PATTERN RECOGNITION AND MACHINE LEARNING

Lecture 02 – From Lunch Orders to Data Analysis

Chizhi Chris ZHANG

zhangchizhi@ciomp.ac.cn
Advanced Computing and Digital Technology Research Center
University of Chinese Academy of Sciences

Fall 2026

Which lunch queue would you choose?

Stall	People ahead	Your first guess
A	4	Shorter queue?
B	7	Faster kitchen?

The question

Can we judge waiting time from queue length alone?

Our evidence. A small teaching table of lunch orders; not a campus survey.

Meet the orders we will actually analyze

Order	Stall	Queue	Wait (min)
101	A	2	4
105	" A "	8	12
107	A	3	missing
108	B	6	10
108	B	6	10
109	A	4	-1

Raw file. 13 rows, 6 columns. This preview shows six selected rows.

One question, four things to produce

Stage	What we will produce
Environment	A Python program that runs
NumPy	A calculation we can check by hand
pandas	A cleaned order table and a stall summary
Exploration	An answer supported by plots and numbers

Our question

What can these orders tell us about waiting for lunch?

Python, NumPy, pandas: different jobs

Tool	Role	Example
Python	Language	Name a value; run a calculation
NumPy	Numerical arrays	Convert all waiting times to seconds
pandas	Labeled tables	Average waiting time by stall
Matplotlib	Plots	Draw a waiting-time histogram

Editor or notebook. The place you write code is not the Python interpreter.

Set up one environment and test its packages

```
python -m venv .venv
.\.venv\Scripts\python.exe -m pip install numpy pandas matplotlib
.\.venv\Scripts\python.exe -c "import numpy, pandas; print('Ready')"
```


Windows terminal, inside your course folder

Use the same Python path for installation and execution.

Success. The final command prints Ready.

Check the interpreter actually running your code

```
import sys
import numpy as np
import pandas as pd

print(sys.executable)
print(sys.version.split()[0])
print(np.__version__, pd.__version__)
```

The first line matters

For the course environment, the path should point into your course .venv folder.

Use this when. A package imports in the terminal but fails in the editor or notebook.

A few values are enough for a first calculation

```
wait_min = 6
print(wait_min * 60)                  # 360 seconds

waits = [4, 6, 3, 9]                  # four orders, minutes
mean_wait = sum(waits) / len(waits)
print(mean_wait)                      # 5.5 minutes
```

Read the calculation

22 total minutes divided by 4 orders gives 5.5 minutes per order.

A function returns a result you can use again

```
def average_wait(values):
    return sum(values) / len(values)

first_group = average_wait([2, 6])
second_group = average_wait([3, 6, 9])
print(first_group, second_group)       # 4.0 6.0
print(second_group - first_group)     # 2.0
```

Input, calculation, output

The function expects a nonempty list of known waiting times, in minutes.

Save, run, then change one order


```
waits = [4, 6, 3, 9]
print(f"Mean wait: {sum(waits) / len(waits):.1f} min")
# Mean wait: 5.5 min
```


Save as lunch.py. Run in its folder: `.\.venv\Scripts\python.exe lunch.py`

Try before running

Add a fifth wait of 13 minutes. Predict the new count, total, and mean.

A notebook remembers what you ran

Script

Runs the saved file from the top.

Notebook

Runs selected cells; earlier values stay in memory.

```
# Cell 1: run once
total = 22
# Cell 2: run twice
total = total + 4
print(total)       # 26 first; 30 next
```

To repeat the analysis

Restart the kernel and run all cells in order; select the course environment.

Start JupyterLab in the course environment

```
.\.venv\Scripts\python.exe -m pip install jupyterlab
.\.venv\Scripts\python.exe -m jupyterlab
```

Where	What to do
Terminal	Launch from the course folder
Browser	Create a Python notebook and run a cell
Notebook cell	Check sys.executable before importing packages

The browser is the interface

Keep the server terminal running while you use the notebook.

Read an error from the bottom up

```
import numppy
# ModuleNotFoundError:
# No module named 'numppy'
```


Message	First thing to inspect
ModuleNotFoundError	Spelling and selected interpreter
FileNotFoundError	The exact file path
NameError	Whether the name has been defined

A list repeats; an array calculates


```
import numpy as np

print([4, 6] * 2)
print(np.array([4, 6]) * 2)
# [4, 6, 4, 6]
# [ 8 12]
```

Different objects, different meaning

List multiplication repeats a sequence. Array multiplication is elementwise.

Read an array as rows, columns, and stored values

```
X = np.array([[2, 4], [4, 6],
              [1, 3], [5, 9]], dtype=float)
print(X.shape, X.ndim, X.dtype)
# (4, 2)  2  float64
```


Property	This example
Rows	4 orders
Columns	Queue (people), then wait (minutes)
Dimensions	2 axes: rows and columns
Data type	Floating-point values

Index a value; slice a region

```
print(X[0, 1])
print(X[:2, :])
print(X[:, 1])
# 4.0
# [[2. 4.]
#  [4. 6.]]
# [4. 6. 3. 9.]
```

Zero-based indexing. Row 0 is the first row. A slice stops before its end index.

Reorder whole observations, not each column separately

```
# Four selected orders: identifier, queue, wait in minutes
R = np.array([[101, 2, 4], [105, 8, 12],
              [108, 6, 10], [112, 7, 30]])
positions = np.argsort(R[:, 2])
print(R[positions])
# [[101   2   4]
#  [108   6  10]
#  [105   8  12]
#  [112   7  30]]
```

Do not sort columns independently

Order 112 must keep queue 7 and wait 30 together.

