

Data Acquisition, Processing and Mining for AI

Week 05 — EDA for data quality

Silviu Maniu

What EDA is for

acquire → understand → clean → mine

- Weeks 1–4: getting data from other machines
- Today: finding what is wrong with the data
- Output: written findings

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

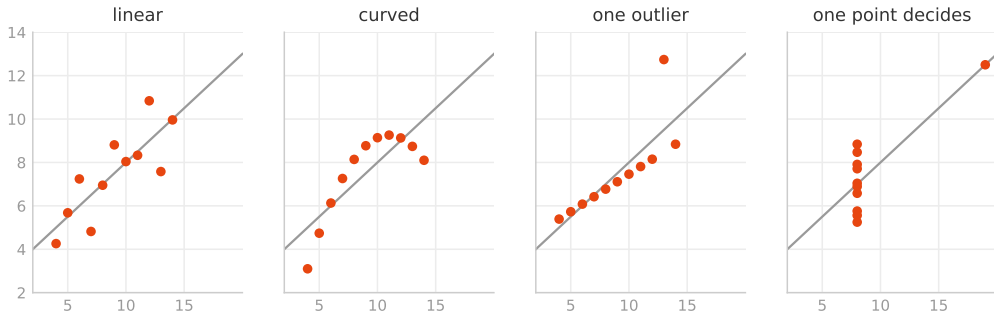
- **EDA** (exploratory data analysis): learning **what the data is**, before any mining or modeling
- Four things to look for:
 - values that are **impossible** (a negative fare)
 - values that are **missing**, and **why**
 - **shape**: what is normal, what is extreme
 - who or what is **not in the data at all**
- **No model tuning**, findings go first

The template

- The questions are **fixed**:
 1. **Missingness**: which columns, how much, and **is it random?**
 2. **Distributions and outliers**: what is typical, what is impossible
 3. **Temporal patterns**: what changes over time, what is a gap
 4. **Representativeness**: who is under- or un-represented
- Decision: **what to do about it**, or that nothing will be done

Summary statistics and issues

Anscombe's quartet



- **Anscombe's quartet (1973):** same mean $x = 9.00$, mean $y = 7.50$, sd $y = 2.03$, same correlation $r = 0.816$, and the same fitted line
- Linear; curved; linear plus one bad row; a single point shifting the line
- Summary statistics compress and hide the shape of the data
- Always plot first, then summarise

The first commands

```
print("shape:", df.shape)
print(df.dtypes.to_string())
print(df[["fare_amount", "trip_distance", "passenger_count"]]
      .describe().round(2).to_string())
missing = df.isna().sum()
missing = missing[missing > 0]
print(missing.to_string())
```

- describe(): read **min and max first**. A **negative minimum** or an impossible maximum are detected here
- object dtype on a numeric column = **some values are not numbers**
- **Before** plotting

Mean and median

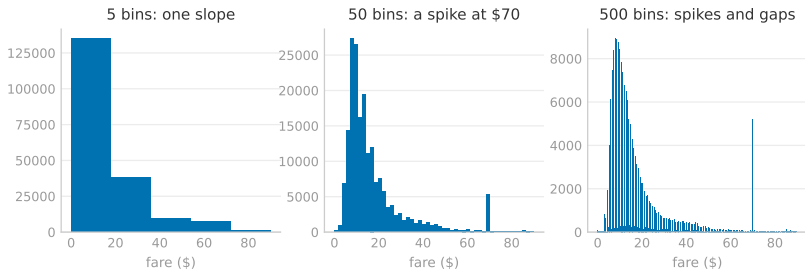
trip	fare (\$)	one typo
1	18.40	18.40
2	19.80	19.80
3	70.00	700.00
4	12.80	12.80
5	56.20	56.20
mean	35.44	161.44
median	19.80	19.80

- The **first five fares** of our file
- Mean \$35.44: **above three of the five**, pulled up by the two long trips
- One typo (70.00 $\rightarrow$ 700.00): mean $\times 4.6$, median **unchanged**
- **Robust** statistic: one that **a few extreme values cannot move**, such as the median

Mean and median

- **Skew**: a distribution with a long tail on one side. Fares, incomes, file sizes, durations are **usually skewed right**
- The **mean follows the tail**; the **median does not**. Our 200 000 fares: **mean \$16.83, median \$12.21**
- **“Typical”** → the median, with the number of rows it came from
- **Total** → the mean (to be able to multiply with the number of rows)
- **Neither replaces the distribution**, need visualisation or quartiles

