

Week 1: Data Wrangling and EDA

Table of contents

1	The Dataset	4
2	Part A: First Contact with the Dataset	4
3	A1. Loading and Inspecting the Dataset	4
3.1	R: Load and Inspect the Data	4
3.1.1	Load required packages	4
3.1.2	Read the dataset	4
3.1.3	Display first few rows	5
3.1.4	Dimensions of the dataset	5
3.1.5	Column names	5
3.2	Python: Load and Inspect the Data	6
3.2.1	Read the dataset	6
3.2.2	Display first few rows	6
3.2.3	Dimensions of the dataset	7
3.2.4	Column names	7
4	A2. What Does One Row Represent?	7
4.1	Interpretation	7
5	A3. Identifying Groups of Variables	8
5.1	Client-Related Variables	8
5.2	Campaign / Contact-Related Variables	9
5.3	Economic Context Variables	9
5.4	Outcome Variable	9
6	A4. Initial Structural Observations	10
7	Key Takeaway from Part A	10
8	Part B: Data Types & Representation	10

9 B1. Why Data Types Matter	11
10 B2. Inspecting Software-Inferred Data Types	11
10.0.1 R: Inspect Column Types	11
10.0.2 Python: Inspect Column Types	12
11 B3. Conceptual Classification of Variables	13
11.1 Client-Related Variables	13
11.2 Campaign / Contact Variables	13
11.3 Economic Context Variables	14
11.4 Outcome Variable	14
12 B4. Numeric Does Not Always Mean Quantitative	15
12.1 Example 1: pdays	15
12.1.1 R: Inspect pdays	15
12.1.2 Python: Inspect pdays	16
12.2 Example 2: duration	16
13 B5. Ordinal vs Nominal: A Subtle Distinction	17
13.1 Example: education	17
14 B6. Representation Is a Modelling Decision	17
15 Key Takeaway from Part B	18
16 Part C: Data Quality & Validity Checks	18
17 C1. Why Data Quality Comes Before Modelling	18
18 C2. Missing Values vs “Unknown” Values	18
18.0.1 R: Check for Missing Values	19
18.0.2 Python: Check for Missing Values	19
19 C3. Identifying “Unknown” Categories	20
19.0.1 R: Frequency of “unknown” Values	20
19.0.2 Python: Frequency of “unknown” Values	21
20 C4. Sentinel Values and Coded Entries	21
20.1 Example 1: pdays	21
20.1.1 R: Examine Distribution of pdays	21
20.1.2 Python: Examine Distribution of pdays	22
20.2 Example 2: campaign and previous	23
21 C5. Variables That Threaten Validity	23
21.1 Critical Example: duration	23

22 C6. Classifying Data Quality Issues	23
22.1 Issue 1: "unknown" in education	24
22.2 Issue 2: pdays = 999	24
22.3 Issue 3: duration	24
23 C7. Should Everything Be “Cleaned”?	25
24 C8. Data Quality Is Question-Dependent	25
25 Key Takeaway from Part C	26
26 Part D: Exploratory Data Analysis (EDA)	26
27 D1. Purpose of Exploratory Data Analysis	26
28 D2. Exploring the Outcome Variable (y)	26
28.0.1 R: Distribution of y	27
28.0.2 Python: Distribution of y	27
29 D3. Exploratory Analysis of Categorical Variables	28
29.1 Example 1: Job Type vs Subscription	28
29.1.1 R: Job vs Outcome	28
29.1.2 Python: Job vs Outcome	29
29.2 Example 2: Contact Method vs Outcome	30
29.3 R: Contact vs Outcome	30
29.4 Python: Contact vs Outcome	30
30 D4. Exploratory Analysis of Numeric Variables	31
30.1 Example 1: Age Distribution	31
30.1.1 R: Age Distribution	31
30.1.2 Python: Age Distribution	32
30.2 Example 2: Duration and Outcome (with Caution)	33
30.2.1 R: Duration by Outcome	33
30.2.2 Python: Duration by Outcome	34
31 D5. Multivariate EDA: Combining Variables	34
31.1 Example: Age vs Outcome by Contact Method	35
31.1.1 R	35
31.1.2 Python	35
32 D6. What EDA Can — and Cannot — Tell Us	36

1 The Dataset

We'll use “[UCI Bank Marketing](#)” dataset to demonstrate week 1 concepts.

The Bank Marketing dataset is a real-world dataset from the UCI Machine Learning Repository about direct marketing campaigns carried out by a Portuguese bank. The bank used telephone calls to contact potential customers and offer them a term deposit product — a kind of savings or investment account.

```
library(reticulate)
use_python("C:/Program Files/Python313/python.exe", required = TRUE)
```

2 Part A: First Contact with the Dataset

3 A1. Loading and Inspecting the Dataset

We begin by loading the dataset and performing a **minimal structural inspection**. At this stage, the goal is *not* analysis, but orientation.

