

Data Acquisition, Processing and Mining for AI

Week 06 — Data cleaning and leakage

Silviu Maniu

Acting on what you found

acquire → understand → clean → mine

- Week 5: written findings
- Today: acting on them (cleaning), then leakage
- Output: a cleaning pipeline

A title ending in (*) marks optional material: context or an extension, dropped first if we run short.

Cleaning decisions

- There is no `clean(df)`. Every choice **changes what the dataset is about**, and the right choice depends on **the question**
- For **every** change, record:
 1. **what** you changed,
 2. **why**,
 3. **how many rows** it touched
- **Never clean in place**. The original `df` stays as acquired; `clean` is a new frame
- **The pipeline is the deliverable**, not the cleaned file

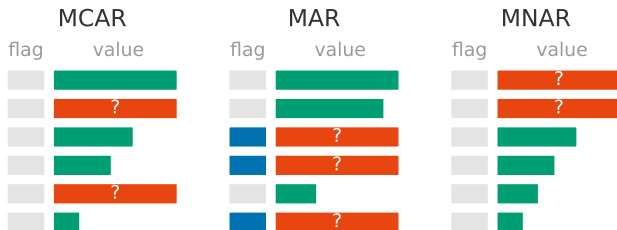
From findings to decisions

finding (week 5)	decision	rows
impossible : negative fare, duration ≤ 0 , over 100 miles	drop	8 412
ambiguous : 0.00 miles or 0 passengers	flag , keep	6 603
Flex Fare: no taxi meter, five fields missing	keep, indicator , impute	31 054

- Ranked **most important first**: the impossible rows took the fare, resulting in distance correlation from **0.891 to 0.006**
- Applied in this order, each step sees **fewer rows**: 903 ambiguous rows are also impossible
- Each decision is **one step** of the pipeline, with its row count

Missing values

MCAR, MAR, MNAR



	the gap	what to do
MCAR	unrelated to anything	drop: costs size only
MAR	depends on other columns in the data	model it, or condition on it
MNAR	depends on the missing value itself	no fix in the data: model it, or report it

- The taxi gap is **MAR**: payment_type 0 explains it entirely → **keep and condition**
- MNAR: **the data cannot usually show it**; only domain knowledge can help

Imputation and its assumptions

choice	what it assumes
drop the rows	they were exchangeable with the rest
drop the column	it carried nothing you need
fillna(0)	zero is a real value here: usually false
mean / median	the typical value is a fair guess
group-wise median	within the group, rows are exchangeable
model-based	the other columns predict it

- **Imputation**: filling a missing value with an estimate
- Each choice is an assumption: record it
- Imputation reduces the variance: the invented values sit at the centre. **Never impute a target**

Imputing passenger_count

	rows	mean	sd	trips with 0 passengers
dropna()	168 946	1.30	0.75	0.8 %
fillna(0)	200 000	1.10	0.83	16.2 %
fillna(median)	200 000	1.25	0.70	0.7 %

- `fillna(0)`: 31 054 Flex Fare trips become **trips with no passenger**.
“Unknown” turns into **“none”**, and the mean drops by 15 %
- `fillna(median)`: the mean barely moves, but the **sd shrinks** ($0.75 \rightarrow 0.70$):
31 054 identical values at the centre

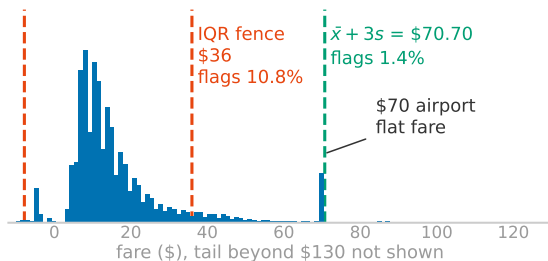
The missing indicator

```
clean["flex_fare"] = clean["passenger_count"].isna()  
median_metered = (clean.loc[~clean["flex_fare"], "passenger_count"]  
                  .median())  
clean["passenger_count"] = (clean["passenger_count"]  
                           .fillna(median_metered))
```

- flex_fare is the **indicator column**: one bit that keeps what imputation destroys, **which values were real**
- Under MAR it **carries signal**: “Flex Fare trip”
- It makes the imputation **auditable**: one can recompute or undo it
- Add it **before** imputing

