

Machine Learning Notes for Handwriting, W1 to W3

Topics: W1 Intro to ML, W2 Regression, W3 Time Series Analysis.

W1. Intro to Machine Learning

1. Learning objectives

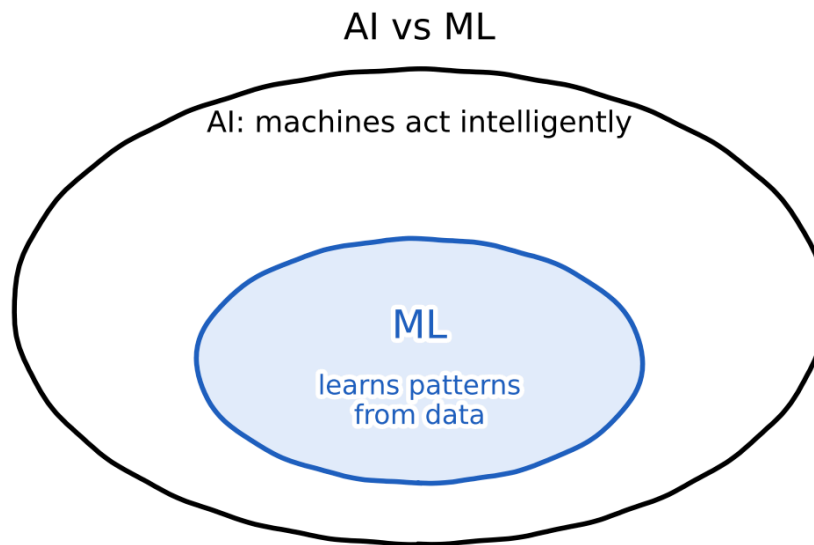
- Know what ML is and how it connects to AI.
- Tell supervised from unsupervised learning.
- Give daily-life examples of ML.

2. Definitions

- Samuel: ML is the field that gives computers the ability to learn without being explicitly programmed.
- Mitchell: a program learns from experience E on tasks T with performance measure P if its performance on T, measured by P, improves with E.
- Example, weather forecast app:
 - E = previous days' weather data
 - T = predict today's weather
 - P = whether the prediction (rain or not) is right
- E = Experience, T = Task, P = Performance measure.

3. AI vs ML

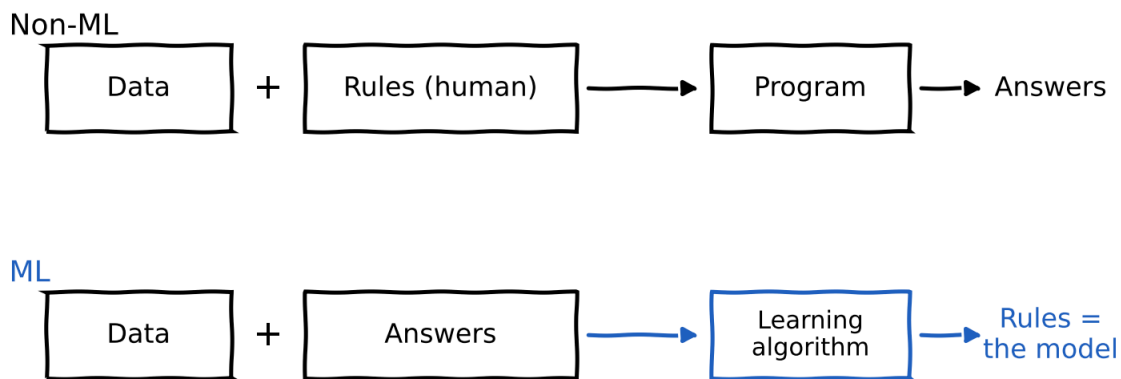
- AI = the big field of making machines act intelligently.
- ML = a part of AI. The machine learns patterns from data instead of following hand-written rules.
- Slide wording: AI = any technique that makes a program sense, act and adapt so that it can mimic human behaviour. ML = the type of technique that makes a machine improve its performance by learning from experience.
- AI vs ML (slide table): AI works as a program that does smart work, ML is a machine that takes data and learns from it. AI's goal is to simulate natural intelligence to solve complex problems, ML's goal is to learn from data on a certain task to maximise performance. AI is decision making, ML lets a system learn new things from data. AI develops a system that mimics humans, ML creates self-learning algorithms. AI leads to intelligence or wisdom, ML leads to knowledge.
- Example: a chess program built from hand-written rules is AI but not ML, because nothing is learned. A spam filter trained on your mails is ML, because it improves with data.



4. Non-ML way vs ML way

- Non-ML: Data + Rules (written by a human) -> program -> Answers.
- ML: Data + Answers -> learning algorithm -> Rules (the model). The model is then used on new data.
- Why it matters: rules are too many or too hard to write by hand (spam, faces, speech).
- Why ML is important (slides): data keeps growing and problems get more complex. Hand-written rules are costly and error prone, so we need a smarter machine.
- ML around us (slides): face recognition (Facebook, Instagram), product recommendation (Netflix), spam filter (Gmail), fraud detection (PayPal), generative AI (ChatGPT).
- Slide example of the non-ML way: datasets -> feature extraction (f1, f2, f3) -> rule -> result. The rule is written by hand: IF f1 is smooth and f2 is orange THEN orange. IF f1 is smooth and f2 is red THEN apple. In the ML way the machine learns that rule from the data.

NON-ML WAY vs ML WAY



Then the model is used on new data.

5. Big picture: 4 kinds of task

Association

- Find rules about things that happen together.
- Written as conditional probability $P(Y | X)$: chance of Y given X.
- Example: $P(\text{chocolate spread} | \text{bread}) = 0.7$. 70 percent of people who buy bread also buy chocolate spread.
- Slide name: Basket Analysis, finding the association between the products customers buy.
- With customer attributes D (gender, marital status, age) the rule becomes $P(Y | X, D)$.

Classification

- Predict a category (a label). Learned from labelled examples.
- Model = rules or a discriminant that separates the classes.
- Example: credit risk (low or high) from income and savings. Spam or not spam.
- Classification rule (slide): IF variable $X > \theta_1$ AND variable $Y > \theta_2$ THEN the decision is A. The boundary that separates the classes is the discriminant.
- Slide examples: IF temperature > 31 Celsius AND humidity $> 90\%$ THEN Sunny. IF payment history $< 45\%$, length of credit history $< 30\%$ and type of credit $< 50\%$ THEN Bad.

Regression

- Predict a number (continuous value).

$$y = wx + w_0$$

- w = slope, w_0 = intercept. Fit a line through the data points.
- Example: price of a used car from its mileage.
- Slide example: predict a student's ML score from their Statistics and Probability, Discrete Math and Programming scores. Training finds the weight of each score.

