

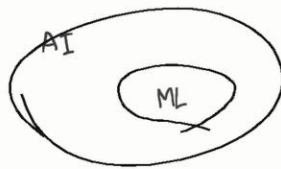
week 1

ML

- enables computers to learn without being explicitly programmed (Arthur Samuel 1959)
- Tom Mitchell = a program learns from experience (E) regarding a task (T) & a performance measure (P) if its performance improves w that experience
eg weather forecasting app experience = historical weather data
task = predicting today's weather perf = rains or not
- real world = face recognition spam filters in email recommendations (Netflix) fraud detection (Paypal) gen AI (GPT)

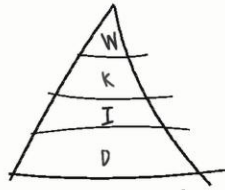
AI vs ML

- AI (broader) → creating a program that can sense act & adapt to mimic human behaviour
 - decision making
 - simulating natural intelligence to solve complex problems → leads to intelligence + wisdom
- ML = subset of AI
 - technique designed to make a machine improve its performance on a specific task by learning from data & experience
 - creating self-learning algorithms → knowledge
 - chess program w hand-written rules = AI but not ML (nothing is learned)
 - spam filter trained on my mails = ML



data vs information

- data = raw unprocessed facts numbers text or symbols
- lacking context no meaning on its own
- information = data all processed or contextualized to make it meaningful & useful
 - answers who what where when
 - DIKW pyramid (Data Information Knowledge Wisdom) = how raw facts are progressively transformed into actionable human judgement



- D = discrete objective facts or observations
- most abundant least valued (database of timestamps & product codes)
- I = data categorized calculated or condensed to give meaning (sales report sales A > sales B)
- K = info synthesised & analyzed to find patterns rules connections
- the how (A > B on weekends cuz leisure item promoted in fri newsletters)
- W = applying knowledge to make strategic judgements
- the why foresight ethics long term (move advertising budget → weekend)

importance

- global volume of data from enterprises etc keeps growing & problems get more complex → automation
- hand-written rules = costly & error prone
- humans provide data extract features
- the algorithm figures out its rules on its own to produce the result

non-ML vs ML

- both have the same pipeline dataset → feature extraction → rule → result
- the only difference is who writes the rule
 - non-ML → human
 - issue there r too many possible cases
 - eg IF smooth & orange THEN orange
 - IF smooth & red THEN apple
 - ML = give the algorithm lots of examples & it figures out the rules by itself
- algorithm = the code the programmer writes
- model = what comes out after the algorithm learns from data

categories

supervised

- every example has the correct label (photos tagged ^{cat or dog})
- classification = predict a specific class label from predefined data
- which one is it or what kind is it
 - determining if a financial profile falls to bad or fair or good

- IF $x > \theta_1$ AND $y > \theta_2$ THEN A
- the threshold numbers are the discriminants
- IF temperature > 31 AND humidity $> 90\%$ THEN sunny $\rightarrow 31$ & 90% are the discriminants
- IF payment history $< 45\%$ AND length of credit history $< 30\%$ AND type of credit $< 50\%$ THEN bad
- regression = predict a continuous real number
- how much or how many
 - $y = wx + w_0$
 - predict ML score from Statistics Discrete Math & Programming scores $\rightarrow$ find the weight of each

classification	regression
category / label	a specific number
which bucket?	how much?
pass / fail	student's exact final score

unsupervised

- no labels
- machine needs to find hidden structures in messy data
- never looks at target y only studies how features relate to each other
- Clustering (grouping profiles) see overall characteristics of ppl or things organize to distinct segments based on how similar they are
 - which of these things belong together in a group eg young parents // healthy eaters
 - grouping customers by their past transactions
 - finding similar pictures and categorizing news
 - clustering cuz nobody gave the groups a name beforehand
- association (connecting items rules) finds specific triggers or links between items or events that freq happen at the same time
 - if X happens what else is likely to happen eg if buy A 70% chance buy B
 - = basket analysis
- dimensionality reduction $\rightarrow$ squeezes many features down to fewer
 - eg dataset of students each student 50 features (scores in 50 subjects)

most of the features move together (if good at calculus also good at alin etc)

 - $50 \rightarrow 3$ math ability language ability creative ability
 - fewer features = faster training less memory less noise $\rightarrow$ prevent overfitting easier visualisation (plot on 2D chart)
- clustering vs dimensionality reduction

- clustering $\rightarrow$ rows (groups the samples)
- group segment similar customers
- dimensionality reduction $\rightarrow$ columns (fewer features)
- too many features compress visualize in 2D or 3D

other types

type	answer key	example
supervised	all labelled	photos tagged cat or dog
unsupervised	no labels	grouping customers
semi-supervised	some labels	100 tagged + 10000 untagged
reinforcement	rewards	game AI +1 win and -1 lose