Outliers and duplicates

Outlier rules: IQR and z fences



- **IQR rule** (Tukey): with Q_1 , Q_3 the quartiles and $IQR = Q_3 - Q_1$, flag outside $[Q_1 - 1.5 IQR, Q_3 + 1.5 IQR]$. **Uses only ranks** → works on skewed data
- **z-score**: flag $|x - \bar{x}|/s > 3$. **Assumes roughly normal**, and **the outliers inflate s** , so extreme values can hide themselves
- Same fares: the IQR fence flags **eight times the rows**
- Neither is an objective definition of “wrong”

What the fences flag

rule	fence (\$)	fares flagged	\$70 airport fares flagged
IQR	-7.83 to 35.98	21 548 (10.8 %)	all 5 207
$\bar{x} \pm 3s$	-37.04 to 70.70	2 718 (1.4 %)	none

- The **airport flat fare** is \$70: ordinary for an airport trip, **an outlier** for the IQR rule
- The z fence sits at \$70.70, **60 cents above** the flat fare: a small shift in the mean or s flips all 5 207
- A fence **knows the distribution, not the trip data**: look at what was flagged before acting

Flag, cap, or drop

- **Flag** (add a boolean column): the default. Nothing is lost, and later steps can choose
- **Cap** (**winsorize** = pull values in to a percentile): keeps the row, removes its influence. **Record it**: the maximum of the data is now “fake” (you chose it)
- **Drop**: only for the **impossible** (a duration of zero or less)

Capping the fares

	max (\$)	mean (\$)	sd (\$)	median (\$)
as is	800.00	18.06	16.82	12.80
capped at the 99th percentile	73.00	17.77	14.82	12.80

- 1 906 fares (1 %) set to \$73.00. No row is removed
- The median **does not move**; the sd drops by 12 %
- The \$596.60 trip of 83 miles, **extreme but real**, is now recorded as \$73.00

Every value normal, the row impossible

pickup	miles	fare (\$)	duration	speed
31 Jan 13:42	3.50	25.50	4 s	3 150 mph
28 Jan 12:01	4.57	2.90	8 s	2 057 mph

- Each value passes **its own IQR fence** and our constraints (duration above 0, distance up to 100 miles)
- Together they are **impossible**: no taxi covers 3.5 miles in 4 seconds
- A **derived column** (speed, price per mile) makes the impossibility obvious

Outliers over several columns (*)

- **Model-based detection** scores a row on **all its columns at once**: an **isolation forest** splits at random and counts how few cuts it takes to isolate a point; odd points are isolated early. **Local outlier factor** compares how crowded a point's neighbourhood is to its neighbours'
- They return **a score**, but the threshold is a parameter. It **does not say why** a row is out of place.
- Useful to **find candidates** on wide tables (40 columns cannot be checked manually). **Manual look still needed**

Duplicates and entities

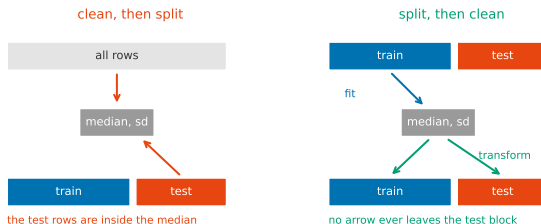
- **Exact duplicates**: `df.duplicated().sum()`. The taxi sample has **zero**
- **The same entity twice**: two rows, different ids, same patient. `duplicated()` does not work
- **Entity resolution**: deciding which records are **the same thing** in the world. Varied research field: blocking, string similarity, learned matchers
- Duplicate entities split **across train and test** make the evaluation wrong

Entity resolution (*)

- Comparing every row to every other is $O(n^2)$: 200 000 rows is 20 billion pairs. So the first step is **blocking**: only compare rows that already share something cheap (postcode, first three letters, year)
- Then **score the pairs**: string similarity for names, numeric distance for amounts, exact match for identifiers
- Then **decide**, with a threshold or a trained classifier, and **decide what is the cost of a false merge**: merging two real people is usually worse than missing a duplicate

Leakage

Split before cleaning



- **Train/test split:** fit on one part, measure on data the model has never seen. The test set simulates the future
- So everything learned from the data is learned from the training part only: the median, the scaler's mean, the categories. These numbers are then applied unchanged to both parts
- The bug: impute or scale before splitting → the test set has influenced the model, and the score becomes too optimistic