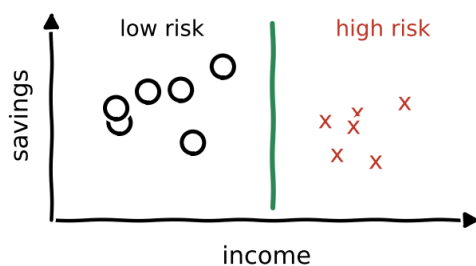
Clustering

- Group similar items when there are no labels. Find the structure in the data.
- Example: customer segments.
- Slide wording: the goal is to learn the regularities in the input (density estimation).
- Slide examples: grouping similar customers by their past transactions, finding similar pictures from one input picture, categorising news reports.

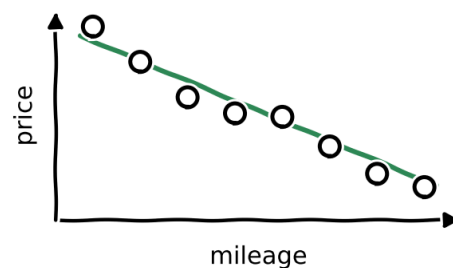
4 KINDS OF TASK

SUPERVISED (data has answers)

Classification: predict a label

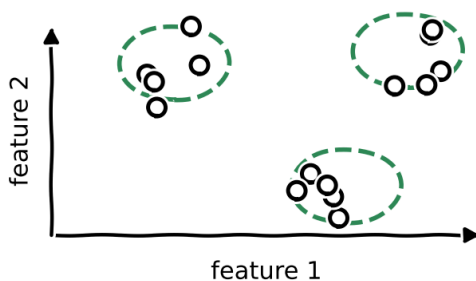


Regression: predict a number



UNSUPERVISED (no labels)

Clustering: group similar items



Association: things that happen together



$$P(\text{spread} \mid \text{bread}) = 0.7$$

6. Supervised vs unsupervised

- Supervised: data has the right answer (label). Classification and regression.
- Unsupervised: data has no labels. Clustering (and finding associations).
- Reinforcement learning = the third branch in the slide's big picture. The machine learns from rewards. A game-playing program that improves by winning and losing is reinforcement learning. The course focuses on supervised and unsupervised learning.
- Example: mails already marked spam or not spam are labelled, so learning from them is supervised. Purchase records with no group names are unlabelled, so grouping them is unsupervised.
- Which task is it? House price from rooms: regression, because the answer is a number. Spam or not spam: classification, because the answer is a label. Customer groups with

no names: clustering, because there are no labels. "People who buy bread also buy spread": association, because it links things that happen together.

7. Python stack

- NumPy: arrays and numeric maths.
- SciPy: scientific computing.
- matplotlib: plots.
- pandas: tables (DataFrames).
- scikit-learn: ML algorithms.

W1 quick recall

- ML = learn from experience E, task T, measure P.
- ML flips programming: data + answers give the rules.
- Tasks: association, classification, regression, clustering.

W2. Regression

1. Learning objectives

- Understand regression.
- Tell linear from non-linear regression.
- Apply non-linear regression algorithms and evaluate them.

2. What is regression

- A supervised algorithm that predicts a continuous value from one or more variables.
- X = independent variables = features (inputs).
- Y = dependent variable = target (output).
- Example: predict Boston house price (Boston dataset).
- In the Boston table the 13 columns from CRIM to LSTAT are X, and the target column (the price) is Y.

3. Applications

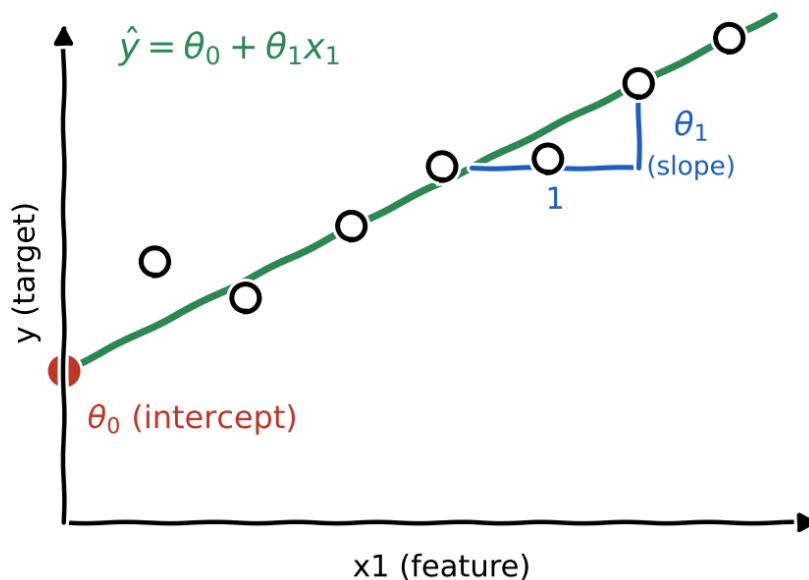
Task	Features (X)	Target (Y)
Weather forecast	temperature, humidity, air pressure change	wind speed
House market	rooms, distance to public transport, land size	house price
Stock market	company profit, sector outlook, month, economic status	stock price
Gasoline use	car weight, horse power, engine type	MPG (miles per gallon)

4. Simple linear regression (1 feature)

$$\hat{y} = \theta_0 + \theta_1 x_1$$

- $\hat{y}$ = predicted value. θ_0 = intercept. θ_1 = slope (weight of the feature).
- Training = find the θ values whose line fits the points best.
- Slide names: $\hat{y}$ is the predicted variable, θ_0 the intercept, θ_1 the slope, and x_1 the single predictor. In the slide, $x_1 = 5.631$ is the number of rooms.

SIMPLE LINEAR REGRESSION



Training = find the line that fits the points best.