3.1 R: Load and Inspect the Data

3.1.1 Load required packages

```
library(tidyverse)
```

3.1.2 Read the dataset

```
bank <- read.csv("data/raw/bank-additional.csv", sep = ";",
stringsAsFactors = FALSE)
```

3.1.3 Display first few rows

```
head(bank)
```

	age	job	marital	education	default	housing	loan	contact
1	30	blue-collar	married	basic.9y	no	yes	no	cellular
2	39	services	single	high.school	no	no	no	telephone
3	25	services	married	high.school	no	yes	no	telephone
4	38	services	married	basic.9y	no	unknown	unknown	telephone
5	47	admin.	married	university.degree	no	yes	no	cellular
6	32	services	single	university.degree	no	no	no	cellular

	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate
1	may	fri	487	2	999	0	nonexistent	-1.8
2	may	fri	346	4	999	0	nonexistent	1.1
3	jun	wed	227	1	999	0	nonexistent	1.4
4	jun	fri	17	3	999	0	nonexistent	1.4
5	nov	mon	58	1	999	0	nonexistent	-0.1
6	sep	thu	128	3	999	2	failure	-1.1

	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
1	92.893	-46.2	1.313	5099.1	no
2	93.994	-36.4	4.855	5191.0	no
3	94.465	-41.8	4.962	5228.1	no
4	94.465	-41.8	4.959	5228.1	no
5	93.200	-42.0	4.191	5195.8	no
6	94.199	-37.5	0.884	4963.6	no

3.1.4 Dimensions of the dataset

```
dim(bank)
```

```
[1] 4119  21
```

3.1.5 Column names

```
colnames(bank)
```

```

[1] "age"          "job"          "marital"      "education"
[5] "default"      "housing"      "loan"         "contact"
[9] "month"        "day_of_week"  "duration"     "campaign"
[13] "pdays"       "previous"     "poutcome"     "emp.var.rate"
[17] "cons.price.idx" "cons.conf.idx" "euribor3m"    "nr.employed"
[21] "y"

```

Explanation (R)

- The file uses ; as a separator, which is common in datasets.
- We explicitly disable automatic factor conversion to retain control over data types.
- dim() confirms the dataset size (rows × columns).
- colnames() provides an overview of available variables.

3.2 Python: Load and Inspect the Data

```
import pandas as pd
```

3.2.1 Read the dataset

```
bank = pd.read_csv("data/raw/bank-additional.csv", sep=";")
```

3.2.2 Display first few rows

```
bank.head()
```

```

   age  job   marital  ... euribor3m nr.employed  y
0   30 blue-collar married ...    1.313     5099.1 no
1   39  services   single ...    4.855     5191.0 no
2   25  services married ...    4.962     5228.1 no
3   38  services married ...    4.959     5228.1 no
4   47   admin. married ...    4.191     5195.8 no

```

```
[5 rows x 21 columns]
```

3.2.3 Dimensions of the dataset

```
bank.shape
```

```
(4119, 21)
```

3.2.4 Column names

```
bank.columns.tolist()
```

```
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day']
```

Explanation (Python)

- As in R, we specify the separator explicitly.
 - `.head()` shows the first five rows.
 - `.shape` reports the dataset size.
 - `.columns.tolist()` gives a readable list of variable names.
-

4 A2. What Does One Row Represent?

Before doing *any* analysis, we must understand **what a single row corresponds to in the real world**.

4.1 Interpretation

A **single row represents one contact attempt** made during a direct marketing campaign by a Portuguese bank, where:

- a specific client was contacted,
- during a specific campaign period,
- under a particular economic context,
- resulting in a subscription outcome (`y = yes or no`).

Importantly:

- The same client may appear **multiple times**.
- Multiple rows do **not** necessarily represent independent clients.
- The dataset records *contact-level events*, not client summaries.

This interpretation has consequences for:

- aggregation (Since the same client can appear multiple times, you must be careful when summarising or aggregating the data.),
- leakage (Some variables describe outcomes of previous contacts or later information in the campaign. If these are used incorrectly, the model may accidentally use information that would not have been known at the time of prediction.),
- model design later in the module (Because rows are not independent clients, common modeling assumptions (such as independence of observations) may be violated.).

5 A3. Identifying Groups of Variables

We now group variables conceptually, based on what aspect of the process they describe.

5.1 Client-Related Variables

These describe **who the client is** (largely static attributes):

- age
- job
- marital
- education
- default
- housing
- loan

These variables typically exist **before** the marketing contact.

5.2 Campaign / Contact-Related Variables

These describe **how and when the client was contacted**:

- `contact`
- `month`
- `day_of_week`
- `duration`
- `campaign`: The number of times the client was contacted during the current marketing campaign.
- `pdays`: The number of days since the client was last contacted in a previous campaign. 999 means the client was not contacted previously.
- `previous`: The number of contacts made with the client before the current campaign.
- `poutcome`: The outcome of the previous marketing campaign for the client.

Some of these (notably `duration`) raise **serious validity and leakage concerns**, which will be revisited later.

5.3 Economic Context Variables

These describe the **macroeconomic environment** at the time of the campaign:

- `emp.var.rate`: The employment variation rate, a macroeconomic indicator.
- `cons.price.idx`: The consumer price index, a measure of inflation.
- `cons.conf.idx`: The consumer confidence index, measuring how optimistic consumers feel about the economy.
- `euribor3m`: The 3-month Euribor interest rate, a key European benchmark rate.
- `nr.employed`: The total number of people employed in the economy (in thousands).

These variables are shared across many rows and are **not client-specific**.

