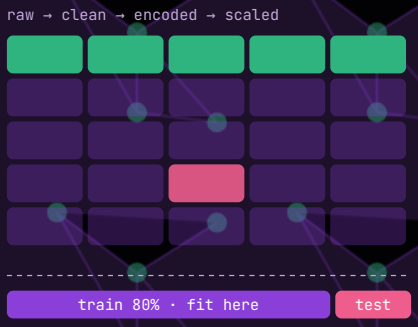
STUDY PAGE · AREA 03 OF 14

Data Prep & Feature Engineering



03

Turn raw, messy tables into honest model inputs: explore, fix gaps and outliers, encode and scale, engineer features — and split so the test score means something.

14 h of study · 4 modules · 9 diagrams · 3 interactives

Free tools · pandas + scikit-learn, or Google Colab · 20-question test · 70% to pass

Lab: a leak-free pipeline

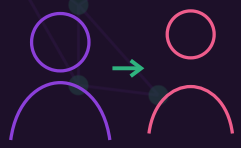
1 · EDA → 2 · Missing & outliers → 3 · Encode & scale → 4 · Splits & leakage → ab · leak-free pipelin → Area test · 70%

shape · gaps · target | impute · IQR · clip | one-hot · z-score | Pipeline · CV | three bugs | 20 questions

BEFORE YOU START · A

Who this area is for

Most of an ML engineer's time goes here, and most silent failures start here. This area makes the boring part rigorous.



CS / data / stats students

Students preparing for ML internships and new-grad roles who want the practical path, not only theory.

Software engineers

Backend and full-stack developers moving into ML engineering, who already write good code and want the ML on top.

Data analysts

Analysts who know SQL and spreadsheets and want to build and ship models, not just report on data.

QA & test engineers

Test automation engineers heading into ML: evaluation, data checks and model testing use the same mindset.

Data engineers

Pipeline builders who want to own the models their data feeds, and the MLOps around them.

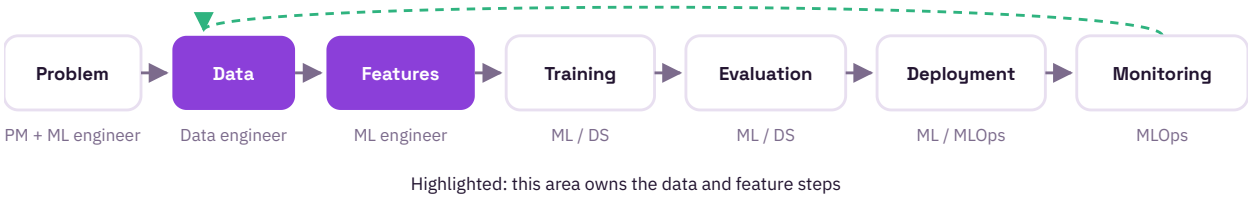
Career switchers

Engineers from other fields (EE, mechanical, finance) with some Python, aiming for an ML engineer job.

Prerequisites

You need	Level	Where to get it
pandas DataFrames, groupby, joins	Comfortable	Area 01
Mean, median, std, correlation	Basic	Area 02, Module 4
scikit-learn	None	Introduced here

Where this area sits in the ML lifecycle — and the roles that use it



Highlighted: this area owns the data and feature steps

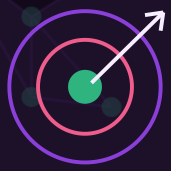
Garbage in, garbage out. Every later stage inherits the choices made here.

Role	How you use this area on the job	Typical first title
ML engineer	Build preprocessing pipelines that run identically in training and serving	ML Engineer I / Machine Learning Engineer
Data scientist	Find signal and leaks during EDA; engineer features	Data Scientist / Applied Scientist
AI / LLM engineer	Clean and split text data and eval sets without contamination	AI Engineer / LLM Engineer
MLOps / platform	Version preprocessing with the model; monitor input quality	MLOps Engineer / ML Platform Engineer
Data engineer	Deliver complete, typed, deduplicated tables with known timestamps	Data Engineer

BEFORE YOU START · B

What you will be able to do

Every outcome is practised in the lab and checked by the area test.

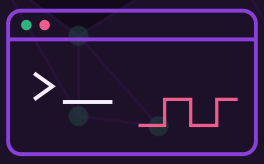


- Run a five-pass EDA and write a data dictionary.
- Spot ids, post-outcome columns and other leaks.
- Choose an imputation strategy per column and add missing indicators.
- Flag outliers with the IQR rule and decide to fix, clip or log-transform.
- Encode categories with one-hot or ordinal encoding and handle unseen values.
- Standardize or min-max scale and know which models need it.
- Engineer ratio, flag and log features from domain sense.
- Build a stratified, leak-free Pipeline with ColumnTransformer and cross-validate it.

BEFORE YOU START · C

Tools & software

pandas for exploring, scikit-learn for the reusable pipeline.



Tool	What it does	Used in this area for	Cost
Python 3.10 – 3.13	The language of ML: every library on this page is a Python package	Every example and the lab	Free
venv + **pip**	An isolated environment per project and the package installer	Installing the area's packages	Free
pandas	Tables (DataFrames): load, clean, group and join data	EDA, cleaning, new features	Free
NumPy	Fast n-dimensional arrays and maths	log transforms, arrays	Free
scikit-learn	Classic ML: preprocessing, models, metrics, pipelines	Imputers, encoders, scalers, Pipeline, splits	Free
Matplotlib	Charts: histograms, scatter plots, learning curves	Histograms and box plots	Free
JupyterLab	Notebooks: run code cell by cell and see tables and plots inline	Interactive EDA	Free
Google Colab	Hosted notebooks in the browser, with a free GPU on request	No-install option	Free

Platforms you will meet at work

Platform	Vendor	What teams use it for
SageMaker	AWS	Managed notebooks, training jobs, model registry and endpoints
Vertex AI	Google Cloud	Training, pipelines, feature store, model serving and Gemini APIs
Azure Machine Learning	Microsoft	Workspaces, MLflow tracking, managed endpoints
Databricks	Databricks	Spark + notebooks + MLflow on a lakehouse
Weights & Biases	W&B	Experiment tracking and model evaluation dashboards

Same code, bigger machines

The same Pipeline object you build here is what gets saved and served in Area 12.