5. How good is the model

- Error (residual) for one point:

$$\text{error} = y - \hat{y}$$

- Example: real $y = 16.5$, predicted $\hat{y} = 15$, error = $16.5 - 15 = 1.5$.
- Mean Squared Error over n points:

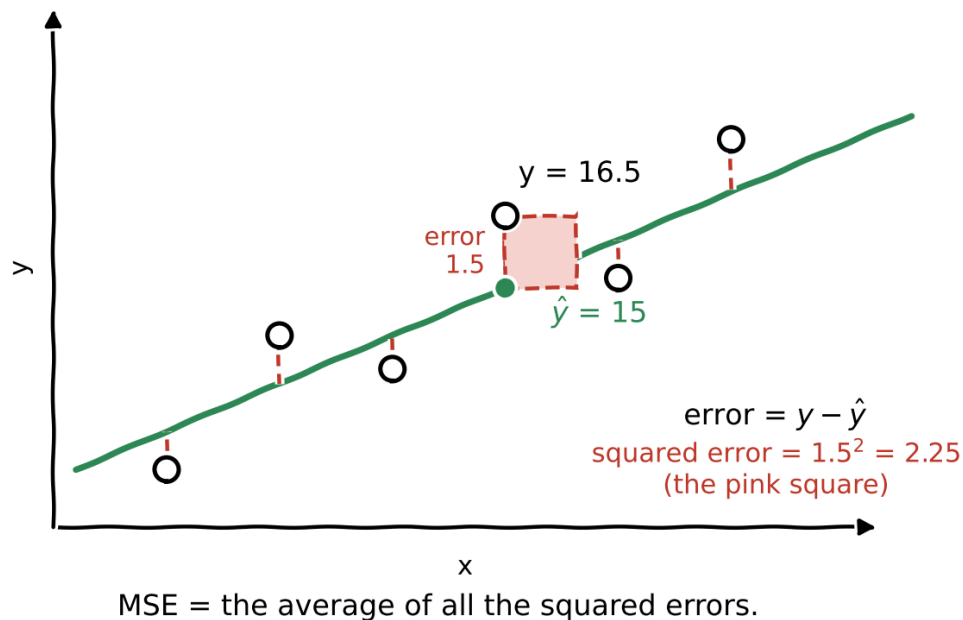
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Squaring stops positive and negative errors from cancelling and punishes big errors more.
- Lower MSE = better fit.
- Objective of linear regression (slide): find the best parameters θ that minimise the MSE.
- Other metrics used in the lab: MAE = mean of $|y - \hat{y}|$. R^2 = share of the variance explained (1 is perfect).

FIT AND CHECK A REGRESSION

1. Pick the model form (line, degree- n polynomial, ...).
2. Start with some θ . Predict $\hat{y}$ for every training point.
3. error = $y - \hat{y}$. Cost = MSE = average of error².
4. Change θ to make the cost smaller.
 - linear in θ : least squares gives θ directly
 - true non-linear: repeat small changes (iterate)
5. Test on new data: MAE, MSE, R^2 .

ERROR (RESIDUAL) AND SQUARED ERROR



Least squares by hand (tiny example)

- Data: $x = 1, 2, 3, 4$ and $y = 3, 4, 6, 8$. Means: $\bar{x} = 2.5$, $\bar{y} = 5.25$.
- The two formulas:

$$\theta_1 = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

- $\theta_0 = \bar{y} - \theta_1 \bar{x}$

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

- $\theta_1 = 8.5 / 5 = 1.7$. $\theta_0 = 5.25 - 1.7 \times 2.5 = 1.0$. So $\hat{y} = 1.0 + 1.7x$.

x	y	$\hat{y} = 1.0 + 1.7x$	error = $y - \hat{y}$	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
sum				0.30

- MSE = $0.30 / 4 = 0.075$.

6. Multiple linear regression (many features)

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

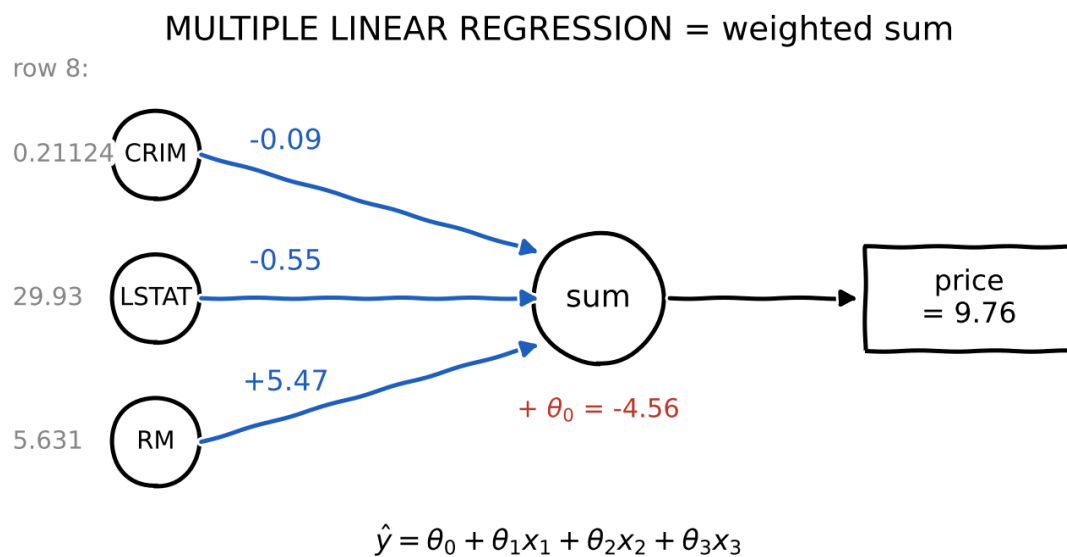
$$\hat{y} = \theta^T X \quad (\text{vector form, } x_0 = 1)$$

- Example (Boston):
price = -4.56 - 0.09 CRIM - 0.55 LSTAT + 5.47 RM

- CRIM = crime rate, LSTAT = % lower-status population, RM = rooms per house.
- Predict row 8 (CRIM = 0.21124, LSTAT = 29.93, RM = 5.631). Add up the terms:

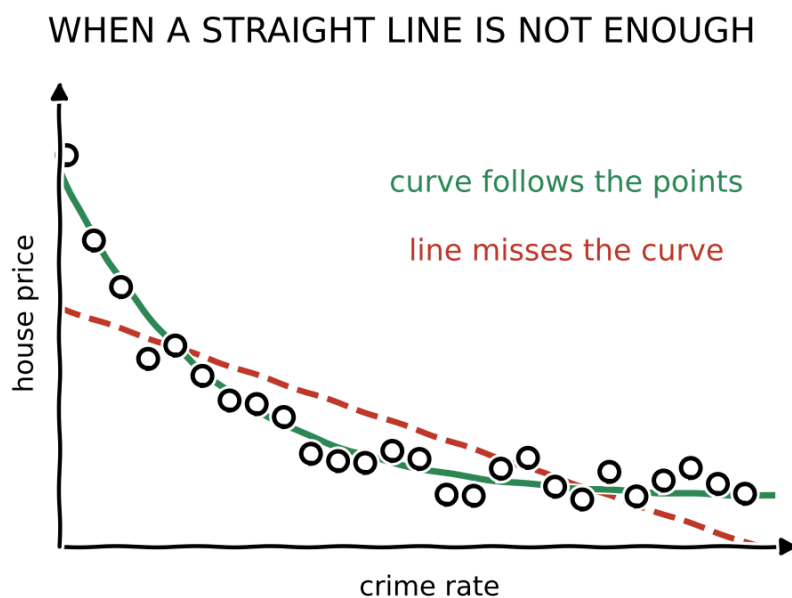
Term	Working	Value
θ_0	intercept	-4.56
CRIM	-0.09×0.21124	-0.019
LSTAT	-0.55×29.93	-16.46
RM	5.47×5.631	30.80
price	sum of the four	9.76

- The real target is 16.5, so the error is $16.5 - 9.76 = 6.74$.
- Reading the signs: more crime and higher LSTAT lower the price. More rooms raise it.
- Slide formula with four features: $\text{Price} = \theta_0 + \theta_1 \text{CRIM} + \theta_2 \text{LSTAT} + \theta_3 \text{RM} + \theta_4 \text{DIS} + \dots$
The worked example uses three of them, with $\theta^T = [-4.56, -0.09, -0.55, 5.47]$.



