, Zumpano says.

Finally, peaches contain a good dose of vitamin C, which can support your immune system, Derocha says. While other fruits (like strawberries and kiwi) contain more, the vitamin C in peaches is a nice bonus in a fruit already packed with other nutrients.

Source: The Today Show, Peach Benefits

Unstructured Data as Numerical Data: Example

```python
# Take the paragraph from the Today Show about the benefits of peaches:
peach_paragraph = (
"Nutrients that can aid in heart health, gut health and immune function are all found in peaches.

For starters, the fiber in peaches - including both soluble and insoluble fiber -
provides health benefits.
'Soluble fiber can stabilize blood sugars and keeps cholesterol level in check,'
Zumpano says, 'and then insoluble fiber more aids in digestion and helps prevent constipation.'

Most of us in the U.S. get far below the recommended 25 to 40 grams of fiber per day,
Derocha adds, and peaches can be a delicious way to get a fiber boost.

Peaches have 'a good amount of potassium,' Derocha says, which is
'a mineral that helps regulate blood pressure and helps with muscle and nerve function.'
Animal studies have also shown that peach extract may help lower cholesterol and blood pressure,
Zumpano says.

Finally, peaches contain a good dose of vitamin C, which can support your immune system,
Derocha says.
While other fruits (like strawberries and kiwi) contain more,
the vitamin C in peaches is a nice bonus in a fruit already packed with other nutrients."
)
```

Unstructured Data as Numerical Data: Example

```
library(stringr)

# Split the paragraph up into a vector of unique words with counts
# and remove punctuation and paragraph breaks:
peach_paragraph %>%
  # remove punctuation and paragraphs
  str_remove_all('[[:punct:][:digit:]\r\n]') %>%
  # remove stopwords
  str_remove_all('\\b(the|and|a|to|in|of|is|that|can|for|with|as|it|are|this|by|on|from)\\b') %>%
  # remove any excess white space
  str_squish() %>%
  strsplit(" ") %>%
  unlist() %>%
  tolower() %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(20)
```

```
.
      fiber     peaches       blood     derocha      health       helps
          6           4           3           3           3           3
       says           c cholesterol     contain    function         get
          3           2           2           2           2           2
       good        have      immune   insoluble        more   nutrients
          2           2           2           2           2           2
      other    pressure
          2           2
```

Rectangular (‘Tabular’) Data

Units and Observations

In rectangular data, rows are observations.

Units and Observations

In rectangular data, rows are observations.

Units: The entities or subjects being studied (e.g., individuals, countries, companies).

Observations: A single “peek” at a unit under specific conditions (e.g., time period).

Units and Observations

In rectangular data, rows are observations.

Units: The entities or subjects being studied (e.g., individuals, countries, companies).

Observations: A single "peek" at a unit under specific conditions (e.g., time period).

- Sometimes units = observations $\rightarrow$ *cross-sectional* datasets:
  - Take a slice (cross-section) at a single point in time
  - Can be done repeatedly ('repeated cross-sections')

Units and Observations

In rectangular data, rows are observations.

Units: The entities or subjects being studied (e.g., individuals, countries, companies).

Observations: A single “peek” at a unit under specific conditions (e.g., time period).

- Sometimes units = observations $\rightarrow$ *cross-sectional* datasets:
  - Take a slice (cross-section) at a single point in time
  - Can be done repeatedly (‘repeated cross-sections’)

- Often units $\neq$ observations $\rightarrow$ *hierarchical* datasets:
  - Each unit can have multiple observations (e.g., repeated measures, time series)
  - Can be done across different units (e.g., countries, individuals)

Units and Observations

In rectangular data, rows are observations.

Units: The entities or subjects being studied (e.g., individuals, countries, companies).

Observations: A single "peek" at a unit under specific conditions (e.g., time period).