- semi-supervised (leans more to supervised) labelling data is expensive so only label a small part $\rightarrow$ model learns from these labels (like Google Photos)
- reinforcement = agent takes actions in an environment & gets reward or penalty $\rightarrow$ over time learns which actions earn most reward

$P(Y|X)$

- the probability of Y given X
- $P(\text{spread} | \text{bread}) = 0.7 \rightarrow 70\%$ of customers who buy bread also buy choc spread
- $P(Y | X, D) =$ of the customers who are young & single (D) and bought bread (X), the fraction who also bought choc spread (Y)
- probability can never be > 1

terms

- column = feature = independent variable (X)
- row = sample = one data point
- target = y = dependent variable
- Boston table one row = a sample (one house area in Boston)
- target = house price $\rightarrow$ what were predicting
- python libs = Numpy (arrays) SciPy (numerical algorithms) matplotlib (plots) pandas (tables) scikit-learn (ML algorithms)
- no notebook this week

exam check

- E T P for a scenario
 - spam filter $\rightarrow$ E = mails marked spam or not and T = decide if a new mail is spam and P = share it gets right
- which task and why
 - price of a used car $\rightarrow$ regression cuz the answer is a number

- pass or fail → classification cuz the answer is a label
- customer groups with no names → clustering cuz there are no labels
- people who buy bread also buy spread → association cuz it links things that happen together
- supervised or not → does the data have the right answer
- AI or ML → is anything learned from data
- mix ups
 - classification and regression are both supervised
 - clustering groups rows and dimensionality reduction shrinks columns
 - algorithm = the code and model = what the code learned
 - feature = column = X and target = Y

week 2

Regression

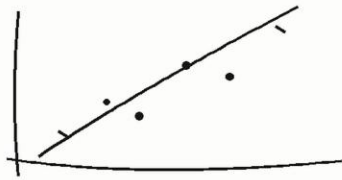
- supervised algorithm that predicts a continuous value from one or more variables
 - house price 24.0 → regression cuz the answer is a number on a scale
 - spam or not spam → not regression cuz the answer is a label
- X = independent variables = features
- Y = dependent variable = target
 - Boston table → the 13 columns CRIM to LSTAT are X and house price is Y

task	features	target
weather	temperature humidity pressure	wind speed
housing	room transport land size	house price
stocks	profit sector month economy	stock price
gasoline	weight horse power engine	MPG

- all four are regression cuz every target is a number

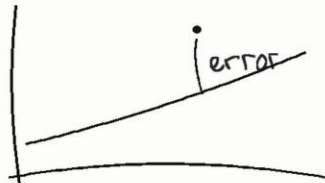
simple linear

- $\hat{y} = \theta_0 + \theta_1 x_1$
- $\hat{y}$ = predicted variable
- θ_0 = intercept → the value of $\hat{y}$ when x is 0
- θ_1 = slope → how much $\hat{y}$ changes when x goes up by 1
- x_1 = the single predictor
 - $\hat{y} = 1.0 + 1.7x \rightarrow$ at $x = 2$ we get $\hat{y} = 4.4$
 - slide example → $x_1 = 6.631$ rooms
- training = find the θ whose line fits the points best



error and metrics

- error = $y - \hat{y}$
 - real $y = 16.6$ and predicted $\hat{y} = 15 \rightarrow$ error = 1.5
- $MSE = (1/n) \sum (y - \hat{y})^2$
- Squaring $\rightarrow$ plus and minus errors cannot cancel and big errors are punished more
- Objective of linear regression = find the θ that minimise the MSE
- MAE = mean of $|y - \hat{y}|$
 - errors 0.3 -0.4 -0.1 0.2 $\rightarrow$ MAE = $1.0 / 4 = 0.25$
 - same units as y so it reads as off by 0.25 on average
- MSE = mean of the squared errors
 - same errors $\rightarrow 0.30 / 4 = 0.075$
 - one big miss dominates it cuz of the square
- RMSE = $\sqrt{MSE} \rightarrow$ back in the units of y
- R^2 = share of the variation in y that the model explains
 - 1 = perfect
 - 0 = no better than always guessing the mean
 - 0.73 = explains 73% of the variation



least squares

- $x = 1\ 2\ 3\ 4$ and $y = 3\ 4\ 6\ 8 \rightarrow \bar{x} = 2.5$ and $\bar{y} = 5.25$
- $\theta_1 = \sum (x - \bar{x})(y - \bar{y}) / \sum (x - \bar{x})^2$
- $\theta_0 = \bar{y} - \theta_1 \bar{x}$

x	y	$x - \bar{x}$	$y - \bar{y}$	product	$(x - \bar{x})^2$
1	3	-1.5	-2.25	3.375	2.25
2	4	-0.5	-1.25	0.625	0.25
3	6	0.5	0.75	0.375	0.25
4	8	1.5	2.75	4.125	2.25
sum				8.5	5