7. Non-linear regression

- Use it when the data cannot be fitted by a straight line.
- Example: house price vs crime rate. Price drops steeply at low crime, then flattens. A line misses the curve.
- Types by shape: linear (degree 1), quadratic (degree 2), cubic (degree 3), polynomial (degree n).



8. Polynomial regression

- Relationship between x and y is written as an n -degree polynomial of x :

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

- Trick: rename the terms as new features.
 - $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$
- Then it becomes multiple linear regression:

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

- Still linear in θ , so it is solved with least squares.
- Goal: minimise the sum of squared residuals (the gap between y and $\hat{y}$).

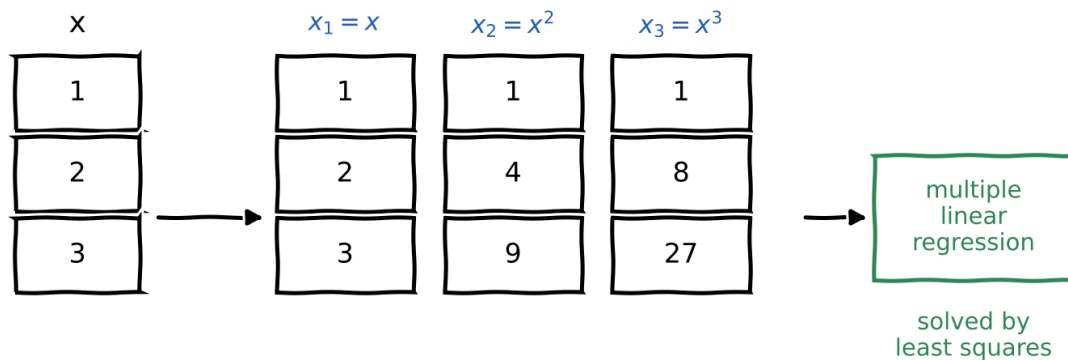
$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Higher degree fits the training data better, but too high overfits.
- Overfitting = the model memorises the training points and does badly on new ones. A degree-10 curve through 12 points is overfitting, because its training error is near zero and its test error is large.

POLYNOMIAL REGRESSION RECIPE

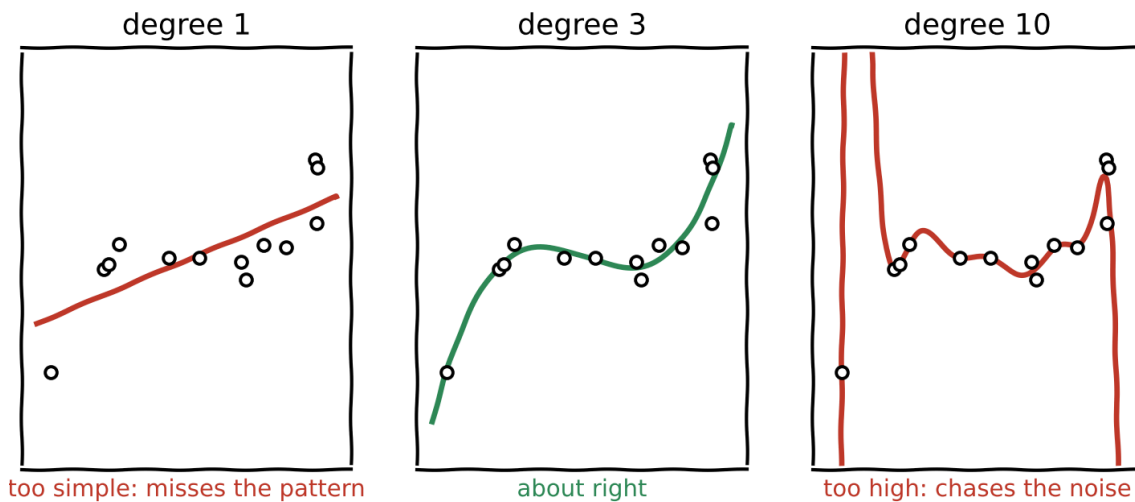
1. Pick the degree n .
2. Build the columns $x, x^2, \dots, x^n$.
code: `PolynomialFeatures(degree=n)`
3. Fit multiple linear regression on those columns.
code: `LinearRegression()`
4. Compare train error and test error.
train falls but test rises = degree too high (overfit)

POLYNOMIAL TRICK: rename the terms as features



$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \quad (\text{linear in } \theta)$$

FITTING: UNDERFIT, RIGHT, OVERFIT

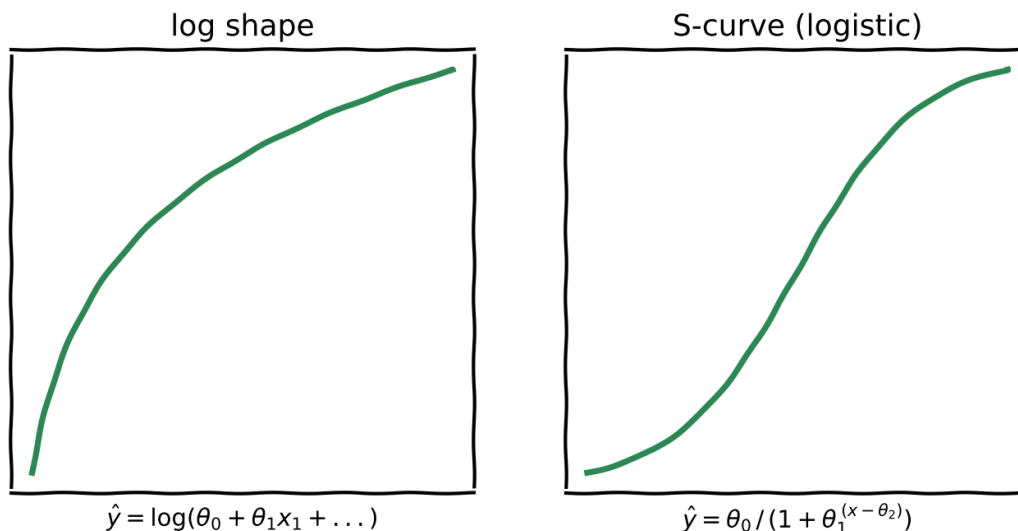


9. True non-linear regression

- The model is non-linear in the parameters θ . It cannot be turned into multiple linear regression.
- The features x themselves may or may not be non-linear.

