

Python, Arrays and Data Exploration

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Pattern Recognition and Machine Learning

Lecture 02 Python, Arrays and Data Exploration



Chizhi Chris ZHANG

zhangchizhi@ciomp.ac.cn

Advanced Computing and Digital Technology Research Center

University of Chinese Academy of Sciences

Autumn 2026

Lecture plan and time allocation

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Topic	Minutes
Arrays and sample identity	45
Data tables and cleaning	45
Exploratory data analysis	45

One lunch-order table

Keep row identities; clean the records;
explain what the summaries support.

The numerical steps are shown on the
slides.



The lunch question starts with a concrete observation table

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Every calculation must preserve which order each row describes and the meaning of each column. The same lunch question connects arrays, cleaning and exploration.

One row should represent one order. The raw CSV has 13 rows and 6 columns.

Order ID	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



One environment makes installation and execution consistent

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

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

Work in your course directory.
The first command creates a local environment. The next commands use the Python inside it.



Variables and functions express a calculation with reusable meaning

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
waits = [4, 6, 3, 9]           # minutes
def average_wait(values):
    return sum(values) / len(values)
mean_min = average_wait(waits)
print(mean_min, mean_min * 60) # 5.5, 330.0
waits.append(13)
print(average_wait(waits))     # 7.0
```

$$4 + 6 + 3 + 9 = 22$$

First total $4 + 6 + 3 + 9 = 22$ and divide by 4. After adding 13, total 35 and divide by 5.



A script runs in order, while a notebook keeps state between cells

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
# Cell 1
total = 22
# Cell 2: run twice without rerunning Cell 1
total = total + 4
print(total) # first: 26; second: 30
```

$22 + 4 = 26$ $26 + 4 = 30$

A script starts at the top. A notebook retains state between cell runs: $22 + 4 = 26$, then $26 + 4 = 30$.



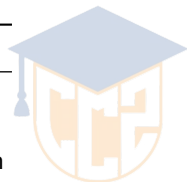
An error message identifies what the program could not do

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

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

Read the final line for the error type. Then locate the line of your own code that caused it. Compare what that operation requires with the object supplied.

Error type	First checks
Module not found	Module spelling and active interpreter
File not found	Path and file existence
Undefined name	Check spelling and whether the assignment was run



Array arithmetic depends on the object type

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
import numpy as np
print([4, 6] * 2)                # [4, 6, 4, 6]
wait = np.array([4, 6, 3, 9], dtype=float)
print(wait * 60)                 # [240. 360. 180. 540.]
print(wait.mean())              # 5.5
```

A list repeats its contents under multiplication by an integer. A numerical array multiplies each stored value.



Array shape and indexing keep each observation identifiable

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
X = np.array([[2, 4], [4, 6], [1, 3], [5, 9]], dtype=float)
print(X.shape, X.ndim)    # (4, 2), 2
print(X[0, 1])            # 4.0
print(X[:2, :])           # [[2., 4.], [4., 6.]]
print(X[:, 1])            # [4., 6., 3., 9.], shape (4,)
print(X[:, 1:2].shape)    # (4, 1)
```

Queue length and wait in minutes. Rows represent four orders. Python indexes from zero and excludes a slice's stop position.



Boolean selection and sorting must preserve complete rows

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
wait = X[:, 1]
mask = wait > 5
print(mask)           # [False, True, False, True]
print(X[mask])        # [[4., 6.], [5., 9.]]
positions = np.argsort(wait)
print(X[positions])   # complete rows ordered by wait
print(mask.sum(), mask.mean()) # 2, 0.5
```

Select the entire row using the condition on its waiting time. Sorting positions similarly reorder entire rows.



A reduction removes an axis, but units still determine meaning

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
print(X.mean(axis=0))           # [3.0, 5.5]
two_visits = np.array([[4, 6], [3, 9]])
print(two_visits.mean(axis=1)) # [5.0, 6.0]
```

Averaging queue length with waiting minutes returns a number but mixes incompatible units. The two visits have the same unit, so their row average is interpretable.