$$\theta_1 = 8.5 / 5 = 1.7$$

$$\theta_0 = 5.25 - 1.7 \times 2.5 = 1.0$$

$$\hat{y} = 1.0 + 1.7x$$

x	y	$\hat{y}$	error	error ²
1	3	2.7	0.3	0.09
2	4	4.4	-0.4	0.16
3	6	6.1	-0.1	0.01
4	8	7.8	0.2	0.04

$$MSE = 0.30 / 4 = 0.075$$

multiple linear

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

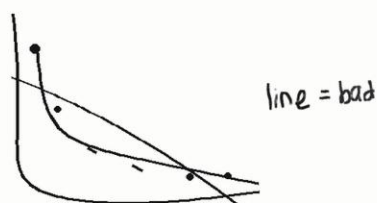
$$\hat{y} = \theta^T X \quad (x_0 = 1)$$

$$\text{Price} = \theta_0 + \theta_1 \text{CRIM} + \theta_2 \text{LSTAT} + \theta_3 \text{RM} + \theta_4 \text{DIS}$$

- same idea with one θ per feature
- slide example $\rightarrow \theta = [-4.56 \quad -0.09 \quad -0.65 \quad 6.47]$
- row 8 $\rightarrow$ CRIM 0.21124 and LSTAT 29.93 and RM 5.631
- price = $-4.56 - 0.09 \times 0.21124 - 0.65 \times 29.93 + 5.47 \times 6.631$
- price = $-4.56 - 0.019 - 16.46 + 30.80 = 9.76$
- real price 16.5 $\rightarrow$ error = 6.74
- reading the signs
 - minus on CRIM and LSTAT $\rightarrow$ more crime or more low status people means lower price
 - plus on RM $\rightarrow$ more rooms means higher price

non-linear

- use it when the data cannot be fit by a straight line
 - price vs crime rate drops fast then flattens $\rightarrow$ a line misses the bend
- shapes = linear and quadratic and cubic and polynomial



polynomial

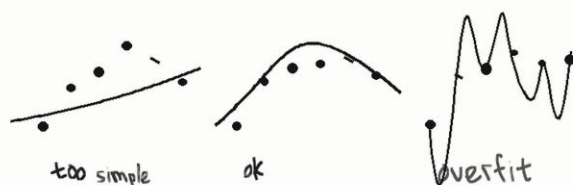
$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

- trick $\rightarrow$ rename $x_1 = x$ and $x_2 = x^2$ and $x_3 = x^3$

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

- now it is multiple linear regression so least squares solves it
- goal = minimise the sum of squared residuals between y and $\hat{y}$
- over-fitting = the model memorises the training points and fails on new ones
 - degree 10 curve through 12 points $\rightarrow$ train error near 0 and test error large

- sign $\rightarrow$ train error falls while test error rises
- underfitting = the model is too simple for the pattern
 - straight line on a curved cloud $\rightarrow$ both train and test error stay high



actual non-linear

- g is a non-linear function of the parameters θ
- how to tell $\rightarrow$ can I rename the x terms and get a plain sum of θ times something
 - $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow$ yes so it is still linear in θ
 - $\hat{y} = \theta_0 + \theta_2^2 x \rightarrow$ no cuz θ_2 is squared
 - $\hat{y} = \theta_0 + \theta_1 \theta_2^2 x \rightarrow$ no cuz two θ are multiplied
 - $\hat{y} = \log(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \rightarrow$ no cuz θ sits inside the log
 - $\hat{y} = \theta_0 / (1 + \theta_1^2 (x - \theta_2)) \rightarrow$ no cuz θ is in a power and a division $\rightarrow$ S curve
- no formula gives θ directly $\rightarrow$ found by repeating small improvements

regularization

- prevents overfitting by adding a penalty on large coefficients
 - Ridge = $\text{MSE} + \alpha(\theta_1^2 + \theta_2^2 + \dots)$
 - Lasso = $\text{MSE} + \alpha(|\theta_1| + |\theta_2| + \dots)$
- Ridge = squared penalty $\rightarrow$ shrinks every coefficient toward 0 but never ± 0
 - use when many features are highly correlated
 - 10 near duplicate columns $\rightarrow$ Ridge keeps all 10 with small coefficients
- Lasso = absolute penalty $\rightarrow$ forces some coefficients to exactly 0 $\rightarrow$ feature selection
 - use when only a few features matter
 - same 10 columns $\rightarrow$ Lasso keeps 2 or 3 and zeroes the rest
- α = penalty strength
 - $\alpha = 0 \rightarrow$ plain linear regression
 - α very big $\rightarrow$ all coefficients near 0 $\rightarrow$ underfit
 - pick α with GridSearchCV
- CAREFUL the slide writes Ridge (L2) and Lasso (L1)
- the standard naming is Ridge = L2 and Lasso = L1
- if asked $\rightarrow$ describe what each one does as well as the name