5.4 Outcome Variable

- `y` — whether the client subscribed to a term deposit (**yes** / **no**)

This is the **target variable** for supervised learning.

6 A4. Initial Structural Observations

From this first inspection, we can already make several important observations:

1. The dataset mixes **client-level**, **event-level**, and **context-level** information.
2. Several variables are categorical but encoded as strings.
3. Some numeric-looking values (e.g. `pdays = 999`) may actually be **codes**, not measurements.
4. The presence of `duration` suggests that **not all variables are valid at prediction time**.

These observations will guide all subsequent wrangling and modelling decisions.

7 Key Takeaway from Part A

Understanding what a row represents is more important than understanding how to fit a model.

Before moving to cleaning or EDA, we must be clear about:

- the data-generating process,
 - the unit of analysis,
 - and which variables are legitimate inputs.
-

8 Part B: Data Types & Representation

9 B1. Why Data Types Matter

Before cleaning or analysis, we must decide **how each variable should be interpreted**. Software may assign a data type automatically, but **meaning comes from context**, not syntax.

A common mistake is to assume that:

- numeric values are always quantitative, or
- categorical values are always nominal.

In this section, we examine **what each variable represents**, rather than how it is stored.

10 B2. Inspecting Software-Inferred Data Types

We start by checking how the software interprets each column.

10.0.1 R: Inspect Column Types

```
str(bank)
```

```
'data.frame':  4119 obs. of  21 variables:
 $ age          : int  30 39 25 38 47 32 32 41 31 35 ...
 $ job          : chr   "blue-collar" "services" "services" "services" ...
 $ marital      : chr   "married" "single" "married" "married" ...
 $ education    : chr   "basic.9y" "high.school" "high.school" "basic.9y" ...
 $ default      : chr   "no" "no" "no" "no" ...
 $ housing      : chr   "yes" "no" "yes" "unknown" ...
 $ loan         : chr   "no" "no" "no" "unknown" ...
 $ contact      : chr   "cellular" "telephone" "telephone" "telephone" ...
 $ month        : chr   "may" "may" "jun" "jun" ...
 $ day_of_week  : chr   "fri" "fri" "wed" "fri" ...
 $ duration     : int  487 346 227 17 58 128 290 44 68 170 ...
 $ campaign     : int    2  4  1  3  1  3  4  2  1  1 ...
 $ pdays       : int  999 999 999 999 999 999 999 999 999 999 ...
```

```

$ previous      : int  0 0 0 0 0 2 0 0 1 0 ...
$ poutcome      : chr  "nonexistent" "nonexistent" "nonexistent" "nonexistent" ...
$ emp.var.rate  : num  -1.8 1.1 1.4 1.4 -0.1 -1.1 -1.1 -0.1 -0.1 1.1 ...
$ cons.price.idx: num   92.9 94 94.5 94.5 93.2 ...
$ cons.conf.idx : num  -46.2 -36.4 -41.8 -41.8 -42 -37.5 -37.5 -42 -42 -36.4 ...
$ euribor3m     : num   1.31 4.86 4.96 4.96 4.19 ...
$ nr.employed   : num   5099 5191 5228 5228 5196 ...
$ y             : chr  "no" "no" "no" "no" ...

```

Explanation (R)

- `str()` shows how R currently stores each column.
- At this stage, many variables are character strings, even though they encode categories.
- Numeric-looking columns may still require reinterpretation.

10.0.2 Python: Inspect Column Types

```
bank.dtypes
```

```

age                int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration           int64
campaign           int64
pdays             int64
previous           int64
poutcome           object
emp.var.rate       float64
cons.price.idx     float64
cons.conf.idx      float64
euribor3m          float64

```

```
nr.employed    float64
y              object
dtype: object
```

Explanation (Python)

- Pandas infers data types based on values present.
 - This inference is technical, not semantic.
 - Our task is to judge whether the inferred types make *conceptual sense*.
-

11 B3. Conceptual Classification of Variables

We now classify variables based on **what they mean**, not how they are stored.

11.1 Client-Related Variables

Variable	Conceptual Type	Notes
age	Numeric (continuous)	Measured in years
job	Categorical (nominal)	No inherent ordering
marital	Categorical (nominal)	Categories are labels
education	Categorical (ordinal-ish)	Ordering exists, but spacing is unclear
default	Binary categorical	Yes / No / Unknown
housing	Binary categorical	Yes / No / Unknown
loan	Binary categorical	Yes / No / Unknown

Even when categories look ordered (e.g. education), treating them as numeric is risky.

11.2 Campaign / Contact Variables

Variable	Conceptual Type	Notes
<code>contact</code>	Categorical	Method of contact
<code>month</code>	Categorical (cyclic)	Month names, not numeric
<code>day_of_week</code>	Categorical (cyclic)	Ordering exists but is circular
<code>duration</code>	Numeric (but problematic)	Known <i>after</i> the call
<code>campaign</code>	Numeric (count)	Number of contacts
<code>pdays</code>	Numeric code	999 means “not previously contacted”
<code>previous</code>	Numeric (count)	Past contacts
<code>poutcome</code>	Categorical	Outcome of previous campaign

11.3 Economic Context Variables

Variable	Conceptual Type	Notes
<code>emp.var.rate</code>	Numeric (continuous)	Macro-level
<code>cons.price.idx</code>	Numeric (continuous)	Macro-level
<code>cons.conf.idx</code>	Numeric (continuous)	Macro-level
<code>euribor3m</code>	Numeric (continuous)	Interest rate
<code>nr.employed</code>	Numeric (continuous)	Labour market

These variables are **shared across many rows** and vary slowly over time.