One condition selects the matching orders

```
wait = X[:, 1]
mask = wait > 5
print(mask)           # [False  True False  True]
print(wait[mask])     # [6. 9.]
print(X[mask])        # [[4. 6.]
                      #  [5. 9.]]
print(mask.sum(), mask.mean())    # 2  0.5
```

Same mask, same orders

Two of four orders waited more than five minutes.

axis=0 combines orders within each column

```
print(X.mean(axis=0))
# [3.  5.5]
```

Output position	Calculation	Meaning
0	(2 + 4 + 1 + 5) / 4	3 people ahead
1	(4 + 6 + 3 + 9) / 4	5.5 minutes

The row axis disappears

A (4, 2) array becomes a (2,) result.

axis=1 needs comparable columns


```
two_visits = np.array([[4, 6],
                       [3, 9]])
print(two_visits.mean(axis=1))
# [5. 6.]
```


Row	Column 0	Column 1
Student 1	Visit 1: 4 min	Visit 2: 6 min
Student 2	Visit 1: 3 min	Visit 2: 9 min

Different data on this page

Both columns are minutes. Do not average queue size and waiting time.

Broadcasting reuses values along a compatible axis

```
center = X.mean(axis=0)            # [3.  5.5]
print(X - center)
# [[-1. -1.5]
#  [ 1.  0.5]
#  [-2. -2.5]
#  [ 2.  3.5]]
```

Shapes compared from the right	Compatible?
(4, 2) and (2,)	Yes: reuse the two column means
(4, 2) and (4, 1)	Yes: expand the size-1 column axis
(4, 2) and (4,)	No: 2 and 4 do not match

Rule. Aligned dimensions must be equal, or one must be 1.

Elementwise multiplication is not a matrix product

```
A = np.array([[1, 2], [3, 4]])
v = np.array([10, 20])
print(A * v)                    # [[10 40]
                                #  [30 80]]
print(A @ v)                    # [ 50 110]
```


Operation	First row calculation	Result shape
A * v	[1 * 10, 2 * 20]	(2, 2)
A @ v	1 * 10 + 2 * 20	(2,)

Different questions

Multiplying entries keeps separate products; a matrix product also adds them.

Image shape is part of the information

One grayscale sample

Rows and columns locate a pixel; intensity is the value stored there.

```
pixels = np.zeros((28, 28))
flat = pixels.reshape(-1)
print(flat.shape)       # (784,)
restored = flat.reshape(28, 28)
```

Geometry example. The code uses zeros, not the pictured digit pixels.

Compare a shared slice with an independent copy

Shared slice

Changing b also changes a.

```
a = np.array([4., 6., 3., 9.])
b = a[:2]
b[0] = 99
print(a)
# [99. 6. 3. 9.]
```

Independent copy

Changing b leaves a unchanged.

```
a = np.array([4., 6., 3., 9.])
b = a[:2].copy()
b[0] = 99
print(a)
# [4. 6. 3. 9.]
```

Choose deliberately. Use a copy when a trial edit must not alter the original data.

Choose a data type that can store the answer


```
a = np.array([4, 6], dtype=int)
a[0] = 4.5
print(a)
# [4 6]

b = np.array([4, 6], dtype=float)
b[0] = 4.5
# b is [4.5 6. ]
```

Storage matters

An integer array cannot retain a fractional value assigned to an element.

Missing does not mean zero

```
a = np.array([4., 6., np.nan])
print(a.mean())
print(np.nanmean(a))
# nan
# 5.0
```

Calculation	What it says
mean	At least one value is unknown
nanmean	Mean of the two observed values

Denominator

5.0 is not a mean based on three known waiting times.

Array challenge: select, convert, summarize

```
wait = np.array([4., 6., 3., 9.])

# 1. Keep waits longer than 5 minutes.
# 2. Convert the selected waits to seconds.
# 3. Find their mean in seconds.
```

Before running

How many results remain? What unit will the final number use?

Array challenge: the answer is 450 seconds

```
selected = wait[wait > 5]
seconds = selected * 60
print(seconds)
print(seconds.mean())
# [360. 540.]
# 450.0
```

What arrays do not tell us

Which stall served each order? Which column name belongs to each number?

A DataFrame keeps the column names

order_id	stall	queue	wait_min
101	A	2	4
102	A	4	6
103	B	1	3

DataFrame

A two-dimensional labeled table; different columns can have different data types.

Series. Selecting one column usually gives a one-dimensional labeled Series.

Read the order file next to your script

```
from pathlib import Path
import pandas as pd

base = Path(__file__).resolve().parent
data_file = base / "orders_raw.csv"
raw = pd.read_csv(data_file)
print(raw.shape)                    # (13, 6)
print(raw.head(3).to_string(index=False))
```

One folder. Keep the supplied program and both data files together.

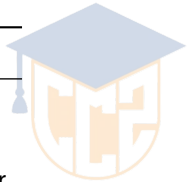
In a notebook

Use a chosen project folder instead of `__file__`.

Inspect the whole table structure with info

```
raw.info()
print(raw.isna().sum())
```

Check	What this raw file shows
Rows	13, before removing the copied row
Columns	6, before deriving the hour
wait_min nonmissing values	12: the stored -1 still counts as a number
Other columns	13 nonmissing values each



What info cannot decide

A stored value can be present and still be invalid for the task.

Define the rows and columns before cleaning

One row

One final order record. The order identifier should be unique.

Column	Meaning	A sensible operation
order_id	Identifier	Find a particular order
stall	Category	Compare groups
queue	Count of people	Calculate a mean
wait_min	Elapsed minutes	Summarize the distribution
ordered_at	Time, initially text	Parse and compare time slots

Do not confuse identity with quantity

An average order identifier has no useful meaning here.

Select by column name, row label, or position

```
small = raw[["order_id", "stall", "wait_min"]]
print(small.shape)                          # (13, 3)
wait_series = raw["wait_min"]               # one Series

by_id = raw.set_index("order_id")
print(by_id.loc[101, "wait_min"])           # 4.0
print(raw.iloc[0]["wait_min"])              # 4.0
```

Selection	Meaning
A list of column names	Keep a table with those columns
loc[101]	Find row label 101
iloc[0]	Find the first row position

Series arithmetic matches labels, not displayed positions


```
prices = pd.Series([12, 15], index=[101, 102])
discounts = pd.Series([3, 1], index=[102, 101])
paid = prices - discounts
print(paid)
# 101    11
# 102    12
```

Separate discount example

Order 101: 12 - 1. Order 102: 15 - 3.