good practice

- not on the slides
- look first $\rightarrow$ scatter each feature vs the target and spot curves and outliers
- split with a seed so results repeat
`train_test_split(X, y, test_size=0.2, random_state=42)`
- baseline $\rightarrow$ always predicting the mean gives $R^2 = 0$ and the model must beat it
- start with a line then check the residual plot of error vs y
 - random cloud $\rightarrow$ fine
 - curve $\rightarrow$ wrong shapes so try polynomial
 - funnel $\rightarrow$ the error grows with y
- pick the degree with cross validation on train and never by the test score
- Ridge and Lasso $\rightarrow$ scale the features first and fit the scaler on train only
- alpha $\rightarrow$ search powers of 10 then use the alpha the search returns
- leakage = anything fitted on test rows $\rightarrow$ a Pipeline prevents it
`make_pipeline(StandardScaler(), PolynomialFeatures(2), Ridge(alpha=1))`
- coefficient sizes are only comparable when features are scaled
- which model
 - line first $\rightarrow$ polynomial if the residuals curve $\rightarrow$ Ridge if many correlated features $\rightarrow$ Lasso to drop useless ones

exam check

- predict by hand $\rightarrow$ plug the row into $y = \theta_0 + \theta_1 x_1 + \dots$ and add the terms
- error = real minus predicted and MSE = mean of the squared errors
- find θ by hand $\rightarrow$ the two least squares formulas then the table
- linear or not $\rightarrow$ check if it is a plain sum of θ times something
- overfit or underfit $\rightarrow$ compare train error with test error
- Ridge or Lasso $\rightarrow$ shrink all vs zero some
- Regression or classification $\rightarrow$ number vs label
- mix ups
 - MAE vs MSE $\rightarrow$ absolute vs squared
 - R^2 is a share and not an error $\rightarrow$ higher is better
 - `coef_` = slopes and `intercept_` = θ_0
 - `fit_transform` on train and `transform` only on test
 - fit learns θ and predict uses θ

lab code

- This is the notebooks own code

- Lines marked CAREFUL are mistakes or traps in it
- Every model is the same 4 steps pick the model fit on training rows
predict test rows score

load

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# the raw file stores each town on 2 lines, so read it as plain
  numbers first
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\t", skiprows=22, header=None)

# glue the 2 lines of each town back into one row of 13 features
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])

# the 3rd number on every second line is the house price = the
  target y
target = raw_df.values[1::2, 2]

# one table: 13 named feature columns + the target column
boston_df = pd.DataFrame(np.c_[data, target], columns =
    np.append(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
    'TAX', 'PTRATIO', 'B', 'LSTAT'], ['target']))
```

- CAREFUL: `sep="\t"` runs but prints SyntaxWarning: invalid escape sequence. Correct form: `sep=r"\t"`

explore

```
boston_df.describe()

# copy of the table for analysis
bdf = boston_df[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
    'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'target']]

# crime rate against price: drops fast, then flattens
plt.scatter(bdf.CRIM, bdf.target, color='blue')
plt.xlabel("Criminal Rate")
plt.ylabel("House Price")
plt.show()

histo = bdf[['CRIM', 'ZN']]
histo.hist()
```

- describe gives count mean std min max per column
- hist shows how each column is spread

split

- ```
one random number per row. True (about 80%) goes to training
dat = np.random.rand(len(boston_df)) < 0.8
train = bdf[dat]
test = bdf[~dat]
```
- ~ means NOT
  - The model learns from train only
  - Test rows check it honestly
  - CAREFUL no seed so the split and every score change on each run

## Simple linear

- ```
from sklearn import linear_model

regr = linear_model.LinearRegression()

# x must be 2-D (rows x features), so use double brackets
train_x = np.asanyarray(train[['CRIM']])
train_y = np.asanyarray(train[['target']])

# fit = least squares finds the best  $\theta$ 
regr.fit(train_x, train_y)
print('Coefficients:', regr.coef_)
print('Intercept:', regr.intercept_)

# draw the fitted line over the training points
plt.scatter(train.CRIM, train.target, color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], 'r')
```
- coef_ is the slope θ_1 , intercept_ is θ_0 . coef_[0][0] takes the number out of the 2-D array
 - The evaluation cell is blank in the lab (an exercise)
 - Standard version

```
test_x = np.asanyarray(test[['CRIM']])
test_y = np.asanyarray(test[['target']])
test_y_ = regr.predict(test_x)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ -
    test_y)))
print("Mean Square Error (MSE): %.2f" % np.mean((test_y_ -
    test_y) ** 2))
```
 - Lab output = coef -0.408 intercept 23.99 MAE 5.92 MSE 63.20

multiple linear