- Sometimes units = observations $\rightarrow$ *cross-sectional* datasets:
  - Take a slice (cross-section) at a single point in time
  - Can be done repeatedly ('repeated cross-sections')
- Often units $\neq$ observations $\rightarrow$ *hierarchical* datasets:
  - Each unit can have multiple observations (e.g., repeated measures, time series)
  - Can be done across different units (e.g., countries, individuals)

This is a really important distinction (we'll come back to it later)

Variables or Features

The columns of a rectangular dataset typically indicate 'variables' or 'features.'

Features: Attributes of a unit given the observation.

All columns are variables or features; they could quantitative or qualitative or merely identifiers.

Variables or Features

The columns of a rectangular dataset typically indicate 'variables' or 'features.'

Features: Attributes of a unit given the observation.

All columns are variables or features; they could quantitative or qualitative or merely identifiers.

To this point, all rectangular datasets should have a **primary key**: A variable which uniquely identifies each observation.

This could be implicit – a combination of two or more variables – or explicit, like a unique ID.

This is **extremely important** – we will come back to it.

Numerical Variables: Continuous vs. Discrete

We've actually already seen that numerical data, like peaches, can come in many different flavours:

1. **Continuous**: Can take any value within a range:
   - ➜ **Interval**: Differences have understood meaning, but no absolute zero (e.g., time picked).
   - ➜ **Ratio**: Differences have understood meaning but there is an absolute zero (e.g., weight).

Numerical Variables: Continuous vs. Discrete

We've actually already seen that numerical data, like peaches, can come in many different flavours:

1. **Continuous**: Can take any value within a range:
   - ➜ **Interval**: Differences have understood meaning, but no absolute zero (e.g., time picked).
   - ➜ **Ratio**: Differences have understood meaning but there is an absolute zero (e.g., weight).

2. **Discrete**: Can only take specific values, often whole numbers:
   - ➜ **Count**: Non-negative integers representing the number of occurrences (e.g., number of peaches).
   - ➜ **Ordinal**: A meaningful order but no consistent meaningful difference between values (e.g., rankings).
   - ➜ **Nominal**: Categories with no inherent order (e.g., varieties of peach).
   - ➜ **Binary**: A special case of nominal data with only two categories (e.g. red peach).

Putting it all Together: ‘Tidy’ Data

Tidy data is data where:

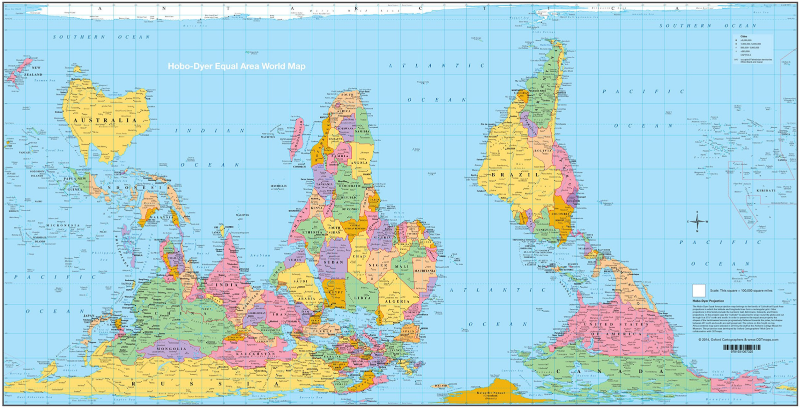
1. Each observation is a row
2. Each variable is a column
3. Each cell is a value

variables observations values

Source: Hadley Wickham, Data Tidying

Example: Geographic Data as Tidy Data

Example: Geographic Data as Tidy Data



Example: Geographic Data as Tidy Data

```
library(sf)

# read in shapefile:
shp <- st_read(".\\data\\ne_10m_admin_0_countries\\ne_10m_admin_0_countries.shp")
head(shp$geometry)
```

```
Geometry set for 6 features
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -109.4537 ymin: -55.9185 xmax: 140.9776 ymax: 7.35578
Geodetic CRS:  WGS 84
First 5 geometries:
```

```
MULTIPOLYGON (((117.7036 4.163415, 117.7036 4.1...
MULTIPOLYGON (((117.7036 4.163415, 117.6971 4.1...
MULTIPOLYGON (((-69.51009 -17.50659, -69.50611 ...
MULTIPOLYGON (((-69.51009 -17.50659, -69.51009 ...
MULTIPOLYGON (((-69.51009 -17.50659, -69.63832 ...
```