Do not assume first minus first

The index links each price to the discount for the same order.

Clean invisible differences in category labels


```
clean = raw.copy()
clean["stall"] = clean["stall"].str.strip()
print(clean["stall"].unique().tolist())
# ['A', 'B']
```

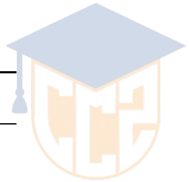

Before and after

`" A "` becomes `"A"`.

Working names. raw stays unchanged; clean is the table we are preparing.

Convert numeric text without hiding parsing failures

```
text_waits = pd.Series(["4", " 6 ", "unknown", "12 min"])
numeric_waits = pd.to_numeric(text_waits, errors="coerce")
print(text_waits[numeric_waits.isna()].tolist())
# ['unknown', '12 min']
```



Text	First conversion	Next decision
"4", " 6 "	4 and 6	Usable numeric values
"unknown"	Missing	Keep as unknown
"12 min"	Missing	Parse the unit only under a known rule

This is a separate import example

The main order file already stores its waiting-time column numerically.

Parse time before extracting the hour

```
clean["ordered_at"] = pd.to_datetime(
    clean["ordered_at"], errors="raise"
)
clean["hour"] = clean["ordered_at"].dt.hour
print(clean["ordered_at"].iloc[0])
print(clean["hour"].head(3).tolist())
# 2026-09-01 11:40:00
# [11, 11, 11]
```

What becomes possible?

Compare orders from different time slots instead of grouping every timestamp separately.

Compute elapsed minutes from complete timestamps

```
started = pd.Series(pd.to_datetime([
    "2026-09-01 11:40", "2026-09-01 23:58"]))
finished = pd.Series(pd.to_datetime([
    "2026-09-01 11:44", "2026-09-02 00:04"]))
elapsed = (finished - started).dt.total_seconds() / 60
print(elapsed.tolist())                    # [4.0, 6.0]
```


Separate two-receipt example

The second interval crosses midnight; complete dates preserve the six-minute duration.

A clock reading is not a duration

Do not subtract strings or only the minute-of-hour fields.

Inspect the copied order, then remove one copy

```
repeated = clean.loc[clean.duplicated(keep=False)]
print(repeated["order_id"].tolist())       # [108, 108]

clean = clean.drop_duplicates().copy()
print(len(raw), len(clean))               # 13 12
print(clean["order_id"].is_unique)         # True
```


Why removal is justified

The two 108 rows are identical copies of one final order record.

A different situation

Conflicting rows with the same identifier need investigation, not arbitrary deletion.

An empty cell and a missing-value code need one meaning

```
print(clean.loc[clean["wait_min"].isna(),
                "order_id"].tolist())          # [107]
print(clean.loc[clean["wait_min"] < 0,
                "order_id"].tolist())          # [109]
```


Record	Stored value	Meaning in this file
107	Empty	Waiting time not recorded
109	-1	Waiting time not recorded

Field definition

Wait is elapsed time from ordering to pickup; it cannot be negative.

Write back with one explicit loc operation


```
clean.loc[clean["wait_min"] < 0, "wait_min"] = np.nan
print(clean["wait_min"].isna().sum())
# 2
```


Read the assignment

For negative-wait rows, replace the waiting-time cell with missing.

Update the intended table

Use one loc assignment, not clean[condition]["wait_min"] = value.

Keep one full table and one observed-wait subset

```
valid = clean.dropna(subset=["wait_min"]).copy()
print(len(raw), len(clean), len(valid))
# 13 12 10
```

Name	Rows	What it contains
raw	13	Original records, including a copied row
clean	12	Distinct orders, including 2 unknown waits
valid	10	Orders with recorded waiting times

Different questions, different denominators

Total orders: 12. Observed waiting times: 10.

Count categories before comparing their outcomes

```
hour_counts = clean["hour"].value_counts().sort_index()
print(hour_counts)
# 11    3
# 12    9
```


Quantity	Question
Frequency of an hour	How many recorded orders belong to it?
Mean waiting time by hour	How long did observed orders in it wait?

Here we count all distinct orders

Three before noon and nine during the twelve o’clock hour; total 12.

Sort to inspect the longest waits

```
top = valid.sort_values("wait_min", ascending=False)
print(top[["order_id", "wait_min"]].head(3)
         .to_string(index=False))
```


Order	Wait (min)
112	30
110	18
106	15

Sorting. Changes order, not which rows belong to valid.

Combine row conditions with parentheses

```
chosen = valid.loc[
    (valid["stall"] == "B") & (valid["wait_min"] > 10),
    ["order_id", "wait_min"]
]
print(chosen.to_string(index=False))
```

order_id	wait_min
106	15
110	18
112	30

Group size and observed count are different

```
print(clean.groupby("stall")["wait_min"]
           .agg(["size", "count"]))
```

Stall	size: all orders	count: observed waits
A	6	4
B	6	6

Two missing waits

Both occur at stall A in this teaching sample.

Group the orders and name the summaries

```
summary = valid.groupby("stall").agg(
    observed_orders=("wait_min", "size"),
    mean_wait=("wait_min", "mean"),
    median_wait=("wait_min", "median")
)
```

Stall	Observed	Mean (min)	Median (min)
A	4	6.75	5.5
B	6	14.17	12.5

Check the mean. A: 27 / 4. B: 85 / 6. Summaries use only recorded waits.

Cross-tabulate the waiting-time threshold

```
valid["late"] = valid["wait_min"] > 10
print(pd.crosstab(valid["stall"], valid["late"]))
```


Stall	At most 10 min	More than 10 min
A	3	1
B	3	3

Definition. The 10-minute threshold is a classroom service target, not a learned rule.