Broadcasting reuses values only along compatible dimensions

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
center = X.mean(axis=0) # [3., 5.5], shape (2,)
print(X - center)       # [[-1., -1.5], [1., .5],
                        #  [-2., -2.5], [2., 3.5]]
per_row = np.array([1, 2, 3, 4]).reshape(4, 1)
print(per_row.shape)    # (4, 1), shape demonstration
```

Compare dimensions from the right.
Corresponding sizes must be equal
or one must be 1. A missing
leading dimension acts like 1.

Shapes	Compatible?	Result
(4, 2) versus (2,)	Yes	(4, 2)
(4, 2) versus (4, 1)	Yes	(4, 2)
(4, 2) versus (4,)	No	Last dimensions 2 and 4 disagree



Elementwise products and matrix products calculate different answers

Elementwise multiplication

$$C = \begin{bmatrix} 2 & 4 \\ 4 & 6 \end{bmatrix}, \quad v = (1, 2).$$

$$C * v = \begin{bmatrix} 2(1) & 4(2) \\ 4(1) & 6(2) \end{bmatrix} = \begin{bmatrix} 2 & 8 \\ 4 & 12 \end{bmatrix}.$$

The second column is doubled. There is no sum across columns.

Matrix-vector multiplication

$$C@v = \begin{bmatrix} 2(1) + 4(2) \\ 4(1) + 6(2) \end{bmatrix} = \begin{bmatrix} 10 \\ 16 \end{bmatrix}.$$

One weighted sum is returned for each row.

```
C = np.array([[2, 4], [4, 6]])  
v = np.array([1, 2])  
print(C * v) # [[2, 8], [4, 12]]  
print(C @ v) # [10, 16]
```

A matrix product predicts one score for each sample

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
features = np.array([[2., 1.], [4., 3.], [1., 2.]])  
w = np.array([1., -0.5])  
b = 2.  
scores = features @ w + b  
print(scores) # [3.5, 4.5, 2.0]
```

$$s_1 = 2(1) + 1(-0.5) + 2 = 3.5.$$

Rows are three observations
and columns are two
comparable input features in a
separate calculation.



Reshaping changes organization, while a view may share stored values

```
pixels = np.zeros((28, 28))
flat = pixels.reshape(-1)           # shape (784,)
restored = flat.reshape(28, 28)    # same number of values
a = np.array([4., 6., 3., 9.])
b = a[:2]                          # a basic slice shares data
b[0] = 99
print(a)                           # [99., 6., 3., 9.]
safe = a[:2].copy()                # independent array
```

Check. Changing a shared slice can change the original data. Make an explicit copy when a separate experiment should leave the source values intact.

Reshaping preserves element count and uses a specified order. It does not learn image features or create additional information.



Data types and missing values affect a numerical result

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
wait = np.array([4., 6., np.nan, 9.])  
print(np.isnan(wait)) # [False, False, True, False]  
print(wait.mean())   # nan  
print(np.nanmean(wait)) # 19/3 = 6.333...
```

`np.nan` represents a missing floating-point value. It is not a zero-minute wait.



A DataFrame preserves column names and the observation unit

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Array operations assume that the values and their meanings are already consistent.

One row is one recorded order. The columns are ID, stall, order time, queue length, waiting minutes and price.

```
from pathlib import Path
import pandas as pd
data_dir = Path(__file__).resolve().parent # script location
orders = pd.read_csv(data_dir / "orders_raw.csv")
print(orders.shape) # (13, 6)
print(orders.head())
orders.info()
```

Column selection, row labels and positions are different selectors

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
orders["wait_min"]           # Series
orders[["queue", "wait_min"]] # DataFrame
orders.loc[0:2, ["stall", "wait_min"]] # labels 0,1,2
orders.iloc[:2, 1:3]         # positions 0,1
```

After filtering or changing the index, read the selector according to its definition.



Pandas arithmetic aligns labels rather than displayed positions

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
a = pd.Series([4, 6], index=[101, 102])
b = pd.Series([20, 10], index=[102, 101])
print(a + b)    # ID 101: 14; ID 102: 26
orders["order_id"].duplicated().sum() # 1 repeated ID
```

For ID 101, pandas adds 4 to 10 even though those entries occupy different displayed positions. The index determines the match.



Category normalization and numeric parsing make hidden differences visible

```
orders["stall"] = orders["stall"].str.strip()
raw_wait = orders["wait_min"]
parsed = pd.to_numeric(raw_wait, errors="coerce")
parse_failed = raw_wait.notna() & parsed.isna()
print(orders["stall"].value_counts())
print(orders.loc[parse_failed, ["order_id", "wait_min"]])
orders["wait_min"] = parsed
```

Whitespace can split one category into several labels. Unparseable text can prevent numerical calculation.



Elapsed time requires the complete timestamps

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

A timestamp combines a date and time. Its meaning is richer than a text string or a standalone hour.