- Examples from the slide, all non-linear in θ :
- $\hat{y} = \theta_0 + \theta_2^2 x$
- $\hat{y} = \theta_0 + \theta_1 \theta_2^2 x$
- $\hat{y} = \log(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)$
- $\hat{y} = \frac{\theta_0}{1 + \theta_1^{(x - \theta_2)}}$ an S-curve (logistic)
- Parameters are found by iterative optimisation, not a closed formula.

TRUE NON-LINEAR: non-linear in θ , cannot be renamed into a line



10. Regularization

- Purpose: prevent overfitting in linear regression by penalising large coefficients.
- Ridge (L2 regularization)
 - Shrinks coefficients towards zero with a squared penalty.
 - Use when many features are highly correlated.
- Lasso (L1 regularization)
 - Forces some coefficients to exactly zero, so it selects features.
 - Use when only a few features are important.
- Memory hook: Ridge shrinks (squared penalty, L2). Lasso zeroes out (absolute penalty, L1).
- Warning: the slide labels them the other way round, "Ridge (L1 Regularization)" and "Lasso (L2 Regularization)". The standard naming is Ridge = L2 and Lasso = L1. The descriptions on the slide are right: Ridge shrinks with a squared penalty, Lasso forces some coefficients to exactly zero.

- Example: a house-price model with rooms, crime rate and ten near-duplicate columns. Ridge keeps all twelve and makes each coefficient small. Lasso keeps a few and sets the rest to 0.

REGULARIZATION = cost + penalty

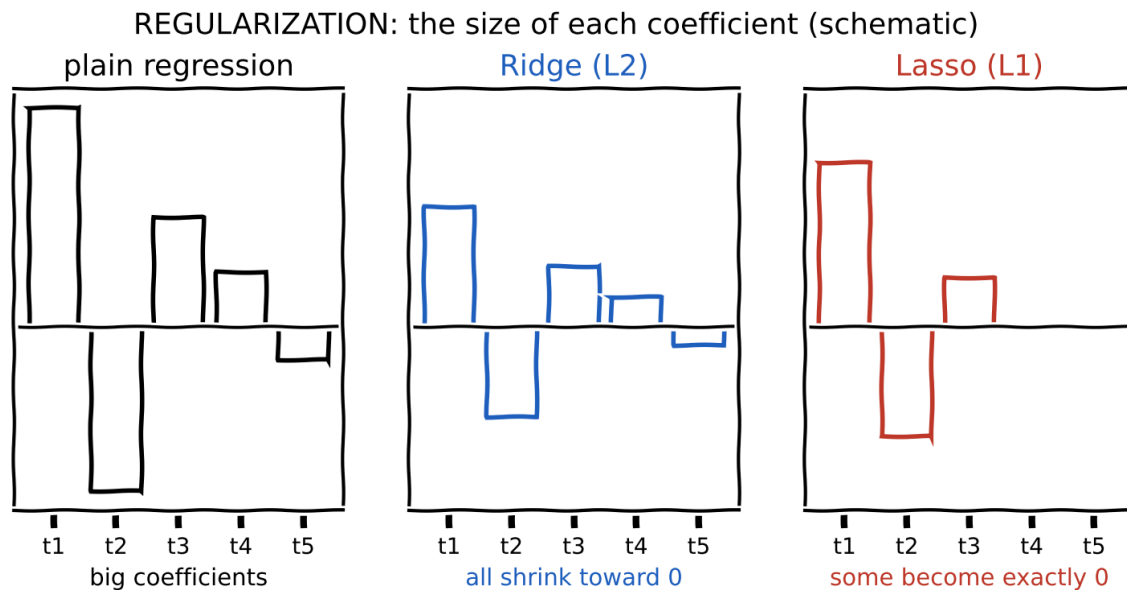
Ridge (L2): minimise $MSE + \alpha \times (\theta_1^2 + \theta_2^2 + \dots)$

Lasso (L1): minimise $MSE + \alpha \times (|\theta_1| + |\theta_2| + \dots)$

α = penalty strength (alpha in code). $\alpha = 0$ is plain regression.

Bigger α = smaller coefficients.

Choose α by trying values: GridSearchCV.



11. Lab results (Boston data, Python)

Model	Result
Simple LR, feature CRIM	coef -0.408, intercept 23.99, MAE 5.92, MSE 63.20
Multiple LR, 3 features	coefs [-0.094, -0.605, 5.07], intercept -1.52, R^2 0.73
Polynomial degree 2 and 3	degree 3: MAE 5.33, MSE 54.19, better than the straight line (MSE 63.20)
Ridge and Lasso (GridSearchCV)	Ridge (alpha 100): MAE 7.42, MSE 99.39. Lasso (alpha 1): MAE 7.39, MSE 98.32. Do not compare with the other rows, see the lesson below.

- Steps used: load data with pandas, split train/test, fit with scikit-learn, predict, compute MAE, MSE, R^2 .
- Polynomial: PolynomialFeatures(degree) then LinearRegression.
- Ridge and Lasso: search the penalty strength alpha with GridSearchCV.
- Lesson: the split is random with no seed, so these scores likely come from a different split than the linear rows. On one feature the penalty barely moves the line. Over 200 random splits, plain regression, Ridge and Lasso all average MAE 6.15. Regularization matters with many features.
- Next meeting: Tree Classifier.

12. Lab code, step by step

- Python with scikit-learn. Every model in the lab repeats the same four lines: pick the model, fit it on training rows, predict on test rows, score the prediction.
- Comments say what each line does. The split is random with no seed, so your numbers change on every run.

THE SAME FOUR LINES FOR EVERY MODEL

```
model = LinearRegression()      # 1. pick the model
model.fit(X_train, y_train)     # 2. learn  $\theta$  from training rows
y_hat = model.predict(X_test)   # 3. predict rows it has not seen
compare y_hat with y_test      # 4. MAE, MSE,  $R^2$ 
```

Load and split

```
import numpy as np, pandas as pd
url = "https://lib.stat.cmu.edu/datasets/boston"
# raw file: each town is split over two lines
raw_df = pd.read_csv(url, sep=r"\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
         'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
# one table: 13 feature columns plus the price
bdf = pd.DataFrame(np.c_[data, target], columns=names + ['target'])
# random 80/20 split (no seed, so numbers change every run)
dat = np.random.rand(len(bdf)) < 0.8
train, test = bdf[dat], bdf[~dat]
```

- It builds one table with 13 feature columns and the price, then keeps about 80 percent of the rows for training and the rest for testing.