Normalize a cross-table to answer a conditional question

```
rates = pd.crosstab(valid["stall"], valid["late"],
                    normalize="index") * 100
print(rates)
```

Stall	At most 10 min	More than 10 min
A	75 percent	25 percent
B	50 percent	50 percent

Read a row

Among the 4 recorded A waits, 1 exceeds 10 min: 25 percent.

A different denominator gives a different question

That same A order is 1 of all 10 recorded waits: 10 percent.

Merge stall information onto each order

```
stalls = pd.read_csv(base / "stalls.csv")
joined = clean.merge(stalls, on="stall", how="left",
                     validate="many_to_one")
print(len(joined))
# 12
```


Stall	Food
A	Noodles
B	Rice

Many orders, one stall record

Each order receives one food label; the number of orders stays 12.

A duplicate key can multiply rows

Orders
101: A
102: A

Stall record	Food
A	Noodles
A	Noodles

Without validation

2 orders $\times$ 2 matching stall records = 4 joined rows.

With many_to_one validation. pandas raises an error when the right table has a duplicate key.

A left join can preserve an unmatched order

Order	Stall	Matched food
201	A	Noodles
202	C	missing

```
two_orders = pd.DataFrame({"order_id": [201, 202],
                           "stall": ["A", "C"]})
result = two_orders.merge(stalls, on="stall", how="left",
                          validate="many_to_one", indicator=True)
```

Separate two-order example

The C order is retained; left_only identifies its missing match.

Concatenate new observations; merge extra attributes

```
day_one = pd.DataFrame({"order_id": [201, 202],
                        "wait_min": [4, 6]})
day_two = pd.DataFrame({"order_id": [203], "wait_min": [9]})
combined = pd.concat([day_one, day_two], ignore_index=True)
print(combined["order_id"].tolist())        # [201, 202, 203]
```


Operation	What it does in these examples
concat	Stack two days of observations into three rows
merge	Match a stall attribute to each existing order

Separate two-day example

Check column names, units, and identifier scope before stacking real files.

Save the cleaned result without a stray index column

```
clean.to_csv(base / "orders_clean.csv", index=False)
summary.to_csv(base / "stall_summary.csv", index=True)
```


Export	Why this index choice?
Order table	order_id is already a named column
Summary table	The index contains the stall labels

Keep the original. Export to a new file; do not overwrite orders_raw.csv.

Your turn: find the slower recorded orders at A

Question

At stall A, which recorded waits exceed 5 minutes? Report the order identifiers, count, and mean.

```
# Start with valid.
# Combine a stall condition and a waiting-time condition.
# Keep the identifiers so another person can check the result.
```

Boundary check. Does an order with a 5-minute wait belong?

Two A orders remain, with a mean wait of nine minutes

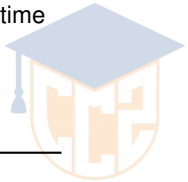

```
slow_a = valid.loc[
    (valid["stall"] == "A") & (valid["wait_min"] > 5)
]
print(slow_a["order_id"].tolist())
print(len(slow_a), slow_a["wait_min"].mean())
# [102, 105]
# 2 9.0
```

Check the meaning

6 and 12 minutes average to 9. This is not the mean for all A orders.

Choose a plot by the question

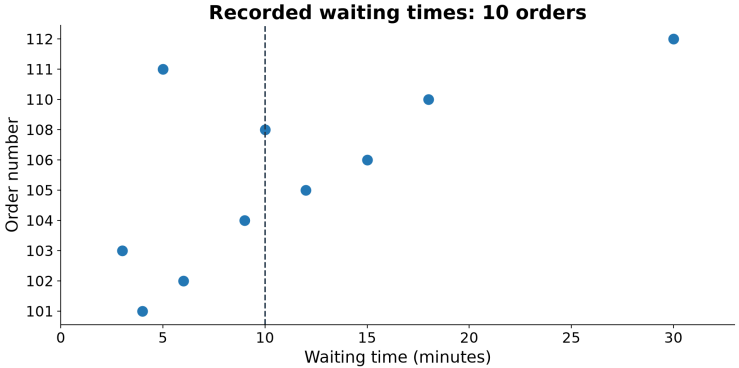
Question	Useful first view
How long do orders take?	Dots or a histogram of waiting time
Do stalls differ?	Waiting times grouped by stall
Does a longer queue mean a longer wait?	Queue versus wait scatter plot
What was not recorded?	Missing counts by stall



Exploratory data analysis

Use summaries and plots to understand the data before building a model.

Start with the ten observed orders



One dot per order. The vertical axis identifies an order. The horizontal axis is waiting time in minutes.

Threshold. The dashed line marks 10 minutes; four dots are strictly to its right.

Mean and median answer different questions

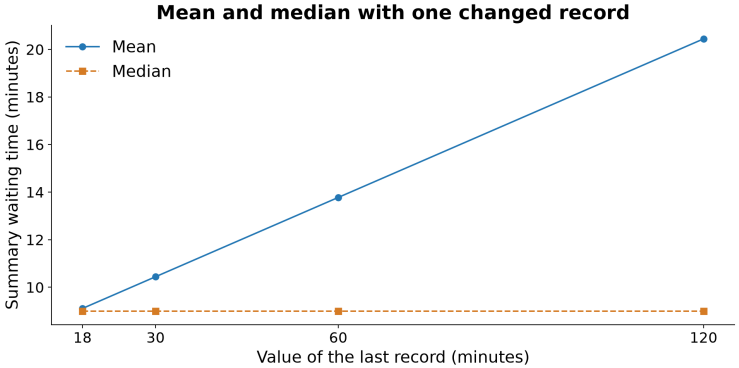
Sorted waits, minutes
3, 4, 5, 6, 9, 10, 12, 15, 18, 30

Summary	Calculation	Result
Mean	112 / 10	11.2 min
Median	(9 + 10) / 2	9.5 min

Why the difference?

The long wait at 30 minutes pulls the mean upward.

One extreme record can move the mean



Sensitivity experiment. Eight waits stay fixed: 3, 4, 5, 6, 9, 10, 12, 15. Only a ninth value changes.