Geographic Data as Tidy Data

```r
library(sf)

# read in shapefile:
shp <- st_read(".\\data\\ne_10m_admin_0_countries\\ne_10m_admin_0_countries.shp")
shp$geometry[1][[1]]
```

```
MULTIPOLYGON (((117.7036 4.163415, 117.7036 4.163415, 117.7381 4.157242, 117.7836 4.157242, 117.8526 4.157242,
4.156683, 117.9121 4.144924, 117.9182 4.100043, 117.9348 4.059882, 117.9014 4.036689, 117.8871 4.031928, 117.87
4.031562, 117.8386 4.040188, 117.8191 4.068671, 117.7627 4.100735, 117.7394 4.13231, 117.7036 4.163415)), ((124
-9.18019, 124.4658 -9.179376, 124.5178 -9.17783, 124.5632 -9.170831, 124.6043 -9.156345, 124.668 -9.113051, 124
-9.107517, 124.6732 -9.099298, 124.6836 -9.074802, 124.688 -9.069513, 124.6916 -9.067804, 124.7581 -9.047947, 1
-9.042901, 124.7888 -9.018976, 124.825 -9.008071, 124.8403 -9.001072, 124.8699 -8.994399, 124.8971 -8.974542, 1
-8.962091, 124.9195 -8.962016, 124.9225 -8.986324, 124.9092 -9.020327, 124.9079 -9.037483, 124.913 -9.05278, 12
-9.065492, 124.933 -9.074484, 124.9479 -9.078515, 124.9659 -9.077274, 124.9789 -9.072003, 125.0044 -9.05557, 12
-9.035003, 125.0653 -9.023221, 125.0686 -9.015883, 125.073 -8.996452, 125.0765 -8.990768, 125.0868 -8.98622, 12
-8.988598, 125.0953 -8.994799, 125.1025 -9.00131, 125.1213 -9.011852, 125.1394 -9.024771, 125.154 -9.040791, 12
-9.060634, 125.162 -9.082339, 125.145 -9.145177, 125.147 -9.154479, 125.1512 -9.165021, 125.1527 -9.170824, 125
-9.17577, 125.1506 -9.185381, 125.1425 -9.189102, 125.1179 -9.187552, 125.107 -9.188482, 125.0915 -9.19675, 125
-9.205535, 125.0739 -9.211323, 125.0597 -9.210496, 125.0534 -9.206672, 125.0414 -9.193236, 125.0345 -9.188172,
-9.186312, 125.0072 -9.186105, 124.9745 -9.191893, 124.9579 -9.211943, 124.9541 -9.241502, 124.9602 -9.276022,
-9.293385, 124.9728 -9.304651, 124.9967 -9.325838, 125.0179 -9.338964, 125.0234 -9.345578, 125.0248 -9.351986,
-9.367696, 125.0238 -9.374724, 125.0463 -9.409864, 125.0509 -9.424697, 125.0616 -9.485772, 125.0613 -9.486017,
-9.507501, 125.002 -9.53631, 124.9866 -9.577325, 124.9797 -9.649021, 124.9707 -9.662367, 124.8748 -9.724705, 12
```

Geographic Data as Tidy Data

```
library(sf)

# read in shapefile:
shp <- st_read(".\\data\\ne_10m_admin_0_countries\\ne_10m_admin_0_countries.shp")
head(shp)
```

```
Simple feature collection with 6 features and 168 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -109.4537 ymin: -55.9185 xmax: 140.9776 ymax: 7.35578
Geodetic CRS:  WGS 84
       featurecla scalerank LABELRANK SOVEREIGNT SOV_A3 ADM0_DIF LEVEL              TYPE TLC     ADMIN ADM0_A3
1 Admin-0 country         0         2  Indonesia    IDN        0     2 Sovereign country   1 Indonesia     IDN
2 Admin-0 country         0         3   Malaysia    MYS        0     2 Sovereign country   1  Malaysia     MYS
3 Admin-0 country         0         2      Chile    CHL        0     2 Sovereign country   1     Chile     CHL
4 Admin-0 country         0         3    Bolivia    BOL        0     2 Sovereign country   1   Bolivia     BOL
5 Admin-0 country         0         2       Peru    PER        0     2 Sovereign country   1      Peru     PER
```