Simple linear regression (one feature)

```
from sklearn import linear_model
# fit needs 2-D X (rows x features) and the target y
train_x = np.asanyarray(train[['CRIM']])
train_y = np.asanyarray(train[['target']])
regr = linear_model.LinearRegression()
regr.fit(train_x, train_y)          # least squares finds  $\theta$ 
print(regr.coef_, regr.intercept_) #  $\theta_1$  and  $\theta_0$ 

# evaluate on test rows the model has not seen
test_x = np.asanyarray(test[['CRIM']])
test_y = np.asanyarray(test[['target']])
test_y_ = regr.predict(test_x)      #  $\hat{y}$  for each test row
print(np.mean(np.absolute(test_y_ - test_y))) # MAE
print(np.mean((test_y_ - test_y) ** 2))      # MSE
```

- It fits $\text{price} = \theta_0 + \theta_1 \text{CRIM}$ by least squares, then scores the line on test rows. The lab got coefficient -0.408, intercept 23.99, MAE 5.92, MSE 63.20.

Multiple linear regression (three features)

```
cols = ['CRIM', 'LSTAT', 'RM']
regr = linear_model.LinearRegression()
x_tr = np.asanyarray(train[cols])
y_tr = np.asanyarray(train[['target']])
regr.fit(x_tr, y_tr)
print(regr.coef_, regr.intercept_) #  $\theta_1.. \theta_3, \theta_0$ 

x = np.asanyarray(test[cols])
y = np.asanyarray(test[['target']])
print(np.mean((regr.predict(x) - y) ** 2)) # MSE
print(regr.score(x, y))                    #  $R^2$ , 1 = perfect
```

- The same model with three input columns. The lab got coefficients -0.094, -0.605, 5.07, intercept -1.52, R^2 0.73.

Polynomial regression (degree 3)

```
from sklearn.preprocessing import PolynomialFeatures
# columns [1, x, x2, x3]: the rename-the-terms trick
poly = PolynomialFeatures(degree=3)
train_x_poly = poly.fit_transform(train_x)
# an ordinary linear regression on those columns
clf = linear_model.LinearRegression()
clf.fit(train_x_poly, train_y)
print(clf.coef_, clf.intercept_)
# test rows get the same columns: transform, not fit_transform
test_y_ = clf.predict(poly.transform(test_x))
print(np.mean(np.absolute(test_y_ - test_y))) # MAE
print(np.mean((test_y_ - test_y) ** 2))      # MSE
```

- PolynomialFeatures builds the columns 1, x, x², x³ and the regression treats them as ordinary features. The first coefficient is always 0 because the column of ones repeats the intercept. The lab got MAE 5.33 and MSE 54.19, better than the straight line (MSE 63.20).

Ridge and Lasso (search for alpha)

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV
# try each alpha with 5-fold cross-validation, scored by MAE
alphas = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
ridge = GridSearchCV(Ridge(), alphas, cv=5,
                    scoring='neg_mean_absolute_error')
ridge.fit(train_x, train_y)
print(ridge.best_params_) # the alpha that scored best
# best_estimator_ is already refit with that alpha
test_y_ = ridge.best_estimator_.predict(test_x)
print(np.mean(np.absolute(test_y_ - test_y))) # MAE
# Lasso: the same three steps with Lasso() in place of Ridge()
```

- GridSearchCV tries each alpha on 5 folds of the training rows and keeps the best. In the lab the final models were trained by hand with alpha 100 (Ridge) and alpha 1 (Lasso), not the alpha the search picked. That gave MAE 7.42 and 7.39. Alpha does not explain the gap to the linear model's 5.92. The likely cause is a different random split.

Score in one line each

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error, r2_score
mean_absolute_error(test_y, test_y_)    # MAE
mean_squared_error(test_y, test_y_)     # MSE
r2_score(test_y, test_y_)                # R2
```

W2 quick recall

- Regression predicts a number. Linear: $\hat{y} = \theta^T X$. Error = MSE.
- Polynomial regression = new features x, x^2, x^3 , then linear regression.
- Ridge = L2, shrink. Lasso = L1, zero out. Code: every model is fit, predict, score.

W3. Time Series Analysis

1. Learning objectives

- Understand the concept of time series analysis.
- Identify the trend and the seasonality in data.
- Build an ML model for time series prediction.

2. What is a time series

- A time series is data in time order. A time series model reads the past and predicts future values.
- Three traits: every row has a timestamp. It tracks variables across time. Neighbouring points depend on each other.
- Because points depend on each other, never shuffle the rows.

Type	Meaning	Example
Univariate	Only the series itself and time	Monthly sales, nothing else
Multivariate	Other variables help predict	Sales plus weather or holidays

3. Use cases

Use	What it does	Example
Forecasting	Predict future values from history	Next month's sales, traffic in 30 minutes
Anomaly detection	Spot unusual behaviour	Odd network activity, odd transaction pattern
Pattern analysis	Find trend, season, change	Weekend peaks, peak hours of power use

4. Autocorrelation

- Autocorrelation is how a series relates to its own past. Lag k means k steps back.
- Autocorrelation at lag k = correlation between y_t and y_{t-k} .
- ACF is the plot of that value for every lag. A tall bar at lag k means the value k steps ago helps predict today.
- PACF is the same with the shorter lags removed. It shows what lag k adds by itself.
- To see lag k , shift the column down k rows. The top k rows are empty.
- Slide example: lag 2 is the observation from 2 days before, lag 3 from 3 days before. Autocorrelation tells whether past observations hold useful information for predicting future ones. Its strength is measured with autocorrelation coefficients (ACF).