11.4 Outcome Variable

Variable	Conceptual Type	Notes
<code>y</code>	Binary categorical	Target variable

12 B4. Numeric Does Not Always Mean Quantitative

Some variables are stored as numbers but **should not be treated as numeric measurements**.

12.1 Example 1: pdays

The variable `pdays` records the number of days since the client was last contacted. However, the value 999 is a **code** meaning “*not previously contacted*”.

Treating 999 as a real number would:

- distort summaries,
 - mislead models,
 - imply false ordering.
-

12.1.1 R: Inspect pdays

```
table(bank$pdays)
```

0	1	2	3	4	5	6	7	9	10	11	12	13	14	15	16
2	3	4	52	14	4	42	10	3	8	1	5	2	1	2	2
17	18	19	21	999											
1	2	1	1	3959											

12.1.2 Python: Inspect pdays

```
bank["pdays"].value_counts().head()
```

```
pdays
999    3959
3         52
6         42
4         14
7         10
Name: count, dtype: int64
```

Interpretation

- The dominance of 999 indicates a categorical state, not a large number of days.
 - This variable must be **re-encoded or handled carefully** later.
-

12.2 Example 2: duration

duration measures the length of the phone call in seconds.

While numeric, it is **not valid for prediction**, because:

- it is only known *after* the call ends,
- it directly encodes success likelihood.

Including it would result in **data leakage**.

This does not mean it should be deleted immediately — it can still be useful for:

- post-hoc analysis,
 - descriptive understanding.
-

13 B5. Ordinal vs Nominal: A Subtle Distinction

Some variables appear ordered but lack meaningful numeric spacing.

13.1 Example: education

Values such as:

- `basic.9y`
- `high.school`
- `university.degree`

suggest ordering, but:

- the distance between levels is not defined,
- numeric encoding would impose artificial structure.

At this stage, it is safest to treat `education` as **categorical**, not numeric.

14 B6. Representation Is a Modelling Decision

From this analysis, an important principle emerges:

Choosing how to represent a variable is already a modelling decision.

Even before fitting any model, we have decided:

- what counts as numeric,
- what counts as categorical,
- which values are codes,
- which variables may be invalid at prediction time.

These decisions will:

- shape EDA results,
 - influence model behaviour,
 - affect interpretability and fairness.
-

15 Key Takeaway from Part B

Data types inferred by software are **not** the same as data meanings.
Thoughtful representation is a prerequisite for valid machine learning.

16 Part C: Data Quality & Validity Checks

17 C1. Why Data Quality Comes Before Modelling

Before any modelling or even detailed EDA, we must assess **whether the data is valid for the questions we intend to ask**.

Data quality is **not** just about missing values. It includes:

- coded values that look numeric,
- categories such as "unknown",
- variables that violate the prediction-time assumption.

In this section, we systematically identify such issues and reason about how to handle them.

18 C2. Missing Values vs “Unknown” Values

The dataset does **not** contain standard missing values (NA / NaN) in most columns. Instead, missingness is often encoded explicitly as "unknown" or implicitly via sentinel values.

18.0.1 R: Check for Missing Values

```
colSums(is.na(bank))
```

age	job	marital	education	default
0	0	0	0	0
housing	loan	contact	month	day_of_week
0	0	0	0	0
duration	campaign	pdays	previous	poutcome
0	0	0	0	0
emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	0	0	0	0
y				
0				

Observation

- There are no classical NA values.
 - This does *not* mean the data is complete.
-

18.0.2 Python: Check for Missing Values

Count missing (NaN) values per column

```
bank.isna().sum()
```

age	0
job	0
marital	0
education	0
default	0
housing	0
loan	0
contact	0
month	0
day_of_week	0
duration	0

```

campaign      0
pdays        0
previous      0
poutcome      0
emp.var.rate  0
cons.price.idx 0
cons.conf.idx 0
euribor3m     0
nr.employed   0
y             0
dtype: int64

```

19 C3. Identifying “Unknown” Categories

Several categorical variables use "unknown" as a valid category.

19.0.1 R: Frequency of “unknown” Values

```

unknown_counts <- sapply(bank, function(x) {
  if (is.character(x)) sum(x == "unknown") else 0
})
unknown_counts

```

```

      age      job      marital  education      default
      0       39        11        167        803
housing  loan      contact      month  day_of_week
  105     105         0         0         0
duration campaign  pdays    previous  poutcome
      0         0         0         0         0
emp.var.rate cons.price.idx cons.conf.idx euribor3m nr.employed
      0         0         0         0         0
y
0

```

19.0.2 Python: Frequency of “unknown” Values

```
unknown_counts = {  
col: (bank[col] == "unknown").sum()  
for col in bank.columns  
if bank[col].dtype == "object"  
}  
  
unknown_counts
```

```
{'job': np.int64(39), 'marital': np.int64(11), 'education': np.int64(167), 'default': np.int64(167), 'housing': np.int64(167), 'loan': np.int64(167), 'other': np.int64(167)}
```

Interpretation

- Variables such as `job`, `education`, `default`, `housing`, and `loan` contain "unknown".
 - These values are **not random missingness** — they reflect operational or reporting limitations.
-

20 C4. Sentinel Values and Coded Entries

Some numeric-looking variables encode **special states**.