```
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['CRIM', 'LSTAT', 'RM']])
y = np.asanyarray(train[['target']])
```

```

regr.fit(X, y)
print('Coefficients:', regr.coef_)
print('Intercept:', regr.intercept_)

```

- Same model X now has 3 columns
- Each feature gets its own θ
- The evaluation cell is blank in the lab
- Standard version

```

y_hat = regr.predict(test[['CRIM', 'LSTAT', 'RM']])
x = np.asarray(test[['CRIM', 'LSTAT', 'RM']])
y = np.asarray(test[['target']])
print("Residual sum of squares: %.2f" % np.mean((y_hat - y)**2))
print('Variance score: %.2f' % regr.score(x, y))

```

- Lab output = coeffs -0.094, -0.605, 5.07
- Intercept -1.51
- Residual sum of squares 22.23
- Variance score 0.73
- CAREFUL that label is wrong
- A mean of squared errors is the MSE not the residual sum of squares
- Variance score = R^2
- 1 is perfect

polynomial

```

from sklearn.preprocessing import PolynomialFeatures
train_x = np.asarray(train[['CRIM']])
train_y = np.asarray(train[['target']])
test_x = np.asarray(test[['CRIM']])
test_y = np.asarray(test[['target']])

# degree 2: turn the column x into 3 columns: 1, x, x^2
poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)

# an ordinary linear regression on those columns = a curve in x
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
print('Coefficients:', clf.coef_)
print('Intercept:', clf.intercept_)

```

- The rename trick still linear in θ so least squares still works
- CAREFUL: `train_y_` is a misleading name. `fit` returns the model, not predictions
- The first coefficient is always 0 because the column of ones repeats the intercept
- Lab, degree 2 coeffs 0, -0.841, 0.0084

- Intercept 24.83
- The degree 3 cells are blank in the lab
- Standard version

```
poly3 = PolynomialFeatures(degree=3)
train_x_poly3 = poly3.fit_transform(train_x)
clf3 = linear_model.LinearRegression()
clf3.fit(train_x_poly3, train_y)
```

```
# test rows get the same columns. transform only, never fit again
test_y3_ = clf3.predict(poly3.transform(test_x))
```

- Lab degree 3 MAE 5.33 MSE 54.19
- Better than the straight line (MSE 63.20)

Ridge

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge = Ridge()

# the penalty strengths to try
parameters = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

# For each alpha: split train into 5 folds, train on 4, score on 1, repeat
ridge_regressor = GridSearchCV(ridge, parameters,
    scoring='neg_mean_absolute_error', cv=5)
ridge_regressor.fit(train_x, train_y)
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)

# "train using best ridge model"
ridge_model = Ridge(alpha=100)
ridge_model.fit(train_x, train_y)

test_y_ = ridge_model.predict(test_x)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ -
    test_y)))
print("Mean Square Error (MSE): %.2f" % np.mean((test_y_ -
    test_y) ** 2))
```

- Scoring is negative MAE because sklearn always maximises
- Lab output = best alpha 0.001 best score -6.65
- Then MAE 7.42 MSE 99.39
- CAREFUL: the search says alpha = 0.001 is best, but the lab then trains with alpha = 100. The correct line is Ridge(alpha=0.001), or ridge_regressor.best_estimator_

Lasso

- from sklearn.linear_model import Lasso
- The search, training and evaluation cells are blank in the lab. Same pattern as Ridge with Lasso()
- Lab output best alpha $1e-15$ best score -81.90 with a ConvergenceWarning
- Then Lasso(alpha=1) MAE 739 MSE 98.32
- CAREFUL same mistake
- The search picked $1e-15$ the lab trained with alpha = 1
- CAREFUL -81.90 against Ridges -6.55 means the two searches used different scoring so those two numbers cannot be compared
- CAREFUL Ridge and Lasso score worse (MAE 7.4) than plain regression (5.92) on the same feature
- That comes from a different random split not from the penalty
- A penalty on one unscaled feature does almost nothing
- Ridge and Lasso matter with many features scaled first

week 3

time series

- data in time order → the model reads the past and predicts future values
- every row has a timestamp
- it tracks variables across time
- points depend on each other so never shuffle the rows
 - monthly sales Jan to Dec → time series cuz the order carries the pattern
 - heights of 30 students → not a time series cuz the order means nothing
- univariate = time is the only independent variable
 - monthly sales alone → univariate cuz we predict sales from its own past
- multivariate = more than one independent variable
 - sales plus weather and holidays → multivariate cuz other columns help predict
- use cases
 - forecasting = predict future values from history
 - next months sales or traffic in 30 minutes → forecasting cuz the answer is a future value
 - anomaly detection = detect unusual behaviour
 - odd network activity or a strange transaction → anomaly cuz it breaks the usual pattern
 - pattern analysis = find trends and seasonal effects and changes