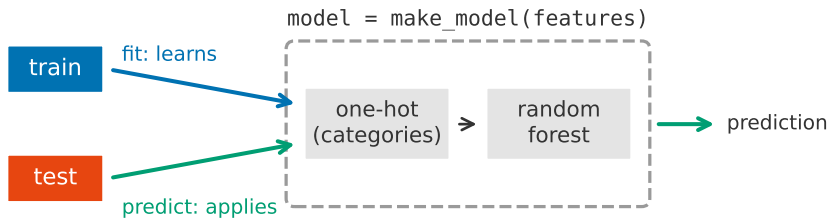
Standardising: fit and transform

```
X_train, X_test = train_test_split(X, test_size=0.3, random_state=0)

mean = X_train["trip_distance"].mean()    # fit: learned from train only
sd = X_train["trip_distance"].std()
train_z = (X_train["trip_distance"] - mean) / sd    # transform
test_z = (X_test["trip_distance"] - mean) / sd    # same mean, same sd
```

- fit: mean and sd from **train only**, 3.216 and 4.320 miles. transform: apply **these two numbers** to any row
- Why: in use, one new trip arrives, say 5.0 miles. There is no set to average over: it becomes $(5.0 - 3.216)/4.320 = 0.41$, with the stored numbers
- The test set plays that new trip. Mean and sd taken from the test set use information the model will never have → **the score is too optimistic**

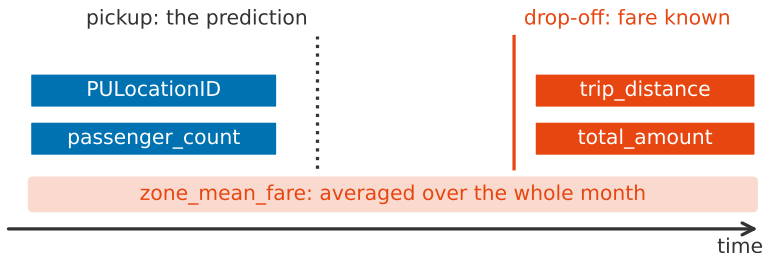
Pipeline



- **Pipeline**: the preparation steps and the model as **one object**. The lab gives it: `make_model(features)`
- `fit` on train: **every step learns**. Predict on test: **every step applies** what it learned
- **Random forest**: a standard classifier (many decision trees, voting), used here as a black box

- **Leakage**: information about the target reaching the model that **would not be available when the prediction is actually made**
- The symptom is **a score that is too good**. Model is excellent in testing and **useless in production**
- Invisible in the code and in the metrics: **only visible by seeing where each column came from**
- Kaufman, Rosset & Perlich cite it among **the top ten data mining mistakes**.

1. Target leakage



- A feature that is **a consequence of the outcome**, recorded **after** the thing being predicted happened
- **Timestamp is a good indicator**: for every column, **when was this value written, relative to the outcome?**

Target leakage in the taxi file

fare	tip	charges	total
18.40	4.48	4.00	26.88
19.80	5.95	4.00	29.75
12.80	0.00	7.25	20.05
56.20	14.58	16.69	87.47

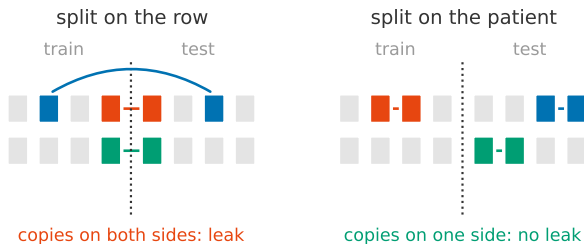
- Target: fare_amount. Feature: total_amount, $r = 0.976$
- The total is **computed from the fare**: fare + tip + charges
- Same for tip_amount: written **after** the fare

When is each column written?

column	written	known when the trip starts?
tpep_pickup_datetime, PULocationID	pickup	yes
passenger_count	pickup	yes
trip_distance, duration_min	drop-off	no
tip_amount, total_amount	payment	no

- Task: **predict the fare when the trip starts**, for example for app pricing
- trip_distance (correlation with fare $r = 0.891$) is **written at drop-off** so it is not usable
- For another task (checking a fare **after** the trip), the same column is legitimate: **leakage depends on the task**