20.1 Example 1: `pdays`

As discussed earlier:

- `pdays = 999` means *client not previously contacted*.
-

20.1.1 R: Examine Distribution of `pdays`

```
table(bank$pdays)
```

0	1	2	3	4	5	6	7	9	10	11	12	13	14	15	16
2	3	4	52	14	4	42	10	3	8	1	5	2	1	2	2
17	18	19	21	999											
1	2	1	1	3959											

20.1.2 Python: Examine Distribution of pdays

```
bank["pdays"].value_counts().sort_index().head(10)
```

```
pdays
0      2
1      3
2      4
3     52
4     14
5      4
6     42
7     10
9      3
10     8
Name: count, dtype: int64
```

Interpretation

- The dominance of 999 indicates a **categorical condition**, not a numeric measurement.
 - Treating `pdays` as continuous without recoding would distort both EDA and modelling.
-

20.2 Example 2: campaign and previous

These variables are counts, but:

- high values may indicate repeated targeting,
- extreme values may represent special cases rather than typical behaviour.

They are **valid**, but require interpretation.

21 C5. Variables That Threaten Validity

Some variables are technically correct but **conceptually problematic**.

21.1 Critical Example: duration

`duration` measures call length (in seconds).

While accurate, it violates a core modelling assumption:

At prediction time, call duration is not known.

Including `duration` would:

- artificially inflate performance,
- create a misleading model,
- invalidate conclusions.

This is an example of **data leakage**, not missing data.

22 C6. Classifying Data Quality Issues

We now classify several observed issues.

22.1 Issue 1: "unknown" in education

- **Type:** Representation / measurement issue
- **Cause:** Information not recorded or disclosed
- **Possible actions:**
 - keep as a separate category,
 - combine with similar levels,
 - treat as missing later.

No single correct choice — justification matters.

22.2 Issue 2: pdays = 999

- **Type:** Representation issue
 - **Cause:** Sentinel coding
 - **Possible actions:**
 - convert to categorical (“previously contacted: yes/no”),
 - replace with missing and add indicator variable.
-

22.3 Issue 3: duration

- **Type:** Validity issue (prediction-time leakage)
 - **Cause:** Variable observed after outcome
 - **Action:**
 - exclude from predictive models,
 - retain for descriptive analysis.
-

23 C7. Should Everything Be “Cleaned”?

A key principle:

Not all unusual values are errors.

- "unknown" may carry information.
- 999 encodes a meaningful state.
- Rare categories may represent real but uncommon situations.

Over-aggressive cleaning can:

- remove signal,
 - introduce bias,
 - oversimplify reality.
-

24 C8. Data Quality Is Question-Dependent

The same variable may be:

- valid for EDA,
- invalid for prediction,
- acceptable for clustering,
- problematic for causal interpretation.

For example:

- **duration** is useful for understanding call outcomes,
 - but invalid for pre-call prediction.
-

25 Key Takeaway from Part C

Data quality is not about making data “look nice”. It is about ensuring that conclusions are valid for the question being asked.

Every data quality decision should be:

- explicit,
 - justified,
 - revisited when the task changes.
-

26 Part D: Exploratory Data Analysis (EDA)

27 D1. Purpose of Exploratory Data Analysis

Exploratory Data Analysis (EDA) is used to:

- understand distributions and patterns,
- identify unusual behaviour,
- generate hypotheses for later modelling.

EDA **does not prove causation** and **does not optimise models**. Its role is to **inform better questions**.

28 D2. Exploring the Outcome Variable (y)

We begin by examining the target variable to understand **class balance**.

28.0.1 R: Distribution of y

Count outcomes

```
table(bank$y)
```

```
no  yes  
3668 451
```

Proportion of outcomes

```
prop.table(table(bank$y))
```

```
no      yes  
0.8905074 0.1094926
```

28.0.2 Python: Distribution of y

```
bank["y"].value_counts()
```

```
y  
no      3668  
yes      451  
Name: count, dtype: int64
```

```
bank["y"].value_counts(normalize=True)
```

```
y  
no      0.890507  
yes      0.109493  
Name: proportion, dtype: float64
```

Interpretation

- The outcome variable is **imbalanced**.
- Most clients do **not** subscribe to the term deposit.
- This imbalance has implications for:
 - model evaluation,
 - metric choice,
 - interpretation of accuracy.

At this stage, we simply **note** the imbalance.

29 D3. Exploratory Analysis of Categorical Variables

We now examine how categorical variables relate to the outcome.