- higher sales on weekends or peak hours of power use $\rightarrow$ pattern cuz we describe what repeats

autocorrelation

- how a series relates to its own past values
- tells if past observations hold useful information for predicting the future
- lag $k = k$ steps back $\rightarrow$ lag 2 = the value 2 days before
- ACF at lag k = correlation between $y(t)$ and $y(t-k)$
 - a tall bar at lag 12 on monthly data $\rightarrow$ a yearly season cuz this month looks like the same month last year
- PACF = link between $y(t)$ and $y(t-k)$ after removing what the lags in between already explain
 - Jan affects MAR only through Feb $\rightarrow$ ACF at lag 2 is high and PACF at lag 2 is near 0
 - ACF = total link and PACF = direct link only
- which bars count $\rightarrow$ only the ones outside the shaded band
 - band = $1.96 / \sqrt{n}$
 - 48 months $\rightarrow$ band = 0.28
- lag 0 is always 1 so ignore it
- by hand $\rightarrow$ 12 15 14 18 20 19 23 with mean 17.29
 - deviations $\rightarrow$ -5.29 -2.29 -3.29 0.71 2.71 1.71 5.71
 - top = $\sum dev(t) \times dev(t-1) = 33.63$
 - bottom = $\sum dev^2 = 87.43$
 - ACF(1) = $33.63 / 87.43 = 0.386$

trend seasonality cycle

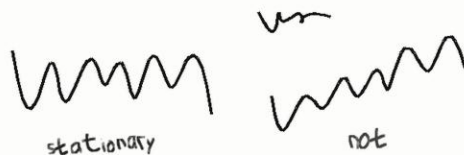
- trend = long term increase or decrease
 - an online store slowly sells more as its customer base grows $\rightarrow$ trend cuz it keeps going one way
- seasonality = a pattern that repeats at a predictable fixed interval
 - hourly daily weekly monthly quarterly or yearly
 - electricity use up every evening $\rightarrow$ seasonal cuz the gap is always one day
 - supermarket sales up every weekend and hotel bookings up every holiday and ice cream up every summer
 - airline passengers peak every year
- cycle = rises and falls whose timing and length are not fixed
 - economic expansions and recessions $\rightarrow$ cycle cuz nobody can say how long each one lasts
- test $\rightarrow$ can I say how many steps until it repeats

- yes $\rightarrow$ seasonality
- no $\rightarrow$ cycle



stationarity

- the statistical characteristics stay stable over time
- three conditions
 - constant mean
 - constant variance
 - covariance depends on the lag and not on the time
- consumption that climbs every month $\rightarrow$ not stationary cuz the mean keeps moving
- consumption that wobbles around one average with steady spread $\rightarrow$ stationary
- why it matters $\rightarrow$ ARIMA assumes a stationary series
- fix a trend by differencing $\rightarrow$ replace $y(t)$ with $y(t) - y(t-1)$
 - $1.2 \ 1.6 \ 1.4 \ 1.8 \ 2.0 \ 1.9 \ 2.3 \rightarrow +3 \ -1 \ +4 \ +2 \ -1 \ +4$
 - the climb is gone and the values hover around 1.8
- ADF test checks it
 - $p < 0.05 \rightarrow$ stationary
 - lab $\rightarrow p = 0.0002$ so stationary
 - $p = 0.40 \rightarrow$ not stationary so difference once and test again



ARIMA

- Auto Regressive Integrated Moving Average $\rightarrow$ uses past values to forecast the future
- p = autoregressive order = number of lags
 - $p = 2 \rightarrow$ today depends on the last 2 values
- d = how many times the series is differenced to become stationary
 - usually 1 to remove a trend
- q = moving average order = number of past errors used
 - $q = 1 \rightarrow$ today also corrects for the last forecast error
- ARIMA(1, 1, 1) $\rightarrow$ 1 past value and differenced once and 1 past error
- AIC = score for comparing models $\rightarrow$ fit quality with a penalty for extra parameters

- lower is better
- AIC 790 vs 1065 → pick 790
- steps for building it
 - visualization
 - ACF and PACF plot
 - seasonal variation modelling
 - stationary check
 - identify p & q for the non seasonal part

SARIMA

- Seasonal ARIMA = ARIMA extended for data with seasonal patterns
- handles periodic fluctuations so it forecasts seasonal data better
- $SARIMA(p, d, q)(P, D, Q, s)$
- p & q = the non seasonal part as in ARIMA
- P = seasonal autoregressive order
- D = seasonal differencing order → $y(t) - y(t-12)$
- Q = seasonal moving average order
- s = length of the season → 12 for monthly data
 - $SARIMA(1,1,1)(1,1,1,12)$ → the same three ideas again but looking 12 months back
- components = seasonal and autoregressive and integrated and moving average