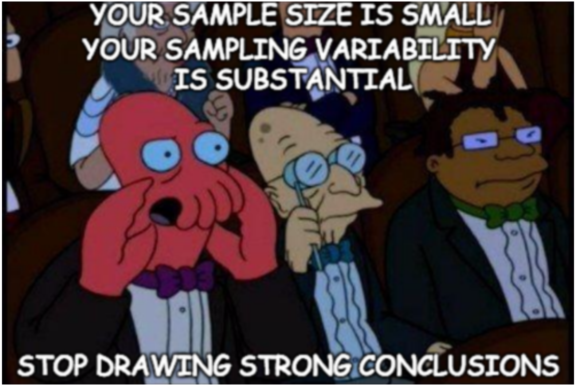
Aside: ‘Big’ Data

A brief but important aside: You’ve probably heard of ‘big’ data.

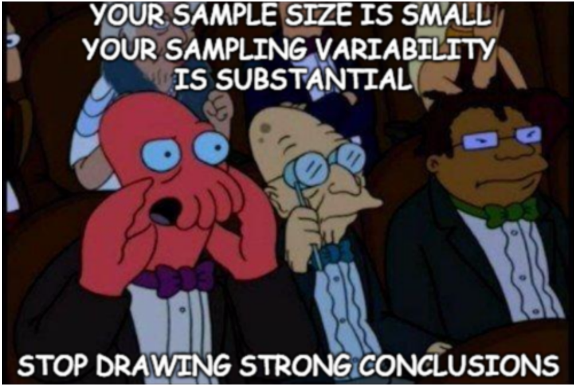
Big data ‘as large n’: a very large number of observations

Big data ‘as large p’: a very large number of variables/features

Aside: 'Big' Data



Aside: ‘Big’ Data



Aside: Databases

In most professional settings, you will get data from a **database system**.

Database system: an organized collection of data that is stored and accessed via a computer

- The way a database is organized is called a **schema**
- Since a database is used for data *storage*, a user typically "reads" and "writes" to a database based on permissions
- Read and write via **queries**
- Queries are often constructed/written in **domain-specific languages** like SQL, but not always
- A user can typically read and write via `R` (or `python`)

Aside: Relational Databases

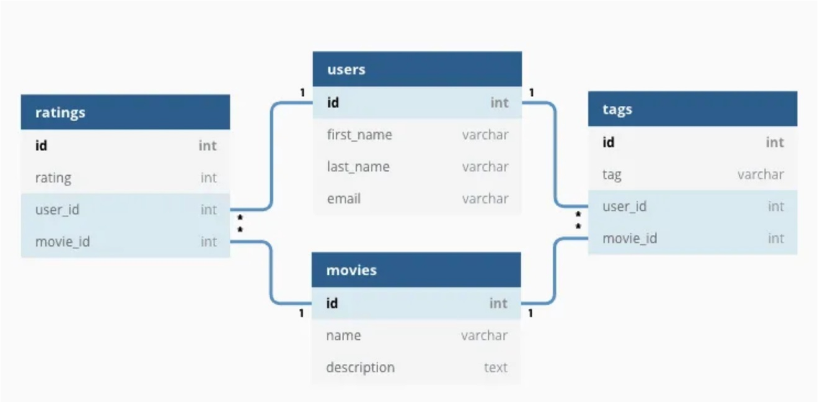
Relational databases

- Data is stored in multiple tables to avoid redundancy
- Tables are linked based on common **keys**
- SQL is used to access data

Non-relational databases

- Data stored in a way that is not based on tabular relations (e.g. MongoDB uses JSON like documents)
- Data is accessed using a wide variety of (sometimes customised) languages

Aside: Relational Databases



Source: Ethan Duong, Relational Databases

Reading and Storing Data

Data Building Blocks in `R` and `python`

In both `R` and `python`, data is stored in objects.