```
orders["ordered_at"] = pd.to_datetime(
    orders["ordered_at"], errors="coerce")
print(orders["ordered_at"].isna().sum()) # inspect failures
orders["hour"] = orders["ordered_at"].dt.hour
print(orders[["order_id", "ordered_at", "hour"]].head())
```

Subtracting 58 from 6 gives -52 , which is not the elapsed time. The dates show that pickup occurred the next day.

```
ordered = pd.Timestamp("2026-09-01 23:58")
picked_up = pd.Timestamp("2026-09-02 00:06")
wait_min = (picked_up - ordered).total_seconds() / 60
print(wait_min) # 8.0
```

Duplicate removal depends on what a row represents

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
copied = orders.duplicated(keep=False)
print(orders.loc[copied])           # both copies of order 108
orders = orders.drop_duplicates().copy()
print(len(orders))                  # 12 distinct orders
print(orders["order_id"].nunique()) # 12
```

Here order 108 was copied as an identical full row. Removing one copy restores one row per order.



A missing wait changes which orders a mean describes

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The -1 marker and one blank become two missing waits. The known waits sum to 112 minutes, so their mean is $112/10 = 11.2$.

```
orders.loc[orders["wait_min"] == -1, "wait_min"] = np
        .nan
print(orders["wait_min"].isna().sum()) # 2 missing
print(orders["wait_min"].count())     # 10 observed
print(orders["wait_min"].mean())      # 11.2
```

$$112/10 = 11.2$$

Condition to check

A zero-minute wait is a possible outcome.
Replacing an unknown with zero or dividing by all 12 orders adds an assumption about the two unobserved outcomes.

A missing waiting time is an unknown outcome. An invalid negative wait is also not an observed nonnegative wait.

$$\bar{w}_{\text{observed}} = \frac{\sum_{i: w_i \text{ observed and valid}} w_i}{\#\{i : w_i \text{ observed and valid}\}}.$$



One full table and one observed subset support different questions

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
orders.loc[orders["wait_min"] < 0, "wait_min"] = np.nan
valid = orders.loc[orders["wait_min"].notna()].copy()
print(len(orders), len(valid))      # 12, 10
print(orders.groupby("stall")["wait_min"].size())
print(valid.groupby("stall")["wait_min"].size())
```

Check. Use all orders to describe missingness; use observed waits for their mean. Dropping rows from the only table hides which stall lacks outcomes.

orders retains all orders and missingness. valid supports calculations requiring observed waits. loc assigns to the intended rows; copy creates a separate subset.

Combined conditions answer a precise row-level question

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
mask = (valid["stall"] == "A") & (valid["wait_min"] > 5)
slow_a = valid.loc[mask, ["order_id", "queue", "wait_min"]]
print(slow_a)
# 102: queue 4, wait 6; 105: queue 8, wait 12
print(slow_a["wait_min"].mean()) # 9.0
```

Among recorded waits, select orders at stall A that took strictly more than 5 minutes.



Group size and observed count report different evidence

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
summary = orders.groupby("stall").agg(  
    all_orders=("order_id", "size"),  
    observed=("wait_min", "count"),  
    mean_wait=("wait_min", "mean"),  
    median_wait=("wait_min", "median"))  
print(summary)
```

Stall	All	Observed	Mean / min	Median / min
A	6	4	6.75	5.50
B	6	6	14.17	12.50

size counts rows, including missing waits. count counts nonmissing values in the selected column. Mean and median use those observed values.



The two stall means can be checked directly from their orders

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The overall mean weights the stall means by their observed counts: $(4 \cdot 6.75 + 6 \cdot 14.1667)/10$. The unweighted average of the two means answers a different question.

Stall	Recorded waits / min	Mean
A	4, 6, 12, 5	$27/4 = 6.75$
B	3, 9, 15, 10, 18, 30	$85/6 \approx 14.17$

$$\text{Overall observed mean} = \frac{27 + 85}{4 + 6} = 11.2 \text{ Minutes .}$$



Cross-tables distinguish counts from conditional proportions

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
late = valid["wait_min"] > 10
counts = pd.crosstab(valid["stall"], late)
rates = pd.crosstab(valid["stall"], late, normalize="index")
```

Check. Column normalization instead asks how each threshold group divides across stalls. Two missing A waits do not enter either observed-wait table.