SARIMAX

- SARIMA with exogenous regressors → also uses external variables
- S = seasonal → periodic patterns like weekly or monthly or yearly cycles
- AR = autoregressive → link between the current value and previous values
- I = integrated → differencing to make the series stationary
- MA = moving average → the current value depends on past error terms
- X = exogenous regressors → external variables that may affect the series
 - a holiday flag or a promo or the weather → X cuz it comes from outside the series
 - it must be known for the months being forecast
- which one
 - no season → ARIMA
 - season → SARIMA
 - season plus a known outside variable → SARIMAX

seasonal index

- not on the slides
- how strong a period is against an average period $\rightarrow 100 = \text{average}$
 $\text{index} = \text{period average} / \text{overall mean} \times 100$
- sales $Y_1 = 20 \ 30 \ 25 \ 45$ and $Y_2 = 24 \ 35 \ 28 \ 52$ and $Y_3 = 28 \ 40 \ 31 \ 59$

quarter	average	index
Q1	24	69.06
Q2	35	100.72
Q3	28	80.68
Q4	52	149.64

- overall mean = 34.75
- Q4 index 149.64 $\rightarrow$ Q4 sells about 50% above an average quarter
deseasonalised Q4 of $Y_3 = 59 / 1.4964 = 39.43$

choosing p d q

- not on the slides
- d first $\rightarrow$ ADF on the series
 $\bar{p} < 0.05 \rightarrow d = 0$
 - else difference once and test again $\rightarrow$ rarely more than 1
- read ACF and PACF on the differenced series and not the raw one
- PACF cuts off after lag p and ACF fades slowly $\rightarrow$ AR(p) so p from PACF and q = 0
- ACF cuts off after lag q and PACF fades slowly $\rightarrow$ MA(q) so q from ACF and p = 0
- both fade slowly $\rightarrow$ mixed so start with (1, d, 1)
 - cuts off = the bars drop inside the band and stay there
 - fades = the bars shrink bit by bit
 - PACF bars outside the band at lag 1 and 2 only $\rightarrow p = 2$ and $q = 0$
- season $\rightarrow$ ACF spikes at 12 and 24 means $s = 12$
 - seasonal spikes fade slowly $\rightarrow D = 1$
 - p from PACF at lag 12 and q from ACF at lag 12 $\rightarrow$ usually 0 or 1
- real plots are messy $\rightarrow$ use them for a first guess then try neighbours and compare AIC
 - only compare AIC between models with the same d and D
- check the fit $\rightarrow$ residuals should look like noise with no bar outside the band
- evaluate $\rightarrow$ split by time and hold out the last season and beat repeat last year
- the lab $\rightarrow$ ADF $p = 0.0002$ so $d = 0$ is enough and the grid pick of $d = 2$ over differences

exam check

- name the pattern → fixed gap is seasonality and no fixed gap is cycle and one direction is trend
- stationary or not → check mean and variance over time or read the ADFp value
- difference by hand → subtract the previous value
- ACF by hand → deviations then top over bottom
- read SARIMA(1,1,1)(1,1,1,12) → name each of the 7 numbers
- pick the model → season or not and outside variable or not
- seasonal index by hand → period average over overall mean times 100
- mix ups
 - ACF = total link and PACF = direct link
 - d removes a trend and D removes a season
 - q counts past errors and not past values
 - lower AIC is better and lower MAE is better
 - never shuffle and never split a time series randomly

lab code

- This is the notebooks own code
- Lines marked CAREFUL are mistakes or traps in it

load

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd

# ggg4 rows, one per item ordered
url = "https://raw.githubusercontent.com/trdsaputr/
sample/refs/heads/main/store_dataset.csv"
data_df = pd.read_csv(url)

# keep the date and the sales amount, turn the date text into
real dates
sales_df = data_df[['order Date', 'Sales']]
sales_df['order Date'] = pd.to_datetime(sales_df['order Date'])
- CAREFUL: filterwarnings("ignore") hides every warning, including useful
ones
- CAREFUL: assigning into a slice of data_df is pandas' SettingWithCopy
case. Safer: data_df[...].copy()
```

daily to monthly

```
import matplotlib.pyplot as plt
```

```
# many orders share a day: add them up per day, in date order
daily_sales = (sales_df.groupby('Order
Date')['Sales'].sum().sort_index())
daily_sales.plot(figsize=(14, 6), title='Daily Sales Record in the Store',
xlabel='Order Date', ylabel='Total Sales')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

daily is too jumpy. make the date the index, then add up per month

```
df1 = sales_df.set_index('Order Date')
monthly_sales = df1.resample('ME').sum()

plt.figure(figsize=(10, 6))
plt.plot(monthly_sales['Sales'], linewidth=3, c='blue')
plt.title("Monthly Sales")
```