Worked example: lag 1 by hand

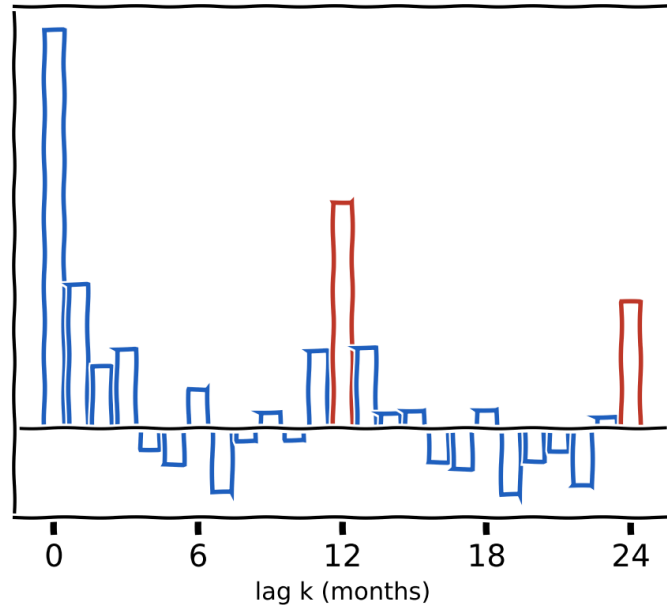
- Data: 12, 15, 14, 18, 20, 19, 23. Mean = 17.29.
- Distance from the mean: -5.29, -2.29, -3.29, 0.71, 2.71, 1.71, 5.71.
- Top = $\sum (\text{dev at } t) \times (\text{dev at } t-1) = 33.63$.
- Bottom = $\sum \text{dev}^2 = 87.43$.
- ACF at lag 1 = $33.63 / 87.43 = 0.385$. At lag 2 it is $23.55 / 87.43 = 0.269$.

AUTOCORRELATION: how $y(t)$ relates to $y(t-k)$

$y(t)$	lag 1	lag 2
72.5		
70.7	72.5	
62.5	70.7	72.5
57.5	62.5	70.7

shift the column down
by k rows = lag k

ACF of monthly sales: lag 12 stands out (0.57)

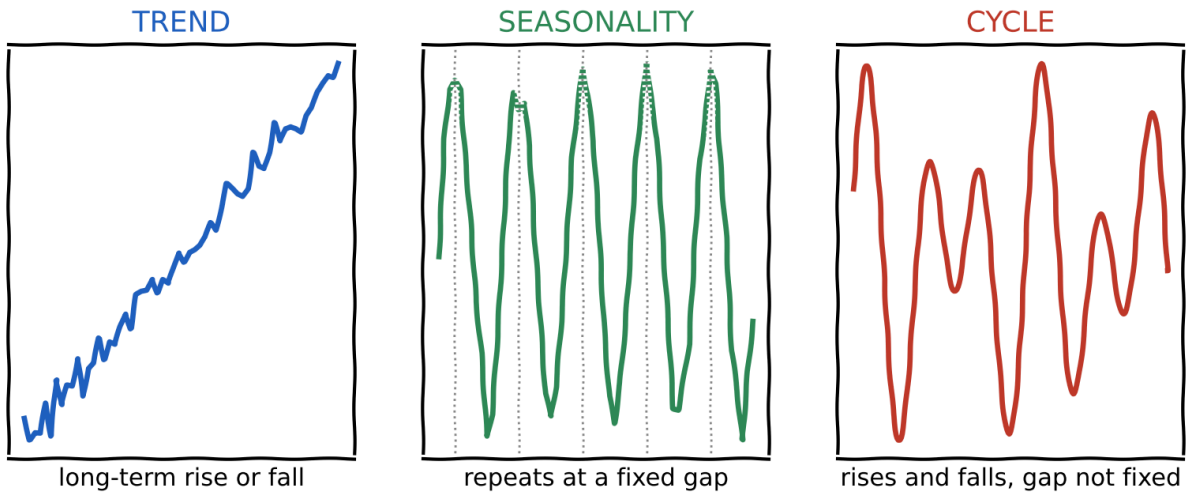


5. Trend, seasonality, cycle

Pattern	Meaning	Example
Trend	Long-term rise or fall	A store's customer base grows, sales rise
Seasonality	Repeats at a fixed, known gap	Demand up every holiday season
Cycle	Rises and falls, gap not fixed	Economic booms and recessions

- Seasonality gaps can be hourly, daily, weekly, monthly, quarterly or yearly.
- Slide examples of seasonality: electricity use rises every evening, supermarket sales rise every weekend, hotel bookings rise in holiday seasons, ice cream sales rise in summer. Airline passenger counts peak every year.
- Test: can you say how many steps until it repeats? Yes means season. No means cycle.

THREE PATTERNS IN TIME



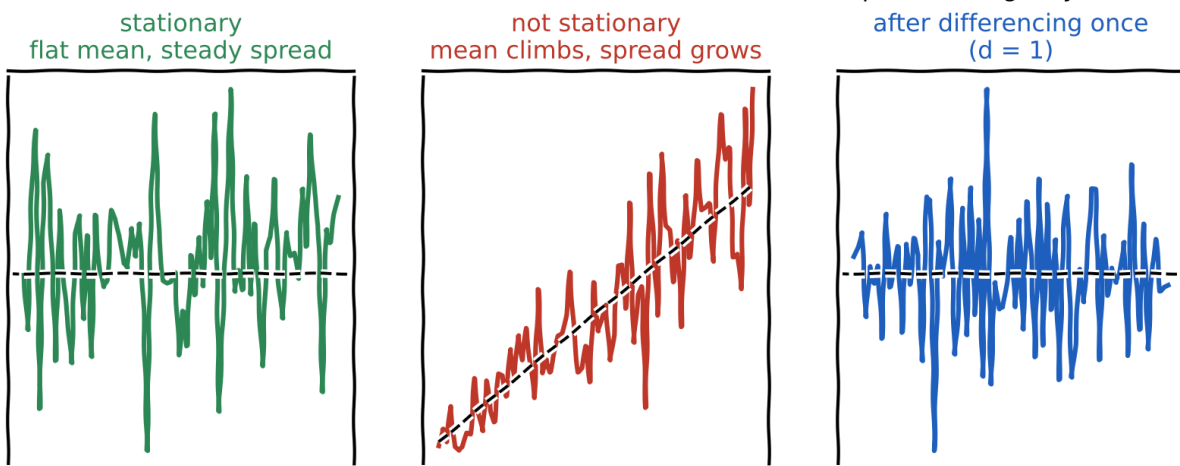
6. Stationarity

- A series is stationary when its statistics stay stable over time.
- Three conditions: constant mean. Constant variance. Covariance depends on the lag, not on the time.
- Consumption that climbs every month is not stationary. Consumption that wobbles around one average is.
- Fix a trend by differencing: replace y_t with $y_t - y_{t-1}$. Differencing once is $d = 1$.
- ADF test checks it. Null: not stationary. If $p < 0.05$, call it stationary.

Worked example: differencing

- Data: 12, 15, 14, 18, 20, 19, 23 climbs. Differences: +3, -1, +4, +2, -1, +4.
- The differences hover around +1.83. No more climb.