Stall	≤ 10 Minutes	> 10 Minutes	Observed	Late-order fraction
A	3	1	4	25%
B	3	3	6	50%

Row normalization divides by the observed count at each stall. It estimates the share exceeding 10 minutes among that stall's recorded waits.



A merge joins attributes by a key and can multiply observations

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
stalls = pd.read_csv(data_dir / "stalls.csv")
enriched = orders.merge(stalls, on="stall", how="left",
                        validate="many_to_one", indicator=True)
print(len(enriched))      # 12 when every right-side key is unique
print(enriched["_merge"].value_counts())
```

Each order gets the food attribute of its stall. Many orders may match one stall record.



Concatenation adds observations, while export preserves the clean schema

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
# next_day has the same columns and row meaning
combined = pd.concat([orders, next_day], ignore_index=True)
orders.to_csv(data_dir / "orders_clean.csv", index=False)
summary.to_csv(data_dir / "stall_summary.csv")
```

Concatenating days can create repeated order IDs if numbering restarts each day. A date-plus-ID key may be necessary. The displayed index and the real order identifier have different roles.

Worked review: explain each change from raw rows to valid waits

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

For each operation, state which observations stay, what changes, and why the calculation's denominator follows from that choice.

Stage	Expected result and reason
Raw file	13 rows, including one copied order
Remove exact copy	12 distinct orders
Normalize missing code	Two unknown waits: orders 107 and 109
Select observed waits	10 orders used for wait summaries
Select stall A with wait > 5	Orders 102 and 105; mean $(6 + 12)/2 = 9$ Minutes



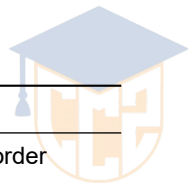
A mean and a median describe different aspects of the same waits

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The cleaned observed waits can be listed and checked.

The 30-minute wait strongly changes the sum. The median depends on the middle positions.

3, 4, 5, 6, 9, 10, 12, 15, 18, 30 minutes, sorted .



Summary	Computation	Question answered
Mean	$112/10 = 11.2$ Minutes	Waiting burden per recorded order
Median	$(9 + 10)/2 = 9.5$ Minutes	Middle position
Range	$30 - 3 = 27$ Minutes	Longest minus shortest
> 10 Percentage	$4/10 = 40\%$	Fraction exceeding a target

One long wait changes the mean while the median stays fixed

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Ten recorded waits have total 112. Hold nine values fixed with sum 82 and change only the largest value $v \geq 18$. The mean becomes $(82 + v)/10$, while the median stays 9.5.

$$\text{Mean} = (82 + v)/10, \quad \text{Median} = (9 + 10)/2 = 9.5.$$

The longest wait v min/minute	Mean / min	Median / min
18	10.0	9.5
30	11.2	9.5
60	14.2	9.5
120	20.2	9.5



Worked calculation

Increasing the longest wait by 10 minutes increases the mean by one minute. The middle positions and median 9.5 stay fixed. Investigate a genuine long wait rather than deleting it to improve the picture.

Histogram counts depend on explicit interval boundaries

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

```
import matplotlib.pyplot as plt
plt.hist(valid["wait_min"], bins=range(0, 36, 5))
plt.xlabel("Waiting time (minutes)")
plt.ylabel("Number of recorded orders")
plt.tight_layout()
```

The interval $[5, 10)$ contains 5, 6 and 9 minutes. A wait of 10 belongs to the next bin. Counts sum to the ten recorded orders.

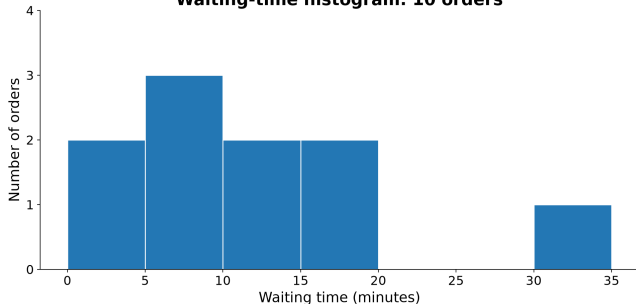
Interval	$[0, 5)$	$[5, 10)$	$[10, 15)$	$[15, 20)$	$[20, 25)$	$[25, 30)$	$[30, 35]$
count	2	3	2	2	0	0	1



Count orders inside each interval

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Waiting-time histogram: 10 orders



The horizontal axis shows waiting minutes and the vertical axis counts orders. A taller bar identifies a more common interval, rather than one exact wait for everyone.