BEFORE YOU START · D

Install step by step

Pick your computer. About 15 minutes. If anything fails, Google Colab works with nothing installed.



Windows

Recommended: WSL 2 (Ubuntu inside Windows) — the same commands as Linux, cloud servers and most ML teams.

1

Turn on WSL and install Ubuntu

Open **PowerShell as Administrator**, run the command, then restart.

```
wsl --install
```

After the restart an Ubuntu window opens: choose a user name and password.

2

Install Python's venv support

```
sudo apt update
sudo apt install -y python3 python3-venv python3-pip git make
```

3

Create an environment and install the packages

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install pandas numpy scikit-learn matplotlib jupyterlab
```

Run `source ~/ml/bin/activate` in every new terminal.

4

Editor

Install VS Code, the **WSL**, **Python** and **Jupyter** extensions, then type `code .` in Ubuntu.

5

Check it works

```
python -c "import sklearn, pandas; print(sklearn.__version__, pandas.__
```

i

No WSL on your laptop?

Install Python 3.12 from python.org (tick *Add python.exe to PATH*), then in PowerShell: `py -m venv %USERPROFILE%\ml` , `%USERPROFILE%\ml\Scripts\activate` and the same `pip install` line. Every lab runs the same.

macOS

1

Install Apple's command-line tools

```
xcode-select --install
```

2

Install Homebrew and Python

Follow the one-line command on brew.sh, then:

```
brew install python@3.12 git make
```

3

Create an environment and install the packages

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install pandas numpy scikit-learn matplotlib jupyterlab
```

4

Editor

Install VS Code with the **Python** and **Jupyter** extensions. Apple-silicon Macs run every lab natively.

5

Check it works

```
python -c "import sklearn, pandas; print(sklearn.__version__, pandas.__
```

Linux

1

Ubuntu / Debian

```
sudo apt update
sudo apt install -y python3 python3-venv python3-pip git make
```

2

Fedora

```
sudo dnf install -y python3 python3-pip git make
```

3 **Arch**

```
sudo pacman -S python python-pip git make
```

4 **Create an environment and install the packages**

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install pandas numpy scikit-learn matplotlib jupyterlab
```

5 **Check it works**

```
python -c "import sklearn, pandas; print(sklearn.__version__, pandas.__
```

Browser only

1 **Open Google Colab**

Go to colab.research.google.com and sign in with a Google account.

2 **New notebook**

File → New notebook. NumPy, pandas, scikit-learn, Matplotlib and PyTorch are already installed.

3 **Upload the lab**

Click the folder icon, upload the lab zip, then run `!unzip lab.zip && cd lab && python check.py` in a cell (prefix shell commands with `!`).

4 **Check the versions**

```
python -c "import sklearn, pandas; print(sklearn.__version__, pandas.__
```

BEFORE YOU START · E

First run in 5 minutes

Load the lab's dataset, count its gaps and make a stratified split.



1

Get the data

Download the lab kit and run `python make_data.py` — it writes `customers.csv`.

2

Create `first.py`

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("customers.csv")
print(df.shape, "| churn rate", round(df["churned"].mean(), 3))
print(df.isna().sum()[lambda s: s > 0].to_dict())
train, test = train_test_split(df, test_size=0.2, stratify=df["churned"]
print("train", len(train), "test", len(test))
```

3

Run it

```
python first.py
```

Output (captured from a real run):

```
(2000, 9) | churn rate 0.299
{'monthly_fee': 169, 'age': 185, 'region': 95}
train 1600 test 400
```

4

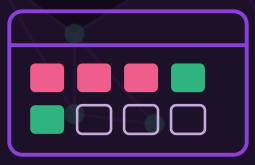
What to notice

Three columns have gaps, and churn is about 30%. Stratifying keeps that 30% in both train and test — the starting point of every honest evaluation.

BEFORE YOU START · F

Study plan

14 hours at about 7 hours a week.



Session	Do this	Time
Day 1	Install · first run · Module 1 (EDA)	3 h
Day 2	Module 2 (missing values, outliers) + outlier widget	3.5 h
Day 3	Module 3 (encoding, scaling, features) + encoder widget	3.5 h
Day 4	Module 4 (splits, leakage, Pipeline) + split widget	2.5 h
Day 5	Lab: leak-free pipeline · area test	1.5 h

MODULE 1 OF 4

EDA & data quality

Know your data before you model it.

- The five-pass EDA
- Types, gaps and cardinality
- Target balance
- Spotting leaks early



3 hours | Area 03 · Data Prep

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

Look before you cook

In plain words

Before training anything, look at the table: how many rows, what each column holds, what is missing.

Think of it like

A chef checks the fridge before planning the menu.

Worked example

question	answer for customers.csv
rows × columns	2,000 × 9
missing	fee, age, region
target	churned, ~30% yes

The word for it

Exploratory data analysis (EDA).

! **Why it matters** Most model bugs are data bugs you could have seen in the first ten minutes.

2 STEP 2 — Would I know this at prediction time?

≈ **In plain words** A column is only a fair input if its value exists at the moment you make the prediction.

✳ **Think of it like** Predicting a football result using the final score is not a prediction.

✎ **Worked example**

column	known before churn?
tenure_months	yes
support_calls	yes
refund_issued	no — set after leaving

Aa **The word for it** **Target leakage**.

! **Why it matters** Leaky columns give brilliant test scores and useless production models.

PAUSE & TRY **A loan model has a column “days_late_on_this_loan”. Use it to predict default at approval time?** Answer

No — it only exists after the loan is running. Leakage.

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

EDA
Exploratory data analysis: looking at the data before modelling it — shape, types, gaps, spread, target.

Data quality
Whether values are complete, valid, consistent and unique. Bad quality caps model quality.

Dtype
The stored type of a column. Numbers stored as text ("45.0") silently break maths.