29.1 Example 1: Job Type vs Subscription

29.1.1 R: Job vs Outcome

```
job_y <- table(bank$job, bank$y)

# Convert to proportions by job
prop.table(job_y, margin = 1)
```

	no	yes
admin.	0.86857708	0.13142292
blue-collar	0.93099548	0.06900452
entrepreneur	0.94594595	0.05405405
housemaid	0.90000000	0.10000000
management	0.90740741	0.09259259
retired	0.77108434	0.22891566

self-employed	0.91823899	0.08176101
services	0.91094148	0.08905852
student	0.76829268	0.23170732
technician	0.88422576	0.11577424
unemployed	0.82882883	0.17117117
unknown	0.89743590	0.10256410

29.1.2 Python: Job vs Outcome

```
pd.crosstab(bank["job"], bank["y"], normalize="index")
```

y	no	yes
job		
admin.	0.868577	0.131423
blue-collar	0.930995	0.069005
entrepreneur	0.945946	0.054054
housemaid	0.900000	0.100000
management	0.907407	0.092593
retired	0.771084	0.228916
self-employed	0.918239	0.081761
services	0.910941	0.089059
student	0.768293	0.231707
technician	0.884226	0.115774
unemployed	0.828829	0.171171
unknown	0.897436	0.102564

Interpretation

- Some job categories appear to have higher subscription rates.
 - However:
 - some categories are rare,
 - proportions may be unstable.
 - This is **association**, not causation.
-

29.2 Example 2: Contact Method vs Outcome

29.3 R: Contact vs Outcome

```
prop.table(table(bank$contact, bank$y), margin = 1)
```

	no	yes
cellular	0.85859729	0.14140271
telephone	0.94819359	0.05180641

29.4 Python: Contact vs Outcome

```
pd.crosstab(bank["contact"], bank["y"], normalize="index")
```

y	no	yes
contact		
cellular	0.858597	0.141403
telephone	0.948194	0.051806

Interpretation

- Different contact methods show different success rates.
- This may reflect:
 - operational decisions,
 - client availability,
 - campaign strategy.

EDA alone cannot disentangle these effects.

30 D4. Exploratory Analysis of Numeric Variables

Numeric variables allow us to explore distributions and conditional patterns.

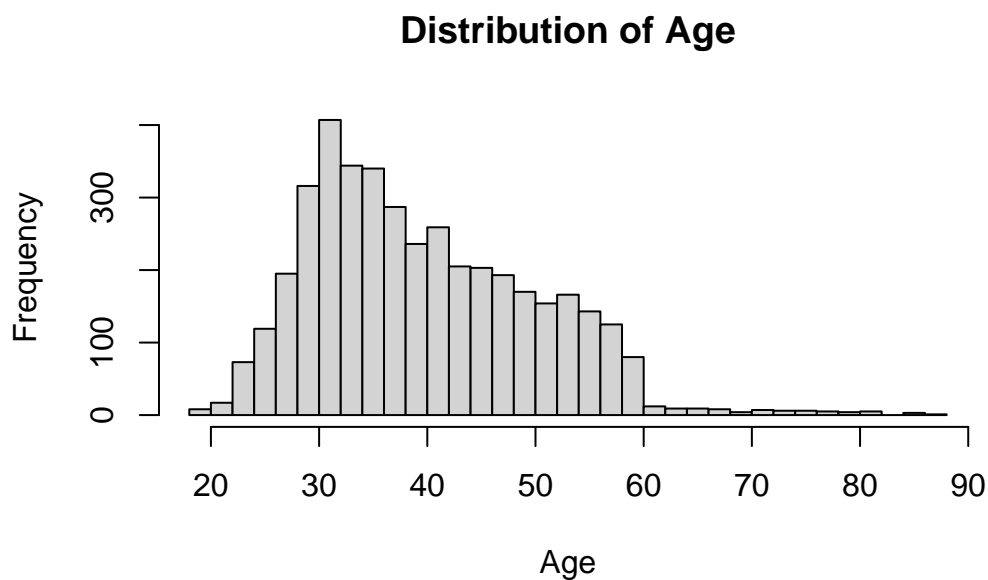
30.1 Example 1: Age Distribution

30.1.1 R: Age Distribution

```
summary(bank$age)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
18.00	32.00	38.00	40.11	47.00	88.00

```
hist(bank$age, breaks = 30,  
main = "Distribution of Age",  
xlab = "Age")
```

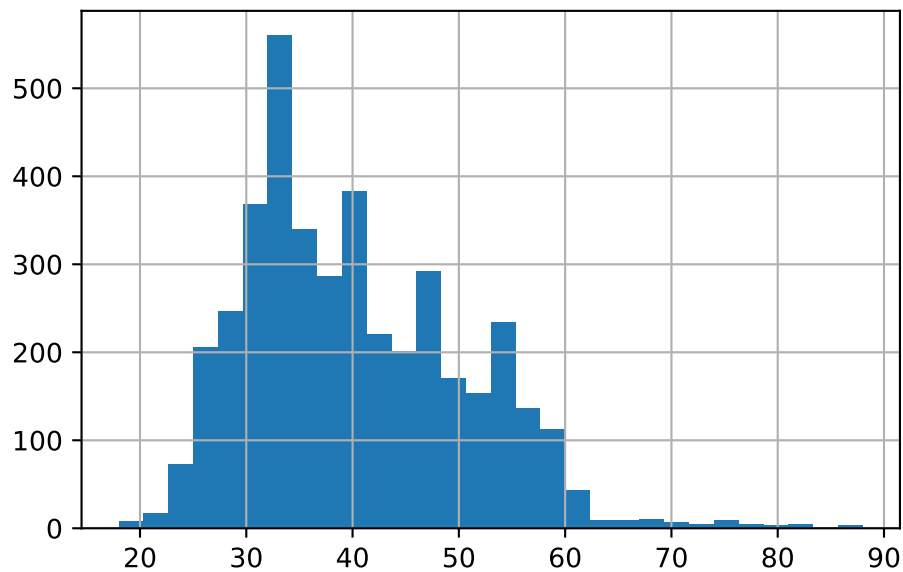


30.1.2 Python: Age Distribution