Density uses bar area when histogram widths differ

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Key idea

Let a bin contain fraction p of observations and have width h . Its rectangle must have area p :
 $\text{height} \times h = p$.

$$\text{height} \times h = p$$

$$\text{density height} = p/h.$$

Interval	Share	width	height
[0, 5)	.20	5	$.20/5 = .04$
[10, 20)	.40	10	$.40/10 = .04$

For the 10–20 minute bin, height 0.04 times width 10 gives 0.40, representing four of ten observations. Areas sum to one.

$$\text{Rule } j \text{ cell density} = \frac{\text{Bin count}}{n \times \text{Bin width}}.$$

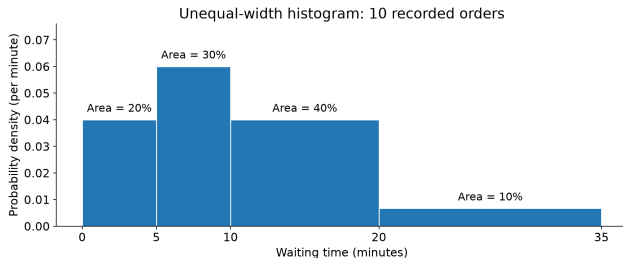
Bin	Count	Width	Density	Share
[0, 5)	2	5	0.04	0.20
[5, 10)	3	5	0.06	0.30
[10, 20)	4	10	0.04	0.40
[20, 35]	1	15	0.0067	0.10

Check. The first and third bins have equal density heights but contain different observation shares because their widths differ. A density has an inverse-unit scale.

Unequal bins require density, not raw height

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The 0–5 bin has height .04 and area .20. The 10–20 bin has the same height but area .40. With unequal widths, area represents the fraction of orders.

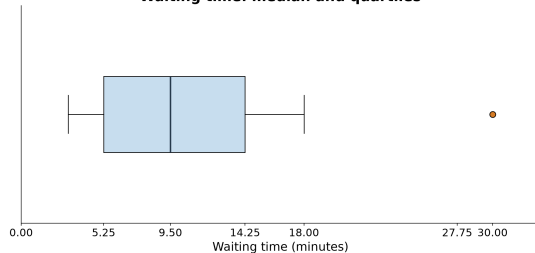


Quartiles and the interquartile range locate the middle half

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The box spans the middle half. The internal line marks the median.

Waiting time: median and quartiles



Quantity	Minutes
Q_1	5.25
Median Q_2	9.50
Q_3	14.25
$IQR = Q_3 - Q_1$	9.00
Upper limit $Q_3 + 1.5IQR$	27.75

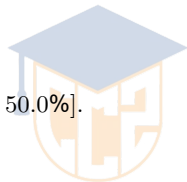
Missing outcomes create a range of possible full-order rates

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Among the two missing waits, zero, one or two could exceed 10 minutes. Without their outcomes or further assumptions, the full-order late rate is not determined.

Stall	All orders	Recorded waits	Missing waits
A	6	4	2
B	6	6	0

Observed late rate = $4/10 = 40\%$, full late rate $\in [4/12, 6/12] = [33.3\%, 50.0\%]$.



Stall comparisons need individual observations as well as averages

In this teaching sample, recorded A orders waited less on average. The records alone do not guarantee a faster future order or a population-wide stall difference.



Pooling unequal mixtures can reverse a within-period comparison

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

This is a separate comparison of stalls across quiet and busy periods. The order counts in each period determine the pooled means.