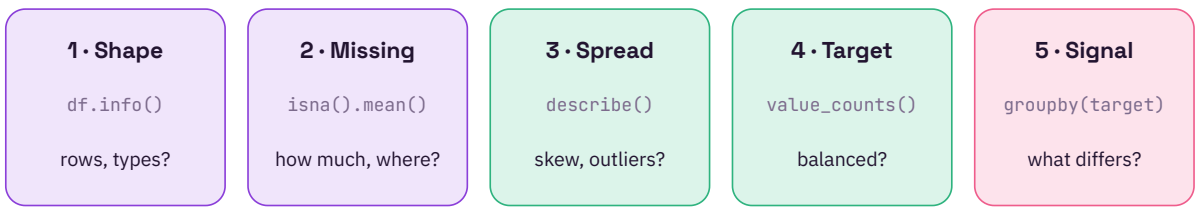
Cardinality
How many distinct values a column has. An id has one per row and is useless as a feature.

Class balance
How often each target value occurs. 30% churn is mildly imbalanced; 1% fraud is severe.

Skew
A long tail on one side. Tenure, income and prices are usually right-skewed.

Duplicates
Repeated rows. They inflate counts and can leak between train and test.

Data dictionary
A table saying what each column means, its unit and when it becomes known.



Twenty minutes of EDA before modelling saves days of debugging after it.

Five passes, in order. Each answers one question and points to a fix in Modules 2–4.

```python
import pandas as pd

df = pd.read_csv("customers.csv")
print(df.shape)
print(df.dtypes.to_string())
print("\nmissing share:")
print(df.isna().mean().round(3)[lambda s: s > 0].to_string())
```

```
# output
(2000, 9)
customer_id        int64
tenure_months    float64
monthly_fee      float64
support_calls      int64
age              float64
plan                 str
region               str
refund_issued      int64
churned            int64

missing share:
monthly_fee    0.084
age            0.092
region         0.048
```

```python
import pandas as pd

df = pd.read_csv("customers.csv")
print(df[["tenure_months", "monthly_fee", "support_calls", "age"]].describe().ro
print("\nchurn rate:", round(df["churned"].mean(), 3))
print("distinct customer_id:", df["customer_id"].nunique(), "of", len(df))
```

```
# output
       tenure_months  monthly_fee  support_calls     age
count         2000.0       1831.0         2000.0  1815.0
mean            17.4         44.9            1.6    41.0
std             12.8         15.1            1.3    11.7
min              0.4          9.0            0.0    18.0
25%              8.1         35.0            1.0    32.7
50%             14.4         45.2            1.0    40.8
75%             23.1         55.3            2.0    48.9
max             89.4        105.9            7.0    77.1

churn rate: 0.299
distinct customer_id: 2000 of 2000
```

Look for signal: what differs between churners and stayers?

```python
import pandas as pd

df = pd.read_csv("customers.csv")
print(df.groupby("churned")[["tenure_months", "support_calls", "monthly_fee"]].m
print()
print(df.groupby("plan")["churned"].mean().round(3).to_string())
print()
print(df.groupby("refund_issued")["churned"].mean().round(3).to_string())
```

```
# output
         tenure_months  support_calls  monthly_fee
churned
0                19.15           1.40        44.26
1                13.24           2.22        46.58

plan
basic    0.358
plus     0.237
pro      0.259

refund_issued
0    0.095
1    1.000
```

A feature that is “too good” is usually a leak

Every customer with `refund_issued = 1` churned — because refunds are issued *after* cancelling. At prediction time the column is always 0. Ask of every column: **would I know this value at the moment I make the prediction?**

Check	pandas	Red flag
shape & types	`df.info()`	numbers stored as object
missing	`df.isna().mean()`	> 30% missing, or missing only for one class
distribution	`df.describe()`	min/max impossible (age −3, fee 99,999)
duplicates	`df.duplicated().sum()`	repeated rows or ids
cardinality	`df.nunique()`	one value per row (ids) or one value total
target	`y.value_counts(normalize=True)`	rare positive class

Common mistakes

Skipping EDA and going straight to `fit` · keeping ids as features · trusting dtypes without checking · not asking when each column becomes known · looking at the test set during EDA (it should stay unseen).

INTERVIEW CORNER

Q. What do you do first with a new dataset? **A.** Shape and types, missing values, distributions, target balance, then relationships — and a data dictionary with when each field is known.

Q. A feature has 0.99 correlation with the target. Good news? **A.** Suspicious: check whether it is recorded after the outcome (leakage) or is a copy of the target.

Key takeaways

1. EDA is five passes: shape, missing, spread, target, signal.
2. Ids and post-outcome columns are not features.
3. Keep the test set out of EDA.

MODULE 2 OF 4

Missing values & outliers

Fill gaps and tame extremes without hiding information.

- Why values go missing
- Imputation strategies
- Missing indicators
- IQR, clipping and log transforms



⏱ 3.5 hours | Area 03 · Data Prep

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

1 Fill the gaps sensibly

≈ In plain words

Missing values must be filled or the model can't run. Use a typical value: the median for numbers, the most common value for categories.

✳ Think of it like

A missing jigsaw piece: you cut a plain piece of roughly the right colour so the picture holds together.

Worked example

```
import pandas as pd
fee = pd.Series([40, 45, None, 50, 400])
print(fee.fillna(fee.median()).tolist())
```

```
# output
[40.0, 45.0, 47.5, 50.0, 400.0]
```

The word for it — **Imputation**; **median imputation**.

Why it matters — The median ignores the 400 outlier that would drag a mean fill upward.

STEP 2

2 A gap can be a clue

In plain words — Sometimes the fact that a value is missing tells you something. Keep a 0/1 flag for it.

Think of it like — A blank “current employer” field on a form says something on its own.

Worked example

fee_missing	churn rate
0	29.6%
1	33.1% — a small hint

The word for it — **Missing indicator**.

Why it matters — When gaps are not random (MNAR), the indicator lets the model use them.