Histograms: the bin width



- **Histogram**: values counted into **bins**. The number of bins is a **parameter**
- Too few bins **can hide information**; too many **can lead to noise**
- Only the middle one shows: **5 207 trips at exactly \$70** (the airport flat fare)
- Rule of thumb: **Try three bin counts before anything**

Missingness

NaN and sentinels

- NaN: the value is explicitly **absent**
- **Sentinel**: a normal value that means “nothing”: 0, -1, 999, "N/A", "Unknown", the empty string
- Sentinels **look like data**, **pass type check**, and **may contribute to averages**
- First check: “**what does this column use to mean nothing?**”

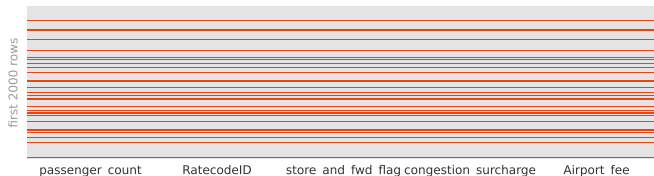
Sentinels in the taxi file

column	value	meaning (data dictionary)	rows
passenger_count	NaN	absent	31 054
RatecodeID	99	Null/unknown	2 418
PULocationID	264	borough "Unknown"	470
payment_type	5	Unknown	1

- `df.isna().sum()` sees **only the first row** and the highlighted rows are **invisible**
- RatecodeID 99 is a **number** and it behaves like a real rate code
- Only the **data dictionary** (the source's documentation) defines what the code means

Missingness map

```
affected = list(missing.index)
nulls = df[affected].isna()
sns.heatmap(nulls.head(2000), cbar=False,
            cmap=["#E4E4E4", "#E74610"], ax=ax)
```



- `sns.heatmap` (using **seaborn**, a plotting library), one cell per value, orange = missing
- **Five columns missing on the same 31 054 rows (15.5% of the file) → one single problem**
- Missingness as a **pattern over rows** → evidence of a **mechanism**

Five example rows

row	payment type	passengers	rate code	store and forward	congestion fee	fare
0	1	1	1	N	2.50	18.40
1	1	1	1	N	2.50	19.80
14	0	NaN	NaN	NaN	NaN	-4.75
24	0	NaN	NaN	NaN	NaN	7.88
27	0	NaN	NaN	NaN	NaN	20.50

- Rows 14, 24, 27: the missing fields are missing together, payment_type is 0 each time
- Rows 0 and 1: nothing missing, payment_type 1 (credit card)
- Five rows give a hypothesis; need to check over all rows

Finding the mechanism

```
gone = df["passenger_count"].isna()
print(pd.crosstab(df["VendorID"], gone, colnames=["missing"])
      .to_string())
```

missing VendorID	False	True
1	38521	5223
2	130360	25809
6	0	22
7	65	0

- VendorID: missing rows under **both** main vendors → not the cause
- payment_type: **all 31 054 missing rows are 0**, and **no other row is**
- payment_type 0 = **Flex Fare**: a **price set up in an app**, no taxi meter → the meter fields **do not exist**
- Finding: **“the 15% represent a different product”**

MCAR, MAR, MNAR

- **MCAR** (missing completely at random), the gap is unrelated to anything → dropping those rows costs **size only**
- **MAR** (missing at random), the gap depends on **other columns in the data**, as `payment_type` does in our example → **model it, or condition on it**
- **MNAR** (missing not at random): the gap depends on **the missing value itself** (the highest earners skip the income question). **The data cannot usually show it**; only domain knowledge can help
- **The mechanism has an influence on what to do with the missing rows**

Issues with dropna()

- dropna() here removes every Flex Fare trip → the dataset silently becomes “street taxis only”
- The numbers move: median fare \$14.11 among the dropped rows, \$12.10 among the kept ones → cheaper than reality
- Dropping rows is a modelling decision, changes who the dataset is about

Outliers, missing values, and correlation

Impossible, extreme, ambiguous

- **Impossible:** violates the constraints of the data. 8 284 negative fares, 121 non-positive durations (5 ending *before* they start, 116 lasting 0 seconds), 8 trips over 100 miles: 8 412 rows
- **Extreme but real:** \$596.60 for 83 miles, trip of two hours. Rare but possible → keep them
- **Ambiguous:** 5 231 trips of exactly 0.00 miles, 1 423 with 0 passengers: 6 603 rows in all (903 of them also impossible). Meter problem or cancelled? Impossible to get from the file
- Delete impossible, keep the extreme, **decide what to do with the ambiguous**