STATIONARITY: constant mean, constant variance, covariance depends on lag only



7. ARIMA

- ARIMA = AutoRegressive Integrated Moving Average. It forecasts from past values and past errors.
- Slide parameter names: p = autoregressive order, the number of lags. d = differentiation, how often the series must be differenced to become stationary (usually 1, to remove trends). q = moving average order, the number of past errors used in the model.
- Example: ARIMA(1, 1, 1) uses 1 past value, differences the series once, and uses 1 past error.

Part	Meaning	Read it from
p (AR)	How many past values to use	PACF cuts off after lag p
d (I)	How many times to difference	Run ADF until stationary, usually 0 to 2
q (MA)	How many past errors to use	ACF cuts off after lag q

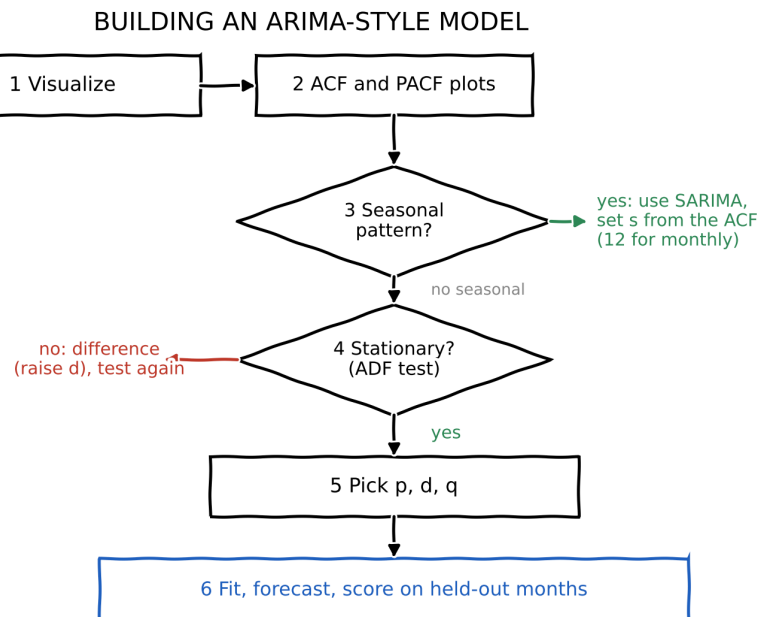
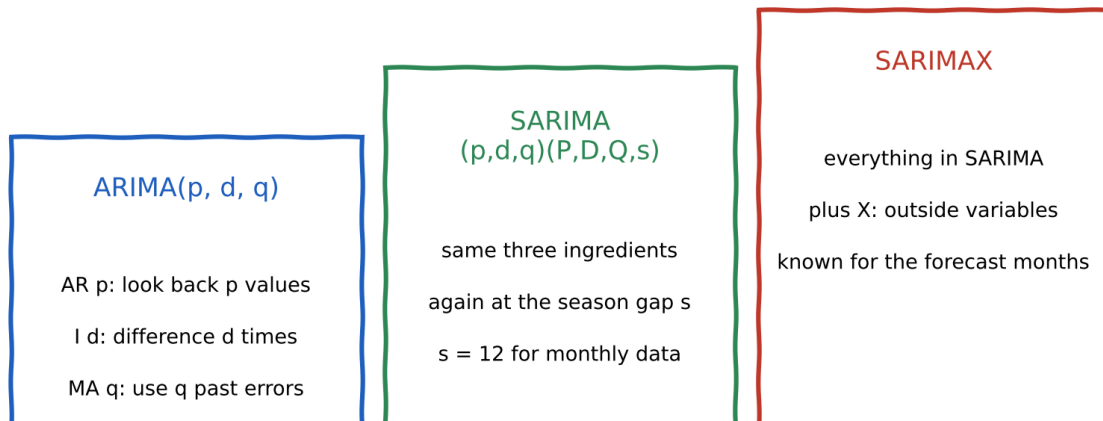
- The p and q reading rule is standard practice and is not on the slides. Real data is messy, so the lab tries many orders and keeps the lowest AIC.
- AIC = fit quality with a penalty for extra parameters. Lower is better.

8. SARIMA and SARIMAX

- SARIMA(p,d,q)(P,D,Q,s) adds a seasonal copy of p, d, q at the season gap s. For monthly data s = 12.
- Slide wording: SARIMA is an extension of ARIMA for data with seasonal patterns. It handles periodic fluctuations and forecasts seasonal data better. Its components are the seasonal, autoregressive, integrated and moving average components.
- Slide parameter names: p autoregressive order, d number of non-seasonal differences, q moving average order, P seasonal autoregressive order, D seasonal differencing order, Q seasonal moving average order, s length of the seasonal period (12 for monthly data).
- Seasonal differencing (D = 1) means $y_t - y_{t-12}$. This is what removes a season. Ordinary differencing removes a trend.
- SARIMAX adds X: outside variables (exogenous) such as a holiday flag. The flag must be known for the months you forecast.
- SARIMAX components (slide): Seasonal (S) captures periodic patterns such as weekly, monthly or yearly cycles. Autoregressive (AR) is the relationship between the current value and previous values. Integrated (I) is differencing to make the series stationary. Moving Average (MA) is the dependency of the current value on past error terms. Exogenous Regressors (X) are external variables that may affect the series.
- Recipe to build the model: visualise, ACF and PACF, check seasonality, check stationarity, pick p, d, q.

- Slide names for the steps: Visualization, ACF and PCF plot, Seasonal variation modelling, Stationary check, Identify p, d, q for the non-seasonal series.

THE ARIMA FAMILY: each model adds one ingredient



9. Seasonal index

- Seasonal index = how strong a period is compared with an average period. 100 = average. 150 = 50 percent above average.
- Beyond the slides, and useful for forecasts. It tells you what a seasonal part is worth.

Recipe (quarterly data)

- Step 1. Average each quarter across all years.
- Step 2. Average those averages. This is the overall mean.
- Step 3. Index = quarter average / overall mean $\times$ 100. The indices add up to 400.
- Deseasonalise: actual / (index / 100). Forecast: deseasonalised level $\times$ (index / 100).

Worked example

- Sales: Y1 = 20, 30, 25, 45. Y2 = 24, 35, 28, 52. Y3 = 28, 40, 31, 59.
- Quarter averages: Q1 = 24, Q2 = 35, Q3 = 28, Q4 = 52. Overall mean = 34.75.

Quarter	Average	Index
Q1	24	69.06
Q2	35	100.72
Q3	28	80.58
Q4	52	149.64

- Q4 of Y3 is 59. Deseasonalised: $59 / 1.4964 = 39.43$. Every quarter of Y3 deseasonalises to about 38.5 to 40.5, so the season explains the ups and downs.