STEP 3

3 Outliers: mistake or treasure?

≈ **In plain words**	A value far from the others is either an error to fix or a real extreme to keep but tame.
☀ **Think of it like**	A 240-year-old customer is a typo; a customer spending 100× the average is your best client.
✎ **Worked example**	**1.** Q1 = 15.5, Q3 = 21.5 → IQR = 6 **2.** upper fence = 21.5 + 1.5 × 6 = 30.5 **3.** 61 > 30.5 → flag it, then decide
Aa **The word for it**	**Outlier**, **IQR rule**, **clipping**.
! **Why it matters**	Extreme values distort means, scales and linear models.

≡

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Missing value
An empty cell: NaN in pandas/NumPy, None in Python, NULL in SQL.

Imputation
Filling missing values with an estimate such as the median or most frequent value.

Median imputation
Fill with the column median; robust to outliers and skew.

Missing indicator
A 0/1 column saying a value was missing, so the model can learn from the gap itself.

Outlier
A value far from the rest. Could be an error (age 240) or real and important (a whale customer).

IQR rule
Flag values below Q1 − 1.5·IQR or above Q3 + 1.5·IQR, where IQR = Q3 − Q1.

Clipping (winsorizing)

Capping values at chosen percentiles, e.g. the 1st and 99th.

Log transform

log1p(x) compresses a long right tail so extreme values pull less.

MCAR · pure chance

a sensor drops 1 reading in 100 at random

→ drop rows or impute; little bias either way

MAR · explained by data

older customers skip the age field more often

→ impute using other columns (median by group)

MNAR · hides the answer

high earners skip the income field

→ the gap is a signal: add a missing indicator

Always ask why a value is missing before choosing how to fill it.

Why it is missing decides what to do. MNAR gaps carry information — dropping them throws it away.

```python
import pandas as pd

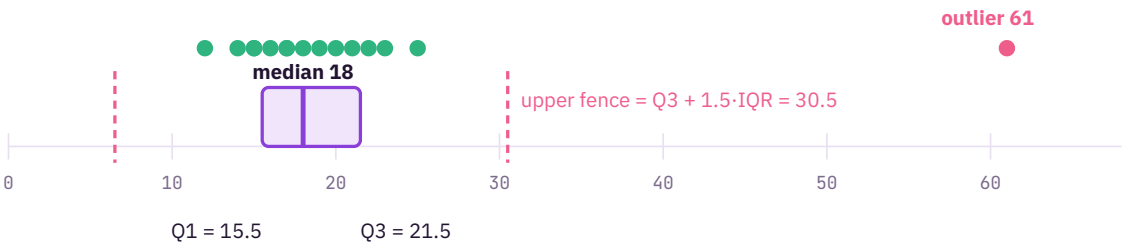
df = pd.read_csv("customers.csv")
fee = df["monthly_fee"]
print("missing:", fee.isna().sum())
print("mean   =", round(fee.mean(), 2), " median =", round(fee.median(), 2))

filled = fee.fillna(fee.median())
flag = fee.isna().astype(int)              # keep the fact that it was missing
print("after fill, missing:", filled.isna().sum(), " indicator sum:", flag.sum()
print("churn when fee missing:", round(df.loc[flag == 1, "churned"].mean(), 3),
      " when present:", round(df.loc[flag == 0, "churned"].mean(), 3))
```

```
# output
missing: 169
mean   = 44.94  median = 45.19
after fill, missing: 0  indicator sum: 169
churn when fee missing: 0.331  when present: 0.296
```

Strategy	When	scikit-learn
drop rows	few rows missing, at random	`df.dropna(subset=[...])`
median	numeric, skewed or with outliers	`SimpleImputer(strategy="median")`
most frequent	categorical	`SimpleImputer(strategy="most_frequent")`
constant	“unknown” is meaningful	`SimpleImputer(strategy="constant", fill_value="unknown")`
+ indicator	the gap may carry signal	`SimpleImputer(add_indicator=True)`
model-based	strong relationships between columns	`KNNImputer`, `IterativeImputer`

Outliers



IQR = Q3 − Q1 = 6. Fences don't delete data — they flag it for a decision.

The IQR rule. One extreme value moves the mean but barely moves the median and quartiles.

Try it: drag an outlier

largest value  clip at the upper fence




```
Q1 = 15.5  Q3 = 21.5  IQR = 6  upper fence = 30.5
mean = 21.00   median = 18
The outlier 61 pulls the mean 3.0 above the median; the median barely
moves.
```

```python
import numpy as np
import pandas as pd

df = pd.read_csv("customers.csv")
t = df["tenure_months"]
q1, q3 = t.quantile([0.25, 0.75])
upper = q3 + 1.5 * (q3 - q1)
print(f"Q1={q1:.1f}  Q3={q3:.1f}  upper fence={upper:.1f}  flagged={(t > upper).
print("skew raw  :", round(t.skew(), 2))
print("skew log1p:", round(np.log1p(t).skew(), 2))
clipped = t.clip(upper=t.quantile(0.99))
print("max raw / clipped:", t.max(), "/", round(clipped.max(), 1))
```

```
# output
Q1=8.1  Q3=23.1  upper fence=45.6  flagged=80
skew raw  : 1.5
skew log1p: -0.33
max raw / clipped: 89.4 / 60.2
```

★

Decide per column, write it down

For each column with gaps or outliers, record the rule (drop / median / clip at p99 / log) and why. That note becomes the spec your pipeline code implements — and what you defend in a review.

Common mistakes

Filling with the mean on skewed data · filling before splitting (the fill value then includes test rows) · deleting every outlier, including the real big customers · forgetting the missing indicator when the gap is informative.

INTERVIEW CORNER

Q. Mean or median imputation? **A.** Median for skewed or outlier-heavy numbers; the mean is pulled by extremes.

Q. How do you handle outliers? **A.** First decide error or real. Fix or drop errors; for real extremes, clip, log-transform or use robust models (trees).

Key takeaways

1. Ask why it is missing; MNAR deserves an indicator.
2. Median for numbers, most-frequent for categories.
3. Flag outliers with IQR; clip or log, don't blindly delete.

MODULE 3 OF 4

Encoding, scaling & new features

Turn every column into numbers a model can use fairly.