Examples of impossible, extreme, ambiguous

pickup	miles	minutes	passengers	fare (\$)	
18 Jan 21:07	1.48	11.90	1	-12.10	impossible
21 Jan 16:01	4.29	-0.33	NaN	2.90	impossible
21 Jan 09:19	222 167.49	26.00	NaN	31.19	impossible
22 Jan 15:09	83.02	106.18	2	596.60	extreme
26 Jan 19:25	0.00	0.25	1	800.00	ambiguous
9 Jan 18:33	1.00	8.30	0	9.30	ambiguous

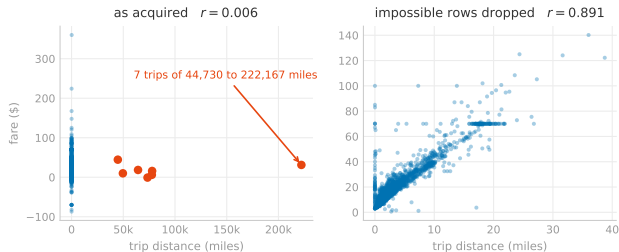
- The **largest fare**, \$800, is a **0-mile, 15-second** trip: **ambiguous**
- \$596.60 for 83 miles in 106 minutes is **consistent** → keep

Correlation: example on 6 trips

trip	miles	fare (\$)
1	2.32	18.40
2	2.18	19.80
3	18.00	70.00
4	1.30	12.80
5	14.14	56.20
6	222	167.49
	167.49	31.19

- **Pearson correlation** r : how close the points are to a **straight line**, from -1 to 1
- Trips 1–5: $r = 0.998$, fare proportional to distance
- With trip 6: $r = -0.075$
- One row flips the correlation: r is **dominated by extremes**

Correlation and outliers



- Fare against distance, all 200 000 trips, $r = 0.006$ → “distance does not affect the fare”
- After dropping the impossible rows, 8 412 of 200 000: $r = 0.891$. **Opposite conclusion**
- Put back **only** the trip of 222 167 miles: $r = 0.009$
- **Never report a correlation without looking at the scatterplot**

Time patterns

Time patterns

```
by_hour = (clean["tpep_pickup_datetime"].dt.hour
            .value_counts().sort_index())
by_day = clean.groupby(clean["tpep_pickup_datetime"].dt.date).size()
```

- `.dt` = the **datetime accessor**: the parts of a timestamp (`.hour`, `.date`, `.day_name()`) as ordinary columns to group by
- Usual questions: **what are the cyclical patterns** (hour of day, day of week), and **what is missing** (a day with no rows = a gap)
- Here: trips peak at **18h** and bottom out at **04h**, over **31 days with no gaps**

Timezones

- Timestamps are **naive** when **no timezone attached**
- “18h” is a rush hour **only on New York’s timezone**
- Timezone here **is** New York’s, but **never explicitly stated: an assumption**
- If interpreted as UTC, the same numbers put the peak at **13h**

One timestamp, two timezones

```
t = pd.Timestamp("2025-01-15 18:00")           # naive
t.tz_localize("America/New_York").tz_convert("UTC")
# Timestamp('2025-01-15 23:00:00+0000', tz='UTC')
t.tz_localize("UTC").tz_convert("America/New_York")
# Timestamp('2025-01-15 13:00:00-0500', tz='America/New_York')
```

- `tz_localize` **declares** the timezone; the clock time stays 18:00
- `tz_convert` **moves** an aware timestamp to another zone
- Same “18:00”, two declarations → **five hours apart**
- Declare the zone **once, at dataset load**

What the data misses

Representativeness

- The rows describe **whatever/whoever the collection process reached**
- Our file: **yellow taxis, January 2025**. It is **not** New York transport in general (no Uber, no Lyft, no subway, etc.)
- “The average New York trip costs \$16.83” is **false in general**
- **The bias is in what was never collected (selection bias)**
- The conclusion is then **wrong about things that were never measured**

Pickups by borough

borough	pickups (our file)	residents (2020 census)
Manhattan	89.0 %	19.2 %
Queens	8.4 %	27.3 %
Brooklyn	1.9 %	31.1 %
Bronx	0.4 %	16.7 %
Staten Island	0.01 %	5.6 %