```
bank["age"].describe()
```

```
count    4119.000000
mean      40.113620
std       10.313362
min       18.000000
25%       32.000000
50%       38.000000
75%       47.000000
max       88.000000
Name: age, dtype: float64
```

```
bank["age"].hist(bins=30)
```



Interpretation

- Age is right-skewed.

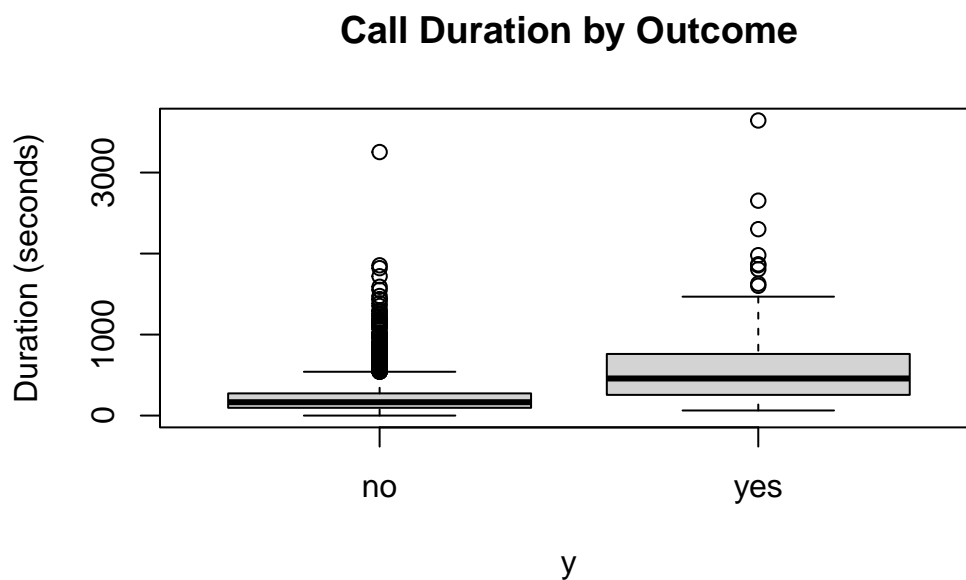
- Need to observe the overlap between subscribers and non-subscribers.
 - Can we identify/conclude threshold from the distribution alone?
-

30.2 Example 2: Duration and Outcome (with Caution)

Even though duration is **not valid for prediction**, it is informative for EDA.

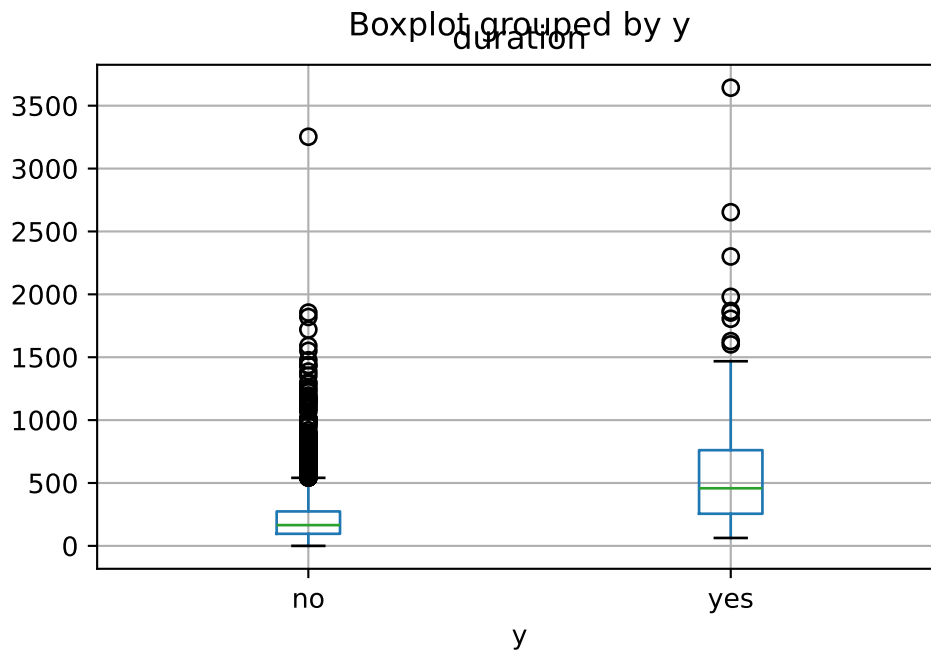
30.2.1 R: Duration by Outcome

```
boxplot(duration ~ y,  
data = bank,  
main = "Call Duration by Outcome",  
ylab = "Duration (seconds)")
```



30.2.2 Python: Duration by Outcome