- One-hot vs ordinal
- Unseen categories
- Standardization and min-max
- Domain features

⏱ 3.5 hours · Area 03 · Data Prep

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

Models only read numbers

In plain words

Text categories like “basic/plus/pro” must become numbers. One clean way: a 0/1 column per category.

Think of it like

A tick-box form: tick exactly one box for your plan.

Worked example

plan	is_basic	is_plus	is_pro
plus	0	1	0
pro	0	0	1

The word for it

One-hot encoding.

Why it matters — Numbering the plans 0, 1, 2 would invent an order the model takes literally.

2 STEP 2 — Put features on the same scale

In plain words — A fee in tens and a call count in ones are not comparable. Rescale each column so its typical spread is 1.

Think of it like — Comparing heights in centimetres with weights in tonnes — convert first.

Worked example

1. fee mean 45, std 15
2. fee 60 → (60 − 45) / 15 = 1.0
3. fee 30 → (30 − 45) / 15 = −1.0

The word for it — **Standardization** (z-score).

Why it matters — Linear models, k-NN and neural nets learn faster and fairer on scaled inputs.

PAUSE & TRY — **Tenure 6 months and 2 support calls. Make a feature that captures "calls per month".**

Answer

2 / (6 + 1) ≈ 0.29 calls per month (the +1 avoids dividing by zero).

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Feature engineering
Creating inputs that make the pattern easier for the model to see.

One-hot encoding
One 0/1 column per category; no fake order between categories.

Ordinal encoding
Map ordered categories to integers (small=0, medium=1, large=2).

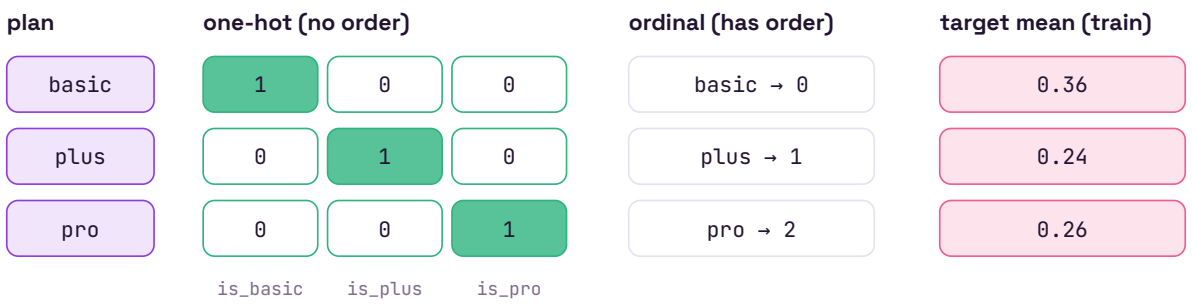
Target encoding
Replace a category with the mean target for it, computed on training folds only.

Standardization
z = (x − mean) / std per column: mean 0, std 1.

Min-max scaling
(x − min) / (max − min): squeezes into [0, 1]; sensitive to outliers.

Ratio / interaction feature
Combining columns, e.g. calls per month = support_calls / tenure.

Binning
Turning a number into ranges (0–6, 6–24, 24+ months).



One-hot for nominal categories · ordinal only when the order is real (small < medium < large)
Target encoding must be computed on training folds only (cross-fitting), or it leaks.

Pick the encoding that tells the truth. Ordinal codes for unordered categories invent an order the model will use.

Try it: encode a column

column plan encoding one-hot

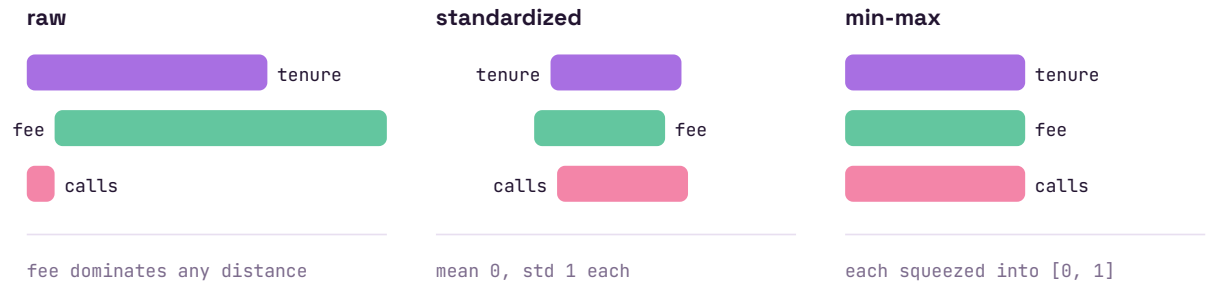
plan	plan_basic	plan_plus	plan_pro
basic	1	0	0
plus	0	1	0
pro	0	0	1
basic	1	0	0
plus	0	1	0

```
5 rows → 3 column(s)
3 columns, exactly one 1 per row; no fake order. A category never seen
in training becomes all zeros with handle_unknown="ignore".
```

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

df = pd.read_csv("customers.csv").dropna(subset=["region"])
enc = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
enc.fit(df[["plan", "region"]])
print(enc.get_feature_names_out())
new = pd.DataFrame({"plan": ["pro"], "region": ["central"]})   # "central" never
print(enc.transform(new))
```

```
# output
['plan_basic' 'plan_plus' 'plan_pro' 'region_east' 'region_north'
 'region_south' 'region_west']
[[0. 0. 1. 0. 0. 0. 0.]]
```



Scale so no feature shouts. Distance- and gradient-based models (k-NN, k-means, linear models, neural nets) need it; tree models don't.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

df = pd.read_csv("customers.csv")
X = df[["tenure_months", "support_calls"]]
z = StandardScaler().fit_transform(X)
m = MinMaxScaler().fit_transform(X)
print("standardized mean:", z.mean(axis=0).round(3), " std:", z.std(axis=0).roun
print("min-max range    :", m.min(axis=0), "→", m.max(axis=0))
```

```
# output
standardized mean: [0. 0.]  std: [1. 1.]
min-max range    : [0. 0.] → [1. 1.]
```