However, they differ in a key respect: ‘mutability’

- `R` objects are *immutable*: once created, they cannot be changed in place
- `R` uses a copy-on-modify logic – if you change an object, a new object is created in memory
- In `python` some objects are *immutable* while others are *mutable*: they can be changed in place

Data in R and python

The basic data objects types in `R` and `python` are similar, but confusingly different:

R	Python	R Example	Python Example	Mutable (Py)?
Numeric	**float**	`x <- 3.14`	`x = 3.14`	No
Integer	**int**	`x <- 42L`	`x = 42`	No
Character	**str**	`x <- "hello"`	`x = "hello"`	No
Logical	**bool**	`x <- TRUE`	`x = True`	No
Vector	**list**	`x <- c(1, 2, 3)`	`x = [1, 2, 3]`	Yes
List	**list**	`x <- list(1, "a")`	`x = [1, "a"]`	Yes
—	**tuple**	—	`x = (1, "a")`	No
Named List	**dict**	`list(a=1, b=2)`	`{"a": 1, "b": 2}`	Yes

Remember, in `R` none of these are mutable! Everything is copy-on-modify.

Data in R and python

The basic data objects types in R and python are similar, but confusingly different:

R	Python	R Example	Python Example	Mutable (Py)?
Numeric	**float**	`x <- 3.14`	`x = 3.14`	No
Integer	**int**	`x <- 42L`	`x = 42`	No
Character	**str**	`x <- "hello"`	`x = "hello"`	No
Logical	**bool**	`x <- TRUE`	`x = True`	No
Vector	**list**	`x <- c(1, 2, 3)`	`x = [1, 2, 3]`	Yes
List	**list**	`x <- list(1, "a")`	`x = [1, "a"]`	Yes
—	**tuple**	—	`x = (1, "a")`	No
Named List	**dict**	`list(a=1, b=2)`	`{"a": 1, "b": 2}`	Yes

Remember, in R none of these are mutable! Everything is copy-on-modify. **Note**: most of these objects are not rectangular data objects! We'll get there...

(Im)mutability in Practice: R

Consider two names bound to the same object:

```r
x <- c(1,2,3)
y <- x
address(y) == address(x)
```

```
[1] TRUE
```

(Im)mutability in Practice: R

Consider two names bound to the same object:

```r
x <- c(1,2,3)
y <- x
address(y) == address(x)
```

```
[1] TRUE
```

Now modify `x`:

```r
x[3] <- 4
y
```

```
[1] 1 2 3
```

(Im)mutability in Practice: R

Consider two names bound to the same object:

```r
x <- c(1,2,3)
y <- x
address(y) == address(x)
```

```
[1] TRUE
```

Now modify `x`:

```r
x[3] <- 4
y
```

```
[1] 1 2 3
```

```r
address(y) == address(x)
```

```
[1] FALSE
```

(Im)mutability in Practice: `python`

Consider a string in `python`, which is immutable:

```
x = "hello"
y = x
id(x) == id(y)
```

```
True
```

(Im)mutability in Practice: `python`

Consider a string in `python`, which is immutable:

```
x = "hello"
y = x
id(x) == id(y)
```

```
True
```

Modify `x`:

```
x = "world"
y
```

```
'hello'
```

(Im)mutability in Practice: python

Consider a string in python, which is immutable:

```
x = "hello"
y = x
id(x) == id(y)
```

```
True
```

Modify x:

```
x = "world"
y
```

```
'hello'
```

```
id(x) == id(y)
```

```
False
```

(Im)mutability in Practice: python

Now consider a list in python, which is mutable:

```
x = [1, 2, 3]
y = x
id(x) == id(y)
```

```
True
```