2. Train-test contamination



- The **same information** lands on both sides of the split, so the model uses **something it memorised**
- **Duplicate entities** are common: one patient, two scans, one in train and one in test. The model **recognises** the patient
- Also: **scaling or imputing before the split, selecting features on the full dataset, tuning against the test set**
- **Fix: split on the entity**, and split **before** doing anything on the data

3. Temporal leakage

- A feature computed with **information from after the moment of prediction**, even when every individual value is correct
- **Target encoding**: replacing a category with **the mean outcome of that category**. When computed over the **whole file** = the outcome is inside the feature
- Same shape: a rolling average over the full series, a **random** split on time-ordered data
- Fix the feature: use **only earlier rows**. A trip's zone mean fare averages the zone's trips **before it**. Leaving the row itself out is the minimum
- Fix the split: **split by time**, e.g. train on 1–24 January, test on 25–31

Target encoding on six rows

zone	fare	with the row	without
A	10	12	13
A	14	12	11
A	12	12	12
B	20	25	30
B	30	25	20
C	70	70	—

- Zone C has **one trip**: its mean fare **is** that trip's fare
- With the row: the feature contains **the answer**, a little in A, entirely in C
- Without the row (**leave-one-out**): $(\text{sum} - \text{own}) / (\text{count} - 1)$. C has no other row $\rightarrow$ no value
- In the taxi sample, **9 of 241** pickup zones have a single trip

Leakage in practice: KDD Cup 2008

- KDD Cup 2008, breast-cancer detection: the patient ID was strongly predictive. It was a trace of which hospital's records a patient came from, and some sources held mostly cancer patients. An ID number "predicted" cancer
- The bug: the identifier carried the provenance of the join, which correlated with the target
- An ID is usually not a feature
- Leakage can enter through how the dataset was assembled

Cross-validation

- A single train/test split gives **one number from one random partition**. Draw other test rows and it changes
- **k -fold cross-validation**: cut the data into k parts, train on $k - 1$ and test on the **held-out** one, **k times**. Every row is tested exactly once → **a spread**
- The pipeline is **refitted on each training fold**
- **Report the spread**: mean $\pm$ sd over the folds.
- **The folds respect the same boundary as the split**: all rows of one patient **in the same fold**, and with time **fold forward** (train on the past, test on the future)

One model, ten random splits

split	1	2	3	4	5	6	7	8	9	10
test R^2	0.80	0.69	0.77	0.89	0.73	0.85	0.83	0.77	0.79	0.85

- Fare from distance, 1 000 trips, a linear regression; R^2 = share of the variance explained (1 = perfect)
- Same data, same model: 0.69 to 0.89, depending on what is in the test set
- A single one is only a sample. Mean and sd: 0.80 ± 0.06

Detecting leakage

- A very good score is suspicious. On a hard problem, 0.99 is likely a bug
- Look at what the model used. Assume one feature carrying nearly all the importance is a leak
- The provenance, per column: when was it written, and is it available at prediction time?
- Remove suspects one at a time. The scores will probably drop

Lab and next week

Today's lab

- **M2 before next week:** your findings, the cleaning notebook, and data card
- **Three mandatory exercises:** the cleaning **pipeline**, removing the **leaks**, and the reason for the **campaign rate** leak
- Solved: reading the findings, encoding and scaling recipes, churn dataset, decisions table
- The churn dataset has **three leaks**, one of each kind

Before next week

- M2: apply the same pipeline to **your own data**, and add the decisions to your data card.
- See the provenance of your columns.
- Next lecture: **clustering**

References

- Rubin, *Inference and Missing Data*, Biometrika 63(3), 1976 – MCAR / MAR / MNAR.
<https://doi.org/10.1093/biomet/63.3.581>
- Kaufman, Rosset & Perlich, *Leakage in Data Mining: Formulation, Detection, and Avoidance*, ACM TKDD 6(4), 2012 (KDD 2011) – the KDD Cup 2008 patient-ID case.
<https://doi.org/10.1145/2382577.2382579>
- Tukey, *Exploratory Data Analysis*, 1977 – the $1.5 \times \text{IQR}$ rule
- scikit-learn, *Common pitfalls and recommended practices*:
https://scikit-learn.org/stable/common_pitfalls.html
- scikit-learn Pipeline: <https://scikit-learn.org/stable/modules/compose.html>