```
bank.boxplot(column="duration", by="y")
```



Interpretation

- Calls resulting in subscription tend to be longer.
 - This relationship is **expected**, but also **problematic**.
 - It reinforces why **duration** must be excluded from predictive models.
-

31 D5. Multivariate EDA: Combining Variables

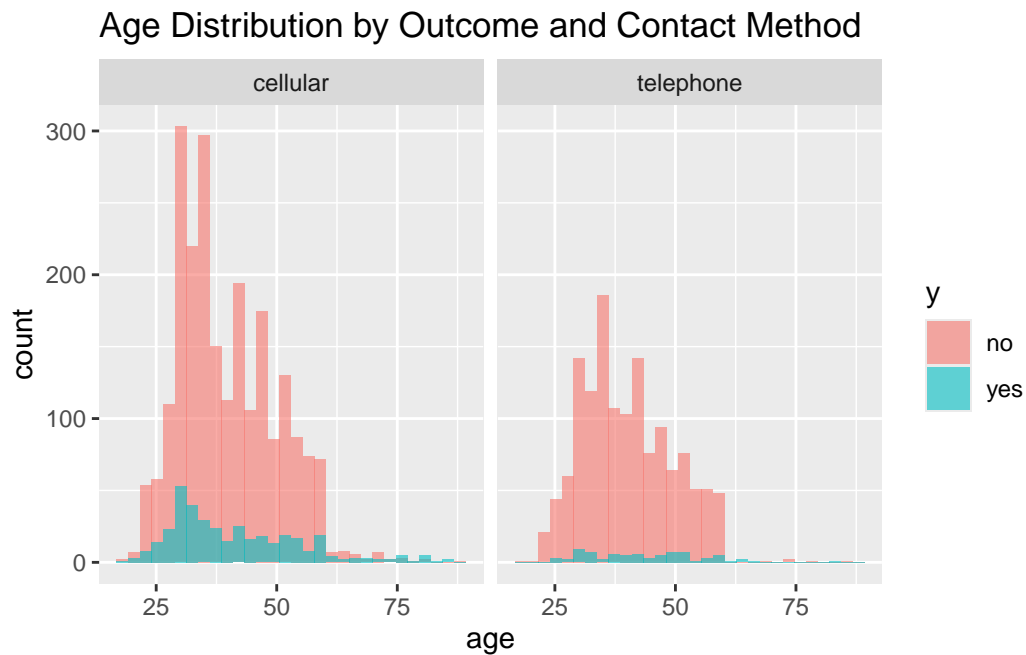
EDA becomes more informative when variables are combined.

31.1 Example: Age vs Outcome by Contact Method

31.1.1 R

```
library(ggplot2)

ggplot(bank, aes(x = age, fill = y)) +
  geom_histogram(bins = 30, position = "identity", alpha = 0.6) +
  facet_wrap(~ contact) +
  labs(title = "Age Distribution by Outcome and Contact Method")
```

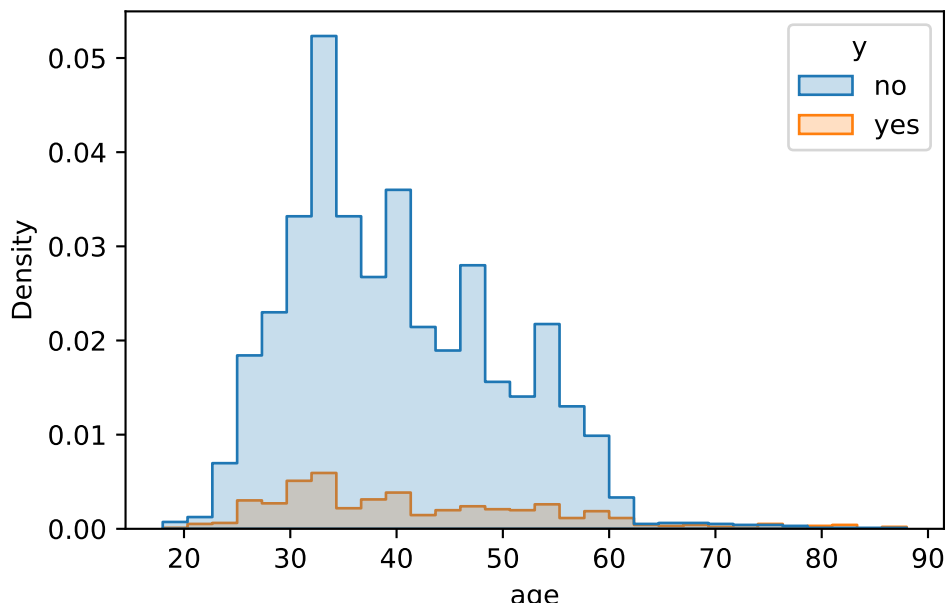


31.1.2 Python

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(data=bank, x="age", hue="y",
             bins=30, element="step", stat="density")

plt.show()
```



Interpretation

- Patterns may differ across contact methods.
- Overlaps remain substantial.
- Multivariate EDA suggests complexity rather than simple rules.

32 D6. What EDA Can — and Cannot — Tell Us

From this EDA, we can say:

- Some variables are associated with subscription outcomes
- Certain categories appear more promising than others

- Data imbalance is present

But we **cannot** conclude:

- which variables *cause* subscription
- how a model should weight features
- what performance is achievable

These questions require **careful modelling**, informed by EDA.