- 89 % of pickups in Manhattan, home to 19 % of New Yorkers
- Most of Queens' 8.4 % is the two airports (6.7 % of all pickups)
- Brooklyn and the Bronx: 48 % of residents, 2.3 % of pickups
- A model of "New York taxi demand" trained on this data learns Manhattan and airports trip behavior

Representativeness: the project data

- “Who would be missing from a conclusion drawn from this?”: one written sentence
- Project data: who uses the source? A GitHub dataset = people who use GitHub; a Hacker News dataset = a specific, narrow readership
- Sample size does not fix it: ten million yellow-taxi trips still contain zero bus passengers

Plotting data

matplotlib and seaborn

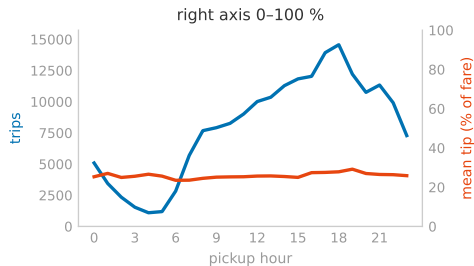
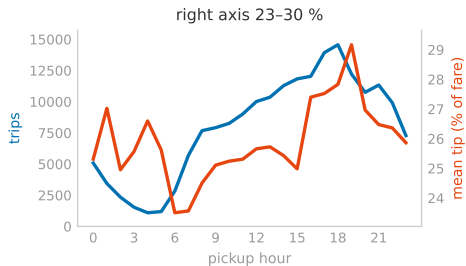
```
import matplotlib.pyplot as plt
import seaborn as sns
shown = df.loc[df["fare_amount"].between(0, 90), "fare_amount"]
titles = ["5 bins", "50 bins",
          "500 bins"]
fig, axes = plt.subplots(1, 3, figsize=(11, 2.6))
for ax, bins, title in zip(axes, (5, 50, 500), titles):
    sns.histplot(shown, bins=bins, ax=ax)
    ax.set_title(title)
```

- **matplotlib** draws; **seaborn** = a layer over it that knows DataFrames and computes the statistics → fewer lines of code
- Uses matplotlib's `fig, ax`

Misleading plots: the axis

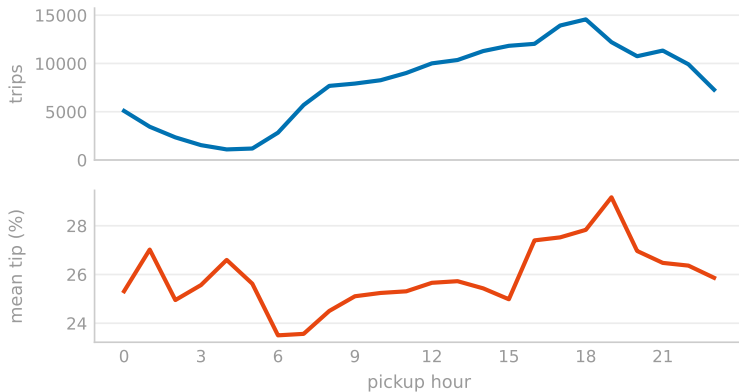
- **Two y-axes**: any two series can look correlated by rescaling one, and the reader cannot see the scaling → **use two charts**
- **Truncated y-axis** on a bar chart: a bar encodes **length**, a cut axis multiplies the difference → bars start at **zero**
- On a **line** chart truncation is ok (a line encodes **change**, and a zero baseline can flatten what is shown).
- **Averaging away the shape**: one bar of means in a skewed distribution.

Two y-axes (*)



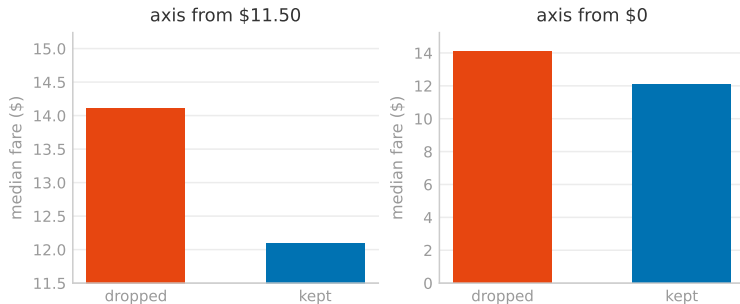
- Same data: trips and mean card tip per pickup hour ($r = 0.48$)
- Only the **right axis** changes: tips **follow the rush hour**, or tips **seem to not move**