(Im)mutability in Practice: python

Now consider a list in `python`, which is mutable:

```
x = [1, 2, 3]
y = x
id(x) == id(y)
```

```
True
```

Modify `x`:

```
x[2] = 4
y
```

```
[1, 2, 4]
```

(Im)mutability in Practice: python

Now consider a list in python, which is mutable:

```
x = [1, 2, 3]
y = x
id(x) == id(y)
```

```
True
```

Modify x:

```
x[2] = 4
y
```

```
[1, 2, 4]
```

```
id(x) == id(y)
```

```
True
```

Rectangular Data Objects in `R` and `python`

Eventually we want to work with 'rectangular' data objects:

Concept	R Example	Python Example
Matrix	`matrix(1:6, nrow=2)`	`np.array([[1, 2, 3], [4, 5, 6]])`
Array	`array(1:8, dim=c(2,2,2))`	`np.array([[[1,2], [3,4]], [[5,6], [7,8]]])`
Data Frame	`data.frame(a=1:3, b=4:6)`	`pd.DataFrame({"a": [1,2,3], "b": [4,5,6]})`

Remember, you will need to `import pandas as pd` and `import numpy as np` for the above to work in `python`.

File Formats for Rectangular Data

Common file formats for storing tabular data:

- Comma-separated values (.csv) – ubiquitous and simple
  - Each *line* is an observation
  - Each variable value is separated by a comma
- Application specific (proprietary) formats (.dta, .sav, .xlsx etc.)
  - Can allow for richer representations including meta-data
  - More complex, and not necessarily human-readable

Often choice is dictated by the source (and size) of the data

Packages like `haven` in `R`, and `pandas` and `pyreadstat` in `python` allow for reading in non-csv formats.

Ingesting Data in R

Recall that we saved `peach_counts.csv` a while back.

In base R we can ingest that `.csv` using the function `read.csv()`:

```
peach_counts <- read.csv("../data/peach_counts.csv",
         header = TRUE,
         stringsAsFactors = FALSE)
head(peach_counts)
```

```
   variety count gbp_per_lb
1     Donut   150        2.0
2  Redhaven   200        3.0
3 Nectarine   100        3.5
4     White   120        2.5
```

We could also use `read_csv()` for a tidy implementation.

Ingesting Data in python

In native python things are a little different:

```
with open("../data/peach_counts.csv", 'r') as file:
    lines = file.readlines()

print(lines)
```

```
['"variety","count","gbp_per_lb"\n', '"Donut",150,2\n', '"F
```

Ingesting Data in python

Much easier is to use pandas, which will behave more like R:

```python
import pandas as pd

peach_counts = pd.read_csv("../data/peach_counts.csv")

print(peach_counts.head())
```

```
     variety  count  gbp_per_lb
0      Donut    150         2.0
1  Redhaven    200         3.0
2  Nectarine    100         3.5
3      White    120         2.5
```

Transforming and Manipulating Data

Wrangling Data in Base R

To work with data in base R, we will typically have to manipulate objects directly:

```
# add a new variable
peach_counts$revenue <- peach_counts$count * peach_counts$gbp_per_lb
# keep only these columns
peach_counts <- peach_counts[, c("variety", "revenue")]
# sort by revenue in descending order
peach_counts <- peach_counts[order(-peach_counts$revenue), ]

head(peach_counts)
```

```
    variety revenue
2  Redhaven     600
3 Nectarine     350
1     Donut     300
4     White     300
```

Wrangling Data in Tidy R

Tidy R gives us an alternative approach.

{dplyr} gives us useful and literal tools for wrangling data in R:

- `mutate()`: Add or modify variables in a data frame
- `select()`: Choose specific columns from a data frame
- `filter()`: Subset rows based on conditions
- `arrange()`: Sort rows by one or more variables
- and many more (also see other tidyverse packages)

Wrangling Data in Tidy R

Using the pipe %>% or |> (native pipe) allows us to chain operations:

```
peach_counts <- read.csv("../data/peach_counts.csv",
         header = TRUE,
         stringsAsFactors = FALSE)

peach_counts %>%
  # add a new variable
  mutate(revenue = count * gbp_per_lb) %>%
  # keep only these columns
  select(variety, revenue) %>%
  # sort by revenue in descending order
  arrange(desc(revenue))
```

```
    variety revenue
1  Redhaven     600
2 Nectarine     350
3     Donut     300
4     White     300
```

Grouping and Hierarchies

Sometimes data has a nested structure.