- Raw orders become 48 monthly totals
- ME = monthend
- resample needs a date index

stationarity ACF PACF

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
def check_stationarity(timeseries):
```

```
# ADF test returns a tuple. [0] is the statistic, [1] is the
p-value
```

```
result = adfuller(timeseries, autolag='AIC')
```

```
p_value = result[1]
```

```
print(f'ADF Statistic: {result[0]}')
```

```
print(f'p-value: {p_value}')
```

```
print('Stationary' if p_value < 0.05 else 'Non-Stationary')
```

```
check_stationarity(monthly_sales['Sales'])
```

```
plot_acf(monthly_sales)
```

```
plot_pacf(monthly_sales)
```

```
plt.show()
```

- Lab output statistic -4.49 p = 0.0002 Stationary
- The lag 12 bar stands out a yearly season

ARIMA

```
import itertools
```

```
from statsmodels.tsa.arima.model import ARIMA
```

```
p = range(0, 4)
```

```
d = range(0, 3)
```

```
q = range(0, 4)
```

```
# every combination of p, d, q:  $4 \times 3 \times 4 = 48$ 
```

```
pda = list(itertools.product(p, d, q))
```

```
best_aic = np.inf
```

```
best_order = None
```

```
best_model = None
```

```
for order in pda:
```

```
    try:
```

```
        model = ARIMA(monthly_sales, order=order)
```

```
        results = model.fit()
```

```
        # lower AIC = better. Remember the best so far
```

```
        if results.aic < best_aic:
```

```
            best_aic = results.aic
```

```
            best_order = order
```

```
            best_model = results
```

```
    except:
```

```
        continue
```

```
print(f'Best ARIMA order: {best_order} with AIC: {best_aic}')
```

```
final_model = ARIMA(monthly_sales, order=best_order)
```

```
results = final_model.fit()
```

```
forecast_values = results.forecast(steps=12)
```

```
↳ output = best order (3, 2, 3), AIC 1065.76, with a  
ConvergenceWarning
```

CAREFUL a bare except hides every error even typos

- Better except Exception

- CAREFUL ADF said stationary yet the search picked $d = 2$

- AIC cannot fairly compare different d because differencing changes the data being fitted

- CAREFUL the model is fitted on all 48 months so

```
forecast(steps=12) predicts 2018
```

evaluate

```
plt.plot(monthly_sales[-12:].index, monthly_sales[-12:],
```

```
        label='Actual', color='blue', linestyle='-', marker='o')
```

```
plt.plot(forecast_values.index, forecast_values, label='Forecast',
```

```
        color='red', linestyle='--', marker='x')
```

```
plt.legend()
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```

observed = monthly_sales.tail(12)
mae = mean_absolute_error(observed, forecast_values)
mse = mean_squared_error(observed, forecast_values)

```

- Lab output = MAE 43745 MSE 2.38e9
- CAREFUL observed is 2017 the forecast is 2018
- Different months so this MAE is not a real accuracy
- Correct way: hold out first. train = monthly_sales[:-12], fit on train, forecast 12, compare with monthly_sales[-12:]

SARIMA

```

p, d, q = 1, 1, 1
P, D, Q, s = 1, 1, 1, 12

from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(monthly_sales, order=(p, d, q), seasonal_order=(P, D, Q, s))
results = model.fit()
forecast = results.forecast(steps=12)

observed = monthly_sales.tail(12)
mae = mean_absolute_error(observed, forecast)
mse = mean_squared_error(observed, forecast)

```

- (p d q) is the normal part (P D Q s) the seasonal part
- s = 12 months
- Lab output = MAE 10612 MSE 1.62e8
- The lab's best model
- CAREFUL the same 2017 against 2018 mismatch
- CAREFUL: the SARIMA plot draws forecast_values, which is the ARIMA forecast. It should draw forecast

SARIMAX

```

p, d, q = 1, 1, 1
P, D, Q, s = 1, 1, 1, 12
stationarity = False
invertibility = False

model = SARIMAX(monthly_sales, order=(p, d, q), seasonal_order=(P, D, Q, s),
                 enforce_invertibility=stationarity, enforce_stationarity=stationarity)
results = model.fit()
forecast = results.forecast(steps=12)

```

- Lab output = MAE 13031 MSE 2.37e8
- CAREFUL there is no exog so this is not really SARIMAX
- It is SARIMA with two checks switched off

- CAREFUL: `enforce_invertibility` uses the variable stationarity. `invertibility` is never used. Both are False, so the result is the same
- CAREFUL its plot also draws the ARIMA forecast
- A real SARIMAX passes an outside variable: `SARIMAX(train, exog=holiday,...)`, and `forecast(steps=12, exog=future_holiday)`
- Lab ranking = SARIMA 10612 SARIMAX 13031 ARIMA 43745
- A model that knows the 12-month season wins
- Always split by time and score on months the model never saw