What to notice. The mean rises with the changed value; the median stays at 9 minutes.

A histogram counts orders within intervals

Bins. Here the intervals are 5 minutes wide. The bars count orders, not average waiting time.

Read one bar. The interval [5, 10) contains waits of 5, 6, and 9 minutes: three orders.

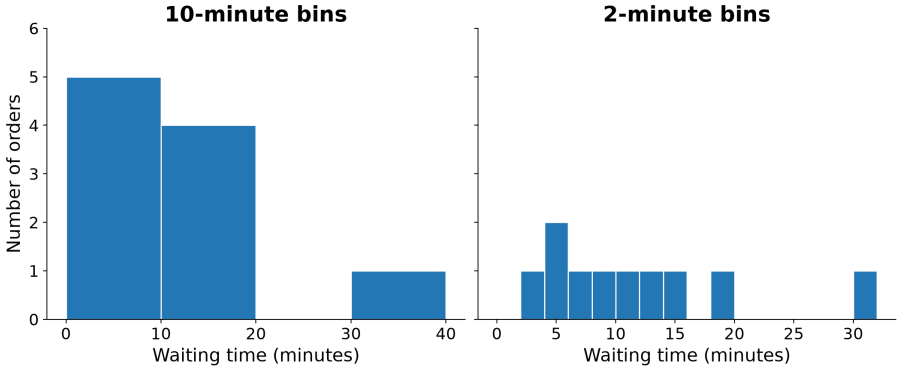
Make a plot from the exact column you analyzed

```
import matplotlib.pyplot as plt

plt.hist(valid["wait_min"], bins=range(0, 36, 5))
plt.xlabel("Waiting time (minutes)")
plt.ylabel("Number of orders")
plt.title("Recorded waits: 10 orders")
plt.tight_layout()
plt.show()
```

Check the input. Use the cleaned observed values, not a screenshot or an unrelated example array.

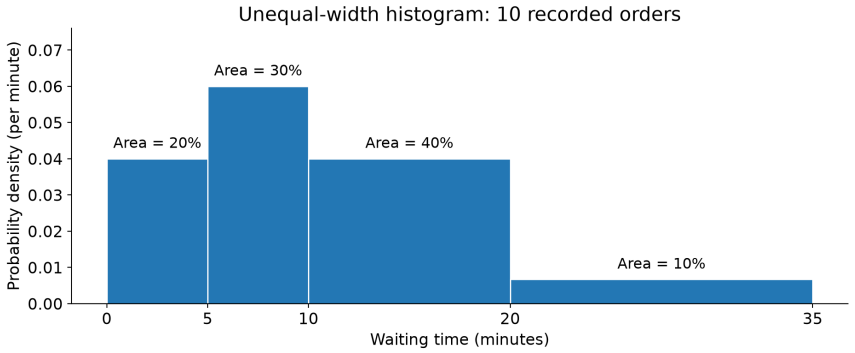
Bin width changes the visible detail



Same ten orders. Left: 10-minute bins. Right: 2-minute bins. Only the binning changes.

Read cautiously. Wide bins hide detail; narrow bins make a small sample look fragmented.

With unequal bin widths, read probability from area



Our ten observed waits. Intervals: 0-5, 5-10, 10-20, and 20-35 minutes.

Density times width. The 10-20 interval has area 0.04 x 10 = 0.40, containing 4 of 10 orders.

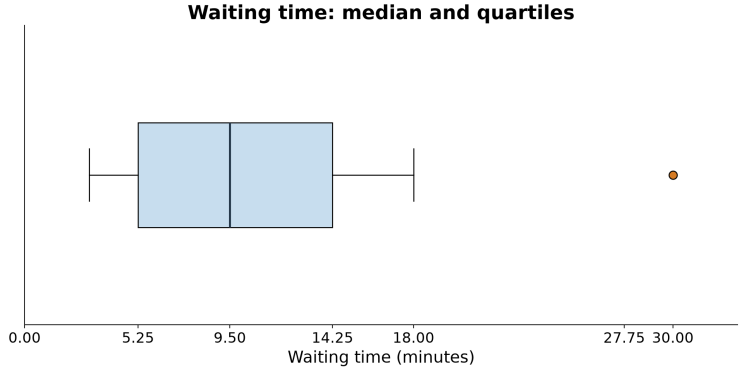
Quartiles locate the middle half of the data

```
q = valid["wait_min"].quantile([0.25, 0.5, 0.75])
print(q.to_list())
# [5.25, 9.5, 14.25]
```


Position	Waiting time
25th percentile	5.25 min
50th percentile	9.50 min
75th percentile	14.25 min

Convention. These values use pandas' default linear interpolation.

A boxplot shows median, spread, and flagged points



Our ten recorded waits. Box: 5.25 to 14.25 min. Median: 9.5 min. Whiskers reach 3 and 18 min.

The point at 30. It lies beyond the 1.5-interquartile-range fence and is plotted separately.

An outlier flag is a question, not a deletion order

Quantity	Value
Interquartile range	14.25 - 5.25 = 9.00 min
Upper fence	14.25 + 1.5 $\times$ 9 = 27.75 min
Order 112	30 min: flagged

Case information

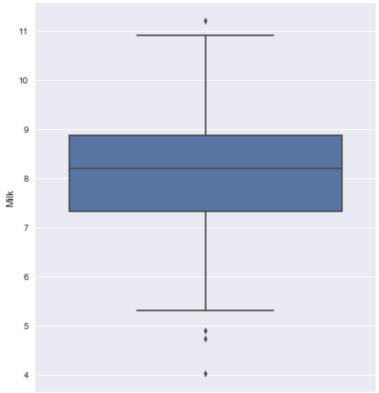
In this invented case, a delayed batch explains order 112. Keep the valid 30-minute wait.