This can occur in different ways:

1. Repeated observations of the same units:
   - ➜ each observation is *nested* under a single unit
2. Hierarchical data:
   - ➜ each unit is *nested* under a higher-level unit (cluster)
3. Binned data:
   - ➜ each observation is *nested* under a bin based on some variable

We may sometimes want to restructure our data given that hierarchy.

Grouping in R (Tidyverse)

Consider data where each *unit* belongs to some hierarchy:

```r
# Example dataset (unique peach = unit)
peach_weights <- data.frame(
  variety = sample(c("Donut", "Redhaven", "Nectarine", "White"),
                   100, replace = TRUE),
  weight = runif(100, 0.5, 1.5)
)

head(peach_weights)
```

```
    variety    weight
1     Donut 1.4168867
2     Donut 1.1818110
3 Nectarine 0.8919321
4 Nectarine 0.5792407
5     Donut 0.9437527
6  Redhaven 0.9334612
```

Grouping in R (Tidyverse)

We can group by a higher variable and summarise across that variable:

```
# Group by 'variety' and summarise:
peach_weights %>%
  group_by(variety) %>%             # variable on which to group
  summarise(                        # summarise across grouping var
    count = n(),                    # counts observations
    total_weight = sum(weight),     # apply a function (sum(x))
    mean_weight = mean(weight)      # a different function (mean(x))
  )
```

```
# A tibble: 4 x 4
  variety   count total_weight mean_weight
  <chr>     <int>        <dbl>       <dbl>
1 Donut        26         24.3       0.933
2 Nectarine    26         26.2       1.01
3 Redhaven     27         27.1       1.00
4 White        21         21.8       1.04
```

Reshaping in R

Now consider data with multiple *observations* per *unit* (e.g., yield per peach variety over time):

```
# Example dataset (peach variety = unit, observed over 25 time periods)
peach_panel <- data.frame(
  variety = rep(c("Donut", "Redhaven", "Nectarine", "White"), 25),
  year = rep(2000:2024, each = 4),
  yield = runif(100, 50, 200)
)

head(peach_panel)
```

```
    variety year    yield
1     Donut 2000 119.60066
2  Redhaven 2000  67.42527
3 Nectarine 2000  57.41907
4     White 2000 164.69897
5     Donut 2001  87.92254
6  Redhaven 2001 191.99072
```

Reshaping in R

Let's reshape our data from long to wide using {tidyr}:

```
# Reshape from long to wide:
peach_panel_wide <- peach_panel %>%
  pivot_wider(
    names_from = year,            # variable to use as column names
    values_from = yield,          # variable to use as values
    values_fill = list(yield = NA) # fill missing values with NA
  )

head(peach_panel_wide)
```

```
# A tibble: 4 x 26
  variety  `2000` `2001` `2002` `2003` `2004` `2005` `2006` `2007` `2008` `2009
  <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl
1 Donut     120.    87.9   111.  162.   146.    50.9  151.   129.    65.5   77.
2 Redhaven   67.4  192.    177.  151.   151.   156.   174.    77.7  185.    79.
3 Nectari~   57.4  147.    171.   61.3   99.9   80.9   54.1  106.   195.   134.
4 White     165.   136.    141.  199.   126.   107.   180.   165.   182.   172.
# i 15 more variables: `2010` <dbl>, `2011` <dbl>, `2012` <dbl>, `2013` <dbl>,
#   `2014` <dbl>, `2015` <dbl>, `2016` <dbl>, `2017` <dbl>, `2018` <dbl>,
#   `2019` <dbl>, `2020` <dbl>, `2021` <dbl>, `2022` <dbl>, `2023` <dbl>,
#   `2024` <dbl>
```