New features from domain sense

```
import numpy as np
import pandas as pd

df = pd.read_csv("customers.csv")
df["calls_per_month"] = df["support_calls"] / (df["tenure_months"] + 1)
df["new_customer"] = (df["tenure_months"] < 6).astype(int)
df["log_tenure"] = np.log1p(df["tenure_months"])
for col in ["support_calls", "calls_per_month", "new_customer"]:
    print(f"{col:16} corr with churn = {df[col].corr(df['churned']):+.3f}")
```

```
# output
support_calls     corr with churn = +0.296
calls_per_month   corr with churn = +0.294
new_customer      corr with churn = +0.075
```

Model family	Scale?	Encode categories?
Linear / logistic regression	Yes	One-hot
k-NN, k-means, SVM	Yes (critical)	One-hot
Neural networks	Yes	One-hot or embeddings
Decision trees, random forests, gradient boosting	No	One-hot or ordinal (native categorical in some libraries)

i

Feature engineering beats model shopping

On tabular data a good ratio or flag often adds more than swapping algorithms. Domain questions — “what does a frustrated customer do?” — lead to features like calls per month.

×

Common mistakes

Ordinal-encoding unordered categories · one-hot on a 10,000-value column (use hashing, target encoding or grouping) · scaling before splitting · computing target encoding on the full data (pure leakage) · forgetting `handle_unknown="ignore"`.

INTERVIEW CORNER

Q. One-hot vs label encoding? **A.** One-hot for nominal categories (no order); integer codes only for truly ordered categories or tree models.

Q. Do random forests need scaling? **A.** No — splits compare values within one feature, so monotonic rescaling changes nothing.

Key takeaways

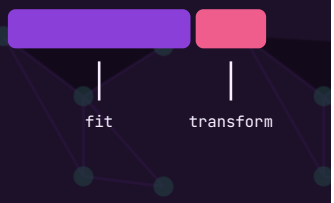
1. One-hot for nominal, ordinal for ordered.
2. Standardize for distance/gradient models; trees don't care.
3. Good domain features often beat a fancier model.

MODULE 4 OF 4

Splits & data leakage

Make the test score an honest preview of production.

- Train / validation / test
- Stratified and k-fold splits
- Target leakage and contamination
- Pipeline + ColumnTransformer



⏱ 2.5 hours | Area 03 · Data Prep

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

1 Keep an exam the model never studied

In plain words

Hide part of the data. Train on the rest. Score on the hidden part to see how the model does on new customers.

Think of it like

A teacher doesn't hand out the exam answers as homework.

✎ **Worked example**

set	rows	used for
train	1,600	learning
test	400	one final score

Aa **The word for it** **Train/test split**.

! **Why it matters** A score on training data only shows memory, not skill.

2 STEP 2 Don’t peek while preparing

≈ **In plain words** Anything that learns from data — a median, a mean for scaling — must learn from the training part only.

☀ **Think of it like** Calibrating the exam's difficulty using the students' exam answers.

✎ **Worked example**

1. split first
2. scaler.fit(train)
3. scaler.transform(train), scaler.transform(test)

Aa **The word for it** **Train-test contamination**.

! **Why it matters** Peeking makes test scores optimistic; production then disappoints.

PAUSE & TRY **You remove duplicates, then fill missing ages with the median, then split. Which step is in the wrong place?**

Answer

The median fill — split first, compute the median on training rows only. (Removing duplicates before splitting is fine and wise.)

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Training set
Rows the model and every preprocessing step learn from.

Validation set
Rows used to compare models and tune hyperparameters.

Test set
Rows held back until the end, used once for an honest estimate.

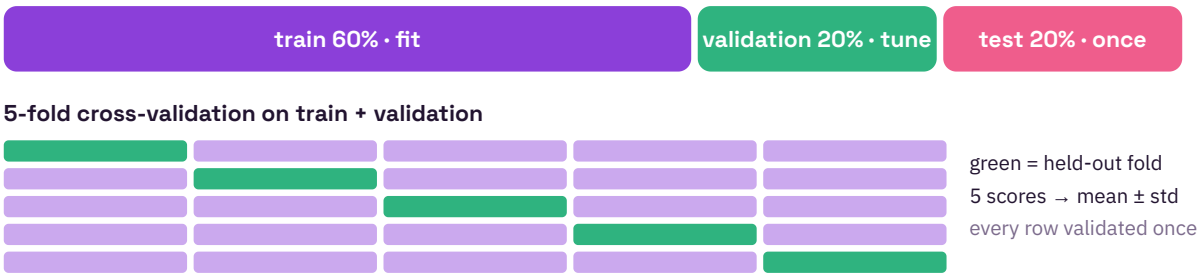
Stratified split
Keeps the class mix (e.g. 30% churn) the same in every split.

Cross-validation
Rotate the held-out fold k times and average the scores.

Data leakage
Information unavailable at prediction time reaching the model during training.

Target leakage
A feature that is set because of, or after, the outcome (refund_issued).

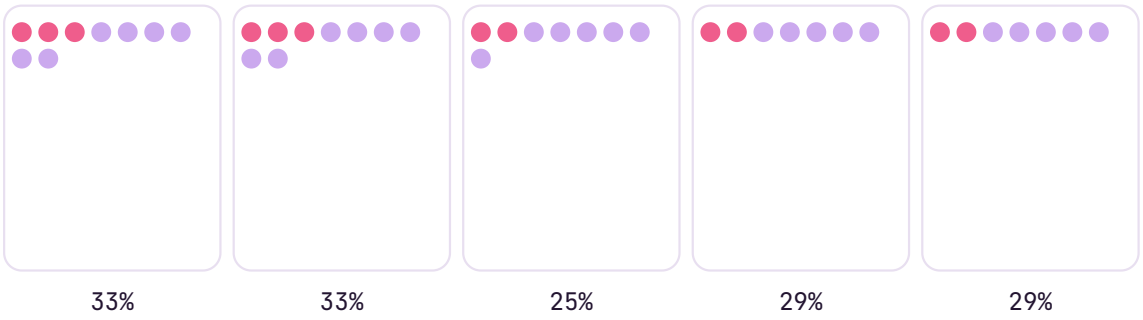
Train-test contamination
Test rows influencing training, e.g. a scaler fitted on all rows, or duplicates across splits.