Read the same boxplot grammar in another example

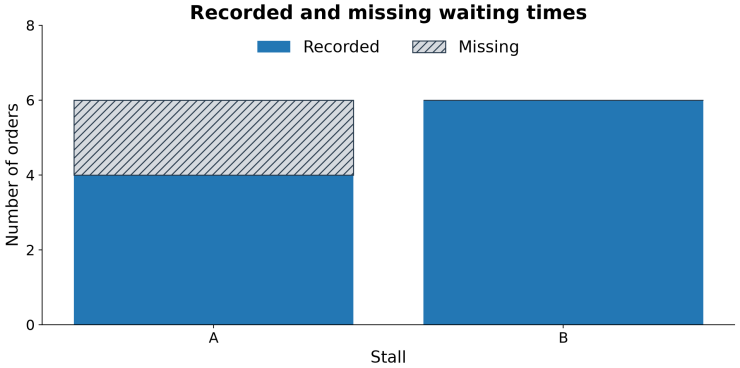
What transfers

The line inside the box is the median; isolated points are flagged observations.

A different variable. The vertical axis is labeled Milk. Its unit is not provided in this excerpt.



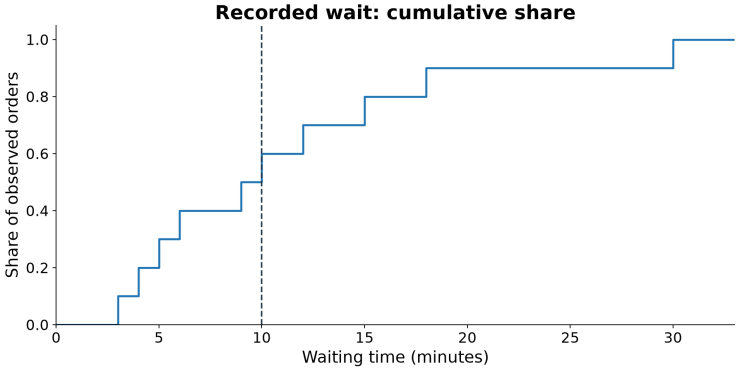
Missingness is concentrated at stall A



All twelve distinct orders. A: 4 recorded, 2 missing. B: 6 recorded, none missing.

What we do not know. Missing waiting times could be shorter or longer than the observed ones.

Ask how many customers meet a waiting-time target



Cumulative share among ten recorded orders. At 10 minutes, the cumulative share is 0.6: six orders waited at most 10 minutes.

Complement. Four of ten recorded orders waited more than 10 minutes: 40 percent.

The missing waits create uncertainty in the full rate

Assumption about 2 missing waits	Late orders / all orders	Rate
Neither exceeds 10 min	4 / 12	33.3 percent
Both exceed 10 min	6 / 12	50.0 percent

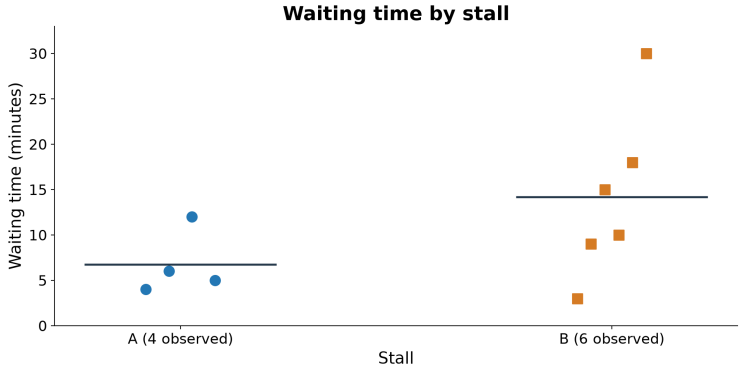
Observed rate

4 / 10 = 40 percent among recorded waits.

Do not mix denominators

The full 12-order late rate is not identified from the available waiting times.

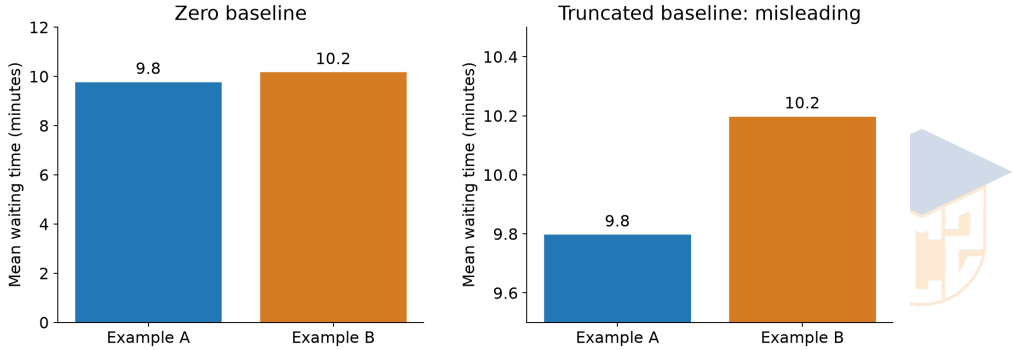
Compare the stalls using both means and individual waits



Recorded waits. A: 4 orders, mean 6.75 min. B: 6 orders, mean 14.17 min.

Read both parts. Points show individual waits; dark horizontal lines show means. The two groups overlap.

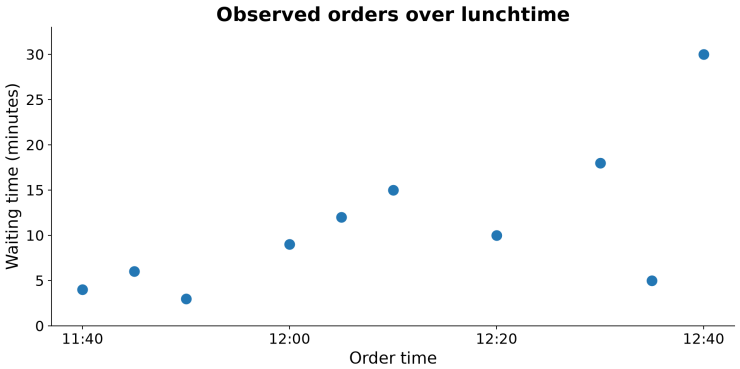
Check the baseline before judging a bar-length difference



Separate scale example. Both panels use the same means: 9.8 and 10.2 minutes.