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

$$\bar{w}_A = (10 \cdot 2 + 90 \cdot 10)/100 = 9.2,$$

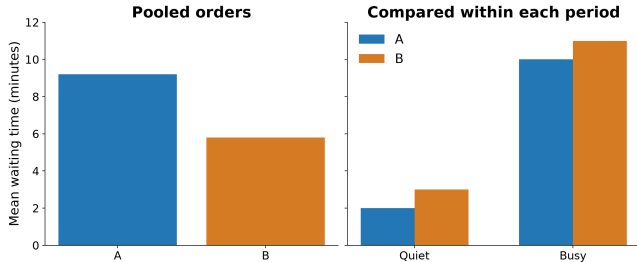
$$\bar{w}_B = (65 \cdot 3 + 35 \cdot 11)/100 = 5.8.$$



Compare the same stalls within each period

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

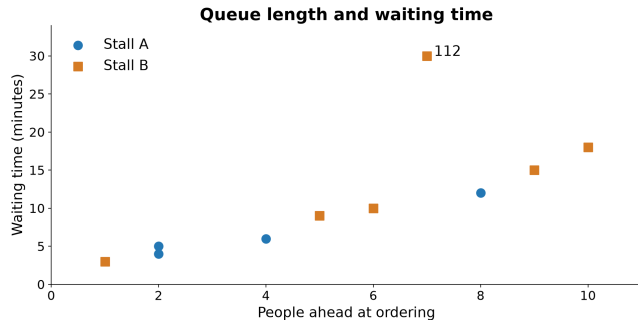
Within both periods, A is faster by one minute. The pooled comparison favors B because the stalls have different proportions of busy-period orders.



A scatter plot checks how two measured variables move together

Pearson correlation for these ten complete orders is approximately $r = 0.718$. It summarizes linear association, not the effect of intervening on queue length.

$r = 0.718$



Zero correlation can coexist with an exact nonlinear relationship

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Here $\bar{x} = 0$ and $\bar{y} = 2$. The numerator is $(-2)(2) + (-1)(-1) + 0(-2) + 1(-1) + 2(2) = 0$, so Pearson correlation is zero. Yet y is exactly determined by x .

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}.$$

x	-2	-1	0	1	2
$y = x^2$	4	1	0	1	4

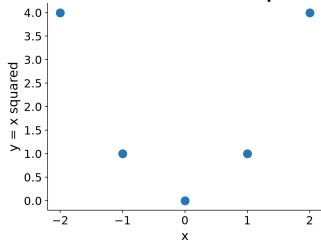
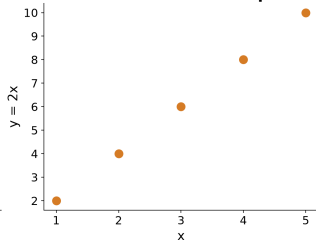
$$\bar{x} = 0 \quad \bar{y} = 2$$

$$(-2)(2) + (-1)(-1) + 0(-2) + 1(-1) + 2(2) = 0$$



A zero correlation can hide a curved rule

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

A curved relationship**A linear relationship**

On the left, $y = x^2$ determines the outcome exactly, yet symmetry makes Pearson correlation zero. Correlation measures linear association and can miss a curved relationship.



Exploration must respect prediction time and training-data boundaries

```
# After a deliberate train/validation/test split:  
mu = train["queue"].mean()  
sd = train["queue"].std(ddof=0)  
z_train = (train["queue"] - mu) / sd  
z_test = (test["queue"] - mu) / sd
```

$z = 2$

A queue measured when ordering can be an input for predicting wait. The realized wait and pickup timestamp become available later.



Worked example: give a defensible answer to the lunch question

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

The recorded orders favor A on average. Missing outcomes, sampling variation and the mix of order conditions limit any claim about the next order.

Part of the answer	Evidence available here
Observed comparison	A mean 6.75 min from 4 waits; B mean 14.17 min from 6
Variation	Individual waits overlap; B has a 30-minute order
Missing outcomes	Two A waits are unknown
Prediction limitation	One observed mean does not guarantee the next order
What to collect next	More dates, time periods and explicit reasons for missing outcomes

What can the lunch records actually establish?

ADVANCED COMPUTING AND DIGITAL TECHNOLOGY RESEARCH CENTER

Question

Stall A has a shorter recorded mean wait. Can we promise that the next order at A will be faster?

Evidence and reasoning

No. The mean summarizes the recorded orders. Missing waits, different busy-period mixtures and sampling variation can change the comparison. Compare like conditions and report the observed difference before making a future claim.

Check. A descriptive mean is not a guarantee for one future order.