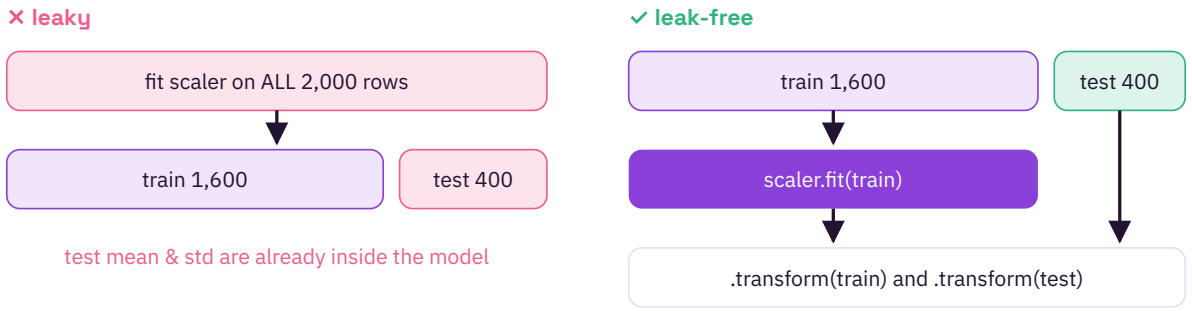
Three jobs, three sets. Cross-validation reuses data for tuning without touching the test set.

Try it: split and stratify

folds k ☑ stratify




```
40 rows, 12 churners (30%), sorted by label · 5 folds · stratified
churn rate per fold: 33%  33%  25%  29%  29%
Every fold keeps close to the overall 30% churn rate.
```



Rule: anything that learns from data (imputer, scaler, encoder, feature selector) is fitted on training rows only.

Split first, fit second. The same fitted object transforms every split.

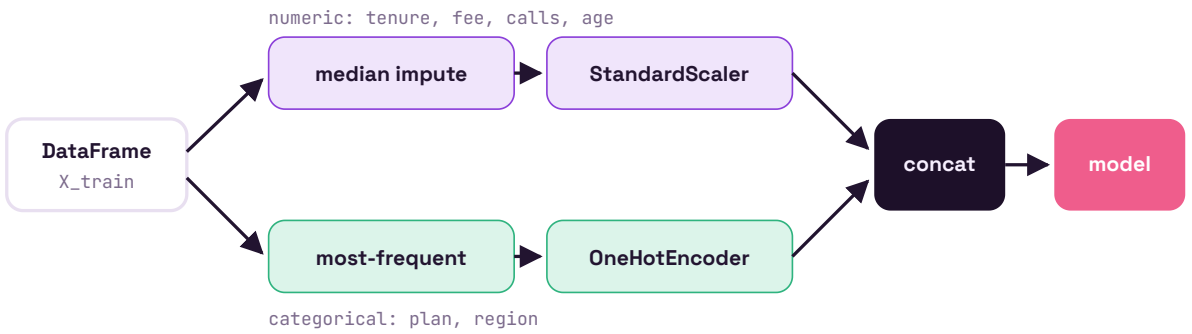
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("customers.csv")
X, y = df[["tenure_months", "support_calls"]], df["churned"]
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.2, stratify=y, rando
print("churn rate  train / test:", round(y_tr.mean(), 3), "/", round(y_te.mean()

scaler = StandardScaler().fit(X_tr)           # learn mean/std from training ro
print("train means used:", scaler.mean_.round(2))
print("test mean after transform:", scaler.transform(X_te).mean(axis=0).round(3)
```

```
# output
churn rate  train / test: 0.299 / 0.3
train means used: [17.54  1.67]
test mean after transform: [-0.062 -0.101] (near, not exactly, 0)
```

One object for the whole recipe: Pipeline + ColumnTransformer



Columns in, model-ready matrix out. Wrapping preprocessing and model in one Pipeline makes leakage hard by construction.

```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler

df = pd.read_csv("customers.csv")
num = ["tenure_months", "monthly_fee", "support_calls", "age"]
cat = ["plan", "region"]
pre = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")), ("sc", StandardScaler())]), num),
    ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                      ("oh", OneHotEncoder(handle_unknown="ignore"))]), cat),
])
model = Pipeline([("pre", pre), ("clf", LogisticRegression(max_iter=1000))])
scores = cross_val_score(model, df[num + cat], df["churned"], cv=5, scoring="roc_auc")
print("5-fold AUC:", scores.round(3), " mean =", round(scores.mean(), 3))
```

```
# output
5-fold AUC: [0.784 0.753 0.728 0.727 0.715]  mean = 0.742
```

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

df = pd.read_csv("customers.csv").fillna({"monthly_fee": 45, "age": 41})
honest = ["tenure_months", "support_calls", "monthly_fee"]
leaky = honest + ["refund_issued"]
for name, cols in [("honest", honest), ("with refund_issued", leaky)]:
    auc = cross_val_score(LogisticRegression(max_iter=1000), df[cols], df["churned"], cv=5, scoring="roc_auc").mean()
    print(f"{name:20} AUC = {auc:.3f}")
```

```
# output
honest               AUC = 0.730
with refund_issued   AUC = 0.931
```

Time-ordered data needs a time split

If rows are events over time (transactions, sensor readings), a random split lets the model peek at the future. Train on the past and test on the most recent period (`TimeSeriesSplit`).

The lab: make the pipeline honest



Three bugs, three kinds of dishonesty: a leaky column, a test-set leak and a crash on unseen categories.

Common mistakes

Fitting any transformer before the split · random-splitting time series · tuning on the test set · duplicates of the same customer in train and test (use `GroupShuffleSplit`) · not stratifying a rare class.

INTERVIEW CORNER

Q. What is data leakage? Give two examples. **A.** Information unavailable at prediction time used in training: a post-outcome column like refund_issued; a scaler fitted on all rows before splitting.

Q. Why wrap preprocessing in a Pipeline? **A.** Cross-validation then refits preprocessing inside each fold, so no fold's held-out rows leak into fitting; deployment uses the identical recipe.