Right panel is intentionally misleading. Its vertical axis starts at 9.5. The numerical difference is only 0.4 minutes.

Preserve time order when the question involves time



Observed orders, one lunch period. Horizontal: order time. Vertical: waiting time. Points are not connected because different orders are distinct observations.

Limited window. One lunchtime cannot establish a daily pattern or a long-term trend.

Compare the same stalls within quiet and busy periods

Period	A count	A mean	B count	B mean
Quiet	10	2 min	65	3 min
Busy	90	10 min	35	11 min

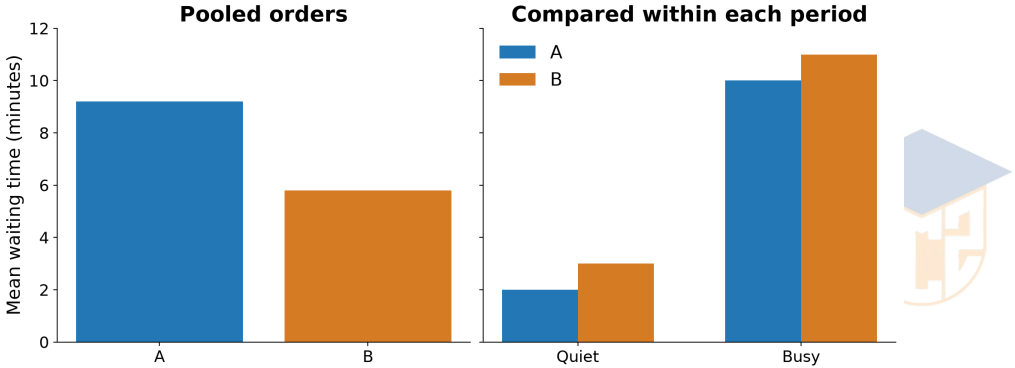
Within each period

A is one minute faster in both periods.

Before pooling

A has mostly busy-hour orders; B has mostly quiet-hour orders.

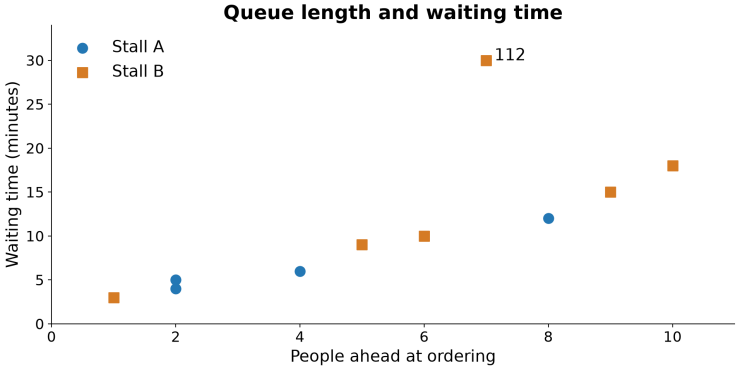
Pooling can reverse a comparison



Same separate example. Pooled means: A = 9.2 min; B = 5.8 min. Within both periods, A is faster.

Different mixture. The reversal comes from different proportions of quiet and busy orders.

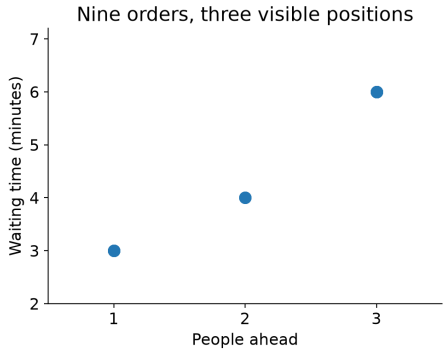
Does a longer queue come with a longer wait?

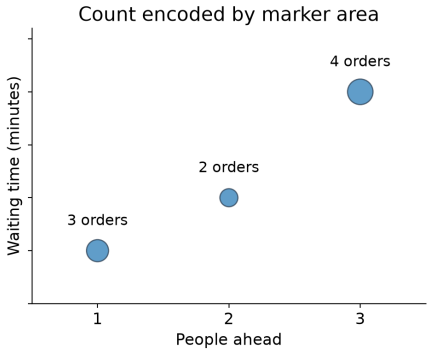


Back to our ten recorded orders. Horizontal: people ahead at ordering. Vertical: waiting time in minutes.

Look for both pattern and exceptions. The overall association is positive; order 112 has an unusually long wait for its queue size.

Several observations can occupy the same scatter position





Separate nine-order example. Three orders at (1, 3), two at (2, 4), and four at (3, 6).

Right panel. Marker area represents the count; labels show exact multiplicities.

Correlation summarizes linear association

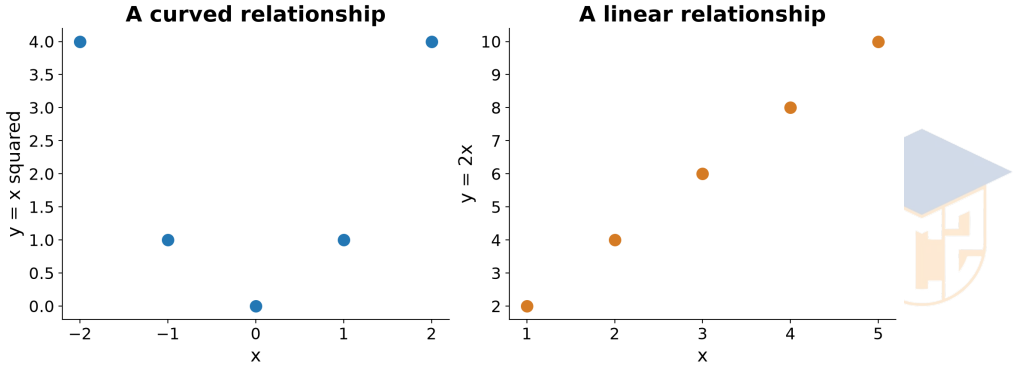
```
r = valid["queue"].corr(valid["wait_min"])
print(round(r, 3))
# 0.718
```

Value	Linear association
Near +1	Strong positive
Near 0	Weak linear association
Near -1	Strong negative

What it does not prove

Correlation does not establish a causal effect or rule out a curved relationship.