Reshaping in R

There and back again (back to long format):

```
# Reshape from long to wide:
peach_panel_wide %>%
  pivot_longer(
    cols = -variety,                  # keep 'variety' as is
    names_to = "year",                # new column for years
    values_to = "yield"               # new column for yield values
  ) %>%
  head()
```

```
# A tibble: 6 x 3
  variety year  yield
  <chr>   <chr> <dbl>
1 Donut   2000  120.
2 Donut   2001   87.9
3 Donut   2002  111.
4 Donut   2003  162.
5 Donut   2004  146.
6 Donut   2005   50.9
```

Merges and Joins in `R`

Recall our relational database setting – we had multiple tables connected by keys.

In `R`, we can merge (join) a left object and a right object together using the same principles:

- **Inner join**: Keep only rows with matching keys in both tables
- **Left join**: Keep all rows from the left table, and matching rows from the right table
- **Right join**: Keep all rows from the right table, and matching rows from the left table
- **Full join**: Keep all rows from both tables

Merges and Joins in R

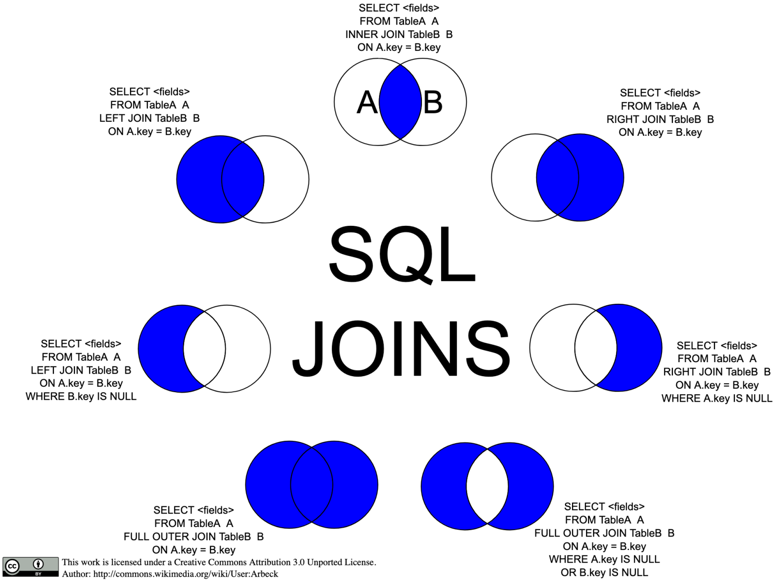
Recall our relational database setting – we had multiple tables connected by keys.

In R, we can merge (join) a left object and a right object together using the same principles:

- **Inner join**: Keep only rows with matching keys in both tables
- **Left join**: Keep all rows from the left table, and matching rows from the right table
- **Right join**: Keep all rows from the right table, and matching rows from the left table
- **Full join**: Keep all rows from both tables

It is very wise to **check** your data after merges. Bad merges can create **very bad** outcomes.

Merges and Joins: {dplyr} and SQL



Source: Wikipedia, Join (SQL)

Merges and Joins in R

Recall our earlier peach_features object? Let's join it to our peach_counts object.

Our **key** is the variety variable, present in both objects:

```
# Join peach_counts to peach_features:
peach_joined <- peach_counts %>%
  # join on 'variety'
  left_join(peach_features, by = "variety")

peach_joined
```

```
    variety count gbp_per_lb  color   size taste fuzziness
1     Donut   150        2.0 Yellow  Small Tangy     Fuzzy
2  Redhaven   200        3.0    Red  Large Sweet     Fuzzy
3 Nectarine   100        3.5    Red Medium Sweet    Smooth
4     White   120        2.5  White  Large Sweet     Fuzzy
```

Summary and Next Steps

Today we covered:

1. What is data?
2. Different kinds of data
3. Working with data in `R` (and `python`)

This afternoon you will practice 2 and 3.

Tomorrow:

1. Data generating processes
2. Probability
3. Visualisation of data in `R`