Key takeaways

1. Split first; fit on train; transform everything.
2. Stratify classification splits; time-split temporal data.
3. Pipeline + ColumnTransformer makes leakage hard.

PRACTICE

Check yourself

Answer in your head, then open each card. The area test has 20 more.



Which column in customers.csv should never be a feature, and why?

refund_issued — it is set after a customer churns, so it is unknown at prediction time (also customer_id: an id).

A skewed income column has 5% missing. Mean or median?

Median — the mean is pulled by the long tail.

Q1 = 10, Q3 = 30. Is 65 an outlier by the IQR rule?

Yes: upper fence = 30 + 1.5 × 20 = 60.

Plan has basic/plus/pro with no real order. Encoding for logistic regression?

One-hot.

Does a random forest need scaled inputs?

No — tree splits are unaffected by monotonic rescaling.

You fit StandardScaler, then split. What's wrong?

The scaler's mean and std include test rows: train-test contamination.

1% of transactions are fraud. Which split option matters?

stratify=y, so train and test both keep ~1% fraud.

Orders over 2 years; predict next month. How to split?

By time: train on earlier months, test on the latest.

REFERENCE

Interview questions

Asked in ML engineer and data scientist interviews. Try an answer out loud before opening.



Walk me through your data cleaning process.

EDA (shape, types, gaps, spread, target), a per-column decision log, then a scikit-learn Pipeline that implements it, fitted on train only.

How do you handle missing data?

Understand why (MCAR/MAR/MNAR), drop only if rare and random, otherwise impute (median/mode/model) and add an indicator when the gap is informative.

What is target leakage? How do you detect it?

Features caused by the outcome; detect with suspiciously high scores, single-feature importance, and asking when each column is recorded.

Standardization vs normalization?

Standardization: mean 0, std 1; min-max normalization: range [0, 1], more outlier-sensitive.

High-cardinality categorical feature — what do you do?

Group rare levels, hashing, frequency or cross-fitted target encoding, or embeddings in neural nets.

Why use cross-validation?

A more stable estimate than one split and uses all data for validation; essential for tuning on small data.

How do you prevent leakage in cross-validation?

Put all preprocessing in a Pipeline so it is refitted inside every fold.

Class imbalance in the split — what do you do?

Stratify; evaluate with precision/recall/PR-AUC (Area 05); consider class weights or resampling on training folds only.

REFERENCE

Glossary

Every term used on this page, A to Z.

A a

Binning

Turning numbers into ranges.

Cardinality

Number of distinct values in a column.

ColumnTransformer
Applies different preprocessing to different columns.

Contamination
Test information influencing training.

Cross-validation
Rotating held-out folds and averaging scores.

Data dictionary
Meaning, unit and timing of each column.

Data leakage
Information unavailable at prediction time used in training.

EDA
Exploratory data analysis.

Feature engineering
Creating new model inputs.

Imputation
Filling missing values.

Interaction feature
A feature combining two or more columns.

IQR
Q3 − Q1, the middle 50% spread.

Log transform
log1p(x) to compress right tails.

MAR
Missing at random given observed data.

MCAR
Missing completely at random.

Min-max scaling
Rescale to [0, 1].

Missing indicator
0/1 flag for a missing value.

MNAR
Missingness depends on the missing value itself.

One-hot encoding
One 0/1 column per category.

Ordinal encoding
Integer codes for ordered categories.

Outlier
A value far from the rest.

Pipeline
A chain of preprocessing steps and a model fitted as one.

Standardization
Scale to mean 0, std 1.

Stratified split
Split preserving class proportions.

Target encoding
Category replaced by its training target mean.

Target leakage
A feature caused by the outcome.

Test set
Held-out data used once at the end.

Validation set
Held-out data for tuning.

REFERENCE

FAQ & troubleshooting

The problems learners hit most in this area, and the fix for each.



You see	Why	Fix
ValueError: Found unknown categories	A category appears at transform time that fit never saw	`OneHotEncoder(handle_unknown="ignore")`
ValueError: Input X contains NaN	Model received missing values	Add a `SimpleImputer` step before the model
could not convert string to float	A categorical column reached a numeric step	Route it through the categorical branch of the ColumnTransformer
Test score much higher than expected	Leakage: post-outcome column or fitting before the split	Check when each feature is known; fit inside a Pipeline
Cross-validation scores vary a lot	Small data or rare class	Stratified k-fold; more folds or repeated CV; report mean ± std
SettingWithCopyWarning	Editing a slice of a DataFrame	Use `.loc` or `.copy()` on the slice

REFERENCE

Further reading

Go deeper when you are ready.



Resource	Why read it
scikit-learn User Guide, “Preprocessing data” and “Common pitfalls and recommended practices” (scikit-learn.org)	Official guide to scalers, encoders, imputers and leakage
Zheng & Casari, *Feature Engineering for Machine Learning* (O'Reilly)	Practical encodings, scaling and text features
Kuhn & Johnson, *Feature Engineering and Selection* (free at feat.engineering)	Missing data, encodings and resampling in depth
Kaufman et al., “Leakage in Data Mining: Formulation, Detection, and Avoidance” (ACM TKDD, 2012)	The classic paper on how leakage happens

HANDS-ON LAB · ABOUT 1.5 HOURS

Lab 03 · A leak-free preprocessing pipeline

`pipeline.py` prepares 2,000 customers for a churn model with scikit-learn. It runs cleanly and reports an impressive AUC of 0.935 — which is a lie. One bug leaks the answer, one leaks the test set into fitting, one crashes on a region the model has never seen.

1. `pip install -r requirements.txt`, then `python check.py`.
2. Check 1 and 5: which column is only known after a customer churns? Exclude it in `split_columns`.
3. Check 3: the imputer's medians match all rows. Move the fit so it only sees training rows.

4. Check 4: a new region crashes `transform`. Which `OneHotEncoder` option fixes it?
5. Re-run after each fix until all five checks are ok — and notice the AUC drop to an honest number.

Done when you see `PASS: all 5 checks passed — leak-free pipeline, honest test AUC = 0.766`.