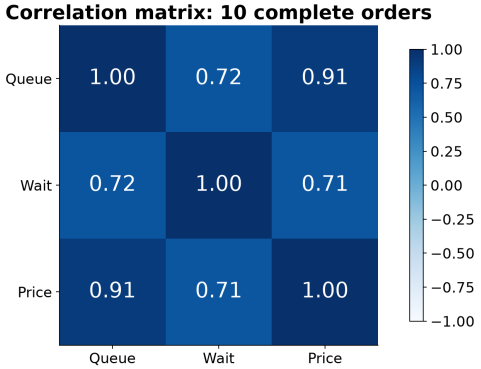
Zero correlation can hide a clear curve



Left panel. x = -2, -1, 0, 1, 2 and y = x squared. Pearson correlation is 0.

Different question. No linear association is not the same as no relationship.

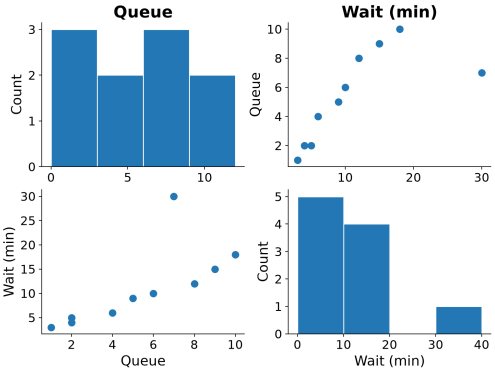
Use a correlation matrix as a compact overview



Ten complete orders. Queue–wait: 0.72; queue–price: 0.91; wait–price: 0.71.

Read one cell, then its scatter plot. The diagonal is 1 because each variable is correlated with itself.

A small pair plot connects distributions and relationships



Two variables, four panels. Diagonal panels summarize one variable. Off-diagonal panels show the pair in opposite axis orders.

Lower-left panel. Queue is horizontal; waiting time is vertical, as in our earlier scatter plot.

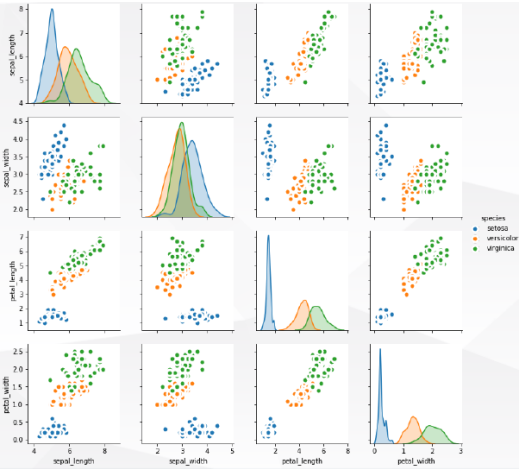
Read the measurements before reading a larger pair plot

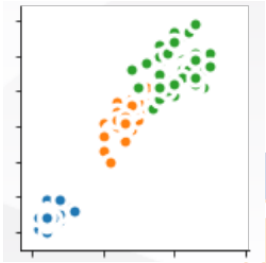
	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Iris flower measurements. Each column is a measurement; count is the number of nonmissing observations.

Read two entries. Sepal length: count = 150; median = 5.8. This is not our lunch table.

Read a larger pair plot without getting lost





Enlarged: row 3, column 4. Horizontal: petal width. Vertical: petal length.

Compare the colored groups

Blue is relatively separated; orange and green overlap.

Separate what is known now from what is known later

At ordering time	Only after completion
Stall	Actual waiting time
Queue length	Pickup time
Order time	Whether the wait exceeded 10 min
Price, if already quoted	A complaint filed after pickup

Prediction setup

Do not use future outcomes as inputs to predict the waiting time.

Estimate preprocessing rules from training data

Training queue lengths	Later evaluation queue lengths
4, 6, missing	30, missing

```
training_median = pd.Series([4., 6.]).median()
# 5.0: reuse this value for the queue-length input column
```

If input imputation is chosen

Learn the fill value from training data; apply the same learned rule to later inputs.

Separate input-feature example

Do not invent missing target waiting times to score predictions.

Who is represented by the collected orders?

Collection plan	Who is likely to be missed?
Ask only your friends	People with different schedules or food choices
Invite voluntary online feedback	Customers who choose not to respond
Sample orders across chosen time slots	People outside the sampled days and hours

State the intended population

All lunch customers, not only the people who answered a survey.

Improve the design

Use the same selection rule at both stalls and follow selected orders to completion.

Design a better recording sheet

Record	Reason
Order identifier	Detect copied records
Stall and menu item	Compare similar services
Order time and pickup time	Compute waiting time consistently
Queue at ordering	Capture the information available before prediction
Recording status	Distinguish a measured value from a missing one

Sampling rule

Use the same selection rule at both stalls; do not record only memorable orders.

Final task: write a short answer for a hungry student

The student asks

Should I always choose A because its average wait is lower?

Your answer must include
One numerical comparison with sample sizes
One important limitation visible in the data
One useful next data-collection step

Avoid. A long list of software names or an unsupported claim about efficiency.

Answer the lunch question with evidence, not a guarantee

Observed comparison

A: 6.75 min from 4 recorded waits; B: 14.17 min from 6 recorded waits.

Limitation. Two A waits are missing, and the observed orders were not collected under matched conditions.

Next step

Record both stalls in the same time slots across several days, including every selected order.