Two y-axes: two charts (*)



- One axis each, **shared x**: both scales are visible

Truncated axis (*)

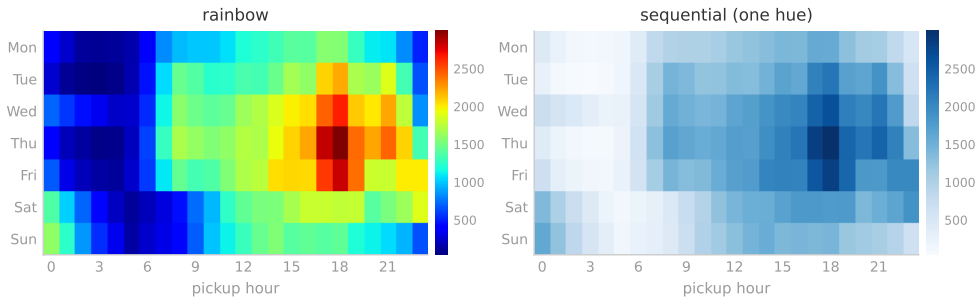


- The two medians of `dropna()`: \$14.11 dropped, \$12.10 kept
- Axis from \$11.50: first bar looks 4× taller. From \$0: 17% taller

Misleading plots: the colour

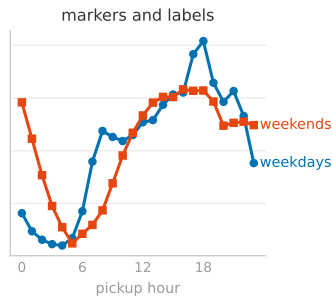
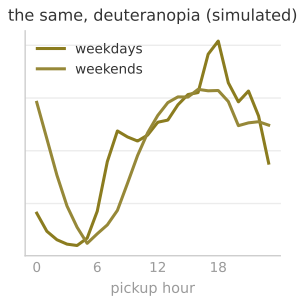
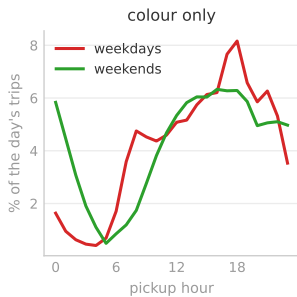
- **Rainbow for ordered values**: ordered data takes **one hue, light to dark** (a **sequential** palette). A rainbow can show boundaries that are not real, and hard to infer **order** in the colours
- **Colour alone** to tell series apart: roughly **1 in 12 men** cannot separate red from green → **label the lines or add markers**, colour as the second signal
- **Too many colours**: over five or six categories, nobody can match the legend to the line → put the tail into **“other”**, or use **small multiples** (one small chart per category)

Rainbow and sequential, on taxi trips (*)



- Same counts: trips per weekday and pickup hour
- Rainbow: **sharp bands** where the hue changes, but not actually in the data.
- Sequential: **darker = more**

Colour alone, and markers (*)



- Share of the day's trips by hour
- Red and green become one olive under deuteranopia. Markers and labels still separate the lines

Lab and next week

Today's lab

- **M1 was due last night**: acquisition log, the script, Parquet shapes
- **Three mandatory exercises**: **mechanism** of the missing rows, **impossible and ambiguous** masks, and a **findings** file
- Solved: the first commands, missingness map, the cost of `dropna()`, time patterns
- **Save findings.md**: it is used in the next lab

Before next week

- Run the **same EDA** on your project data
- If your source returns text, check its **encoding**
- Next lecture: **cleaning and transformation**, and **data leakage**

References

- Anscombe, *Graphs in Statistical Analysis*, The American Statistician 27(1), 1973.
<https://doi.org/10.1080/00031305.1973.10478966>
- Tukey, *Exploratory Data Analysis*, Addison-Wesley, 1977 — the seminal book on EDA
- Rubin, *Inference and Missing Data*, Biometrika 63(3), 1976 — defined MCAR / MAR / MNAR.
<https://doi.org/10.1093/biomet/63.3.581>
- NYC TLC trip-record data dictionary (the payment_type and RatecodeID codes):
<https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>
- Borough populations, 2020 US Census:
https://en.wikipedia.org/wiki/Boroughs_of_New_York_City
- seaborn: <https://seaborn.pydata.org> — matplotlib:
<https://matplotlib.org/stable/users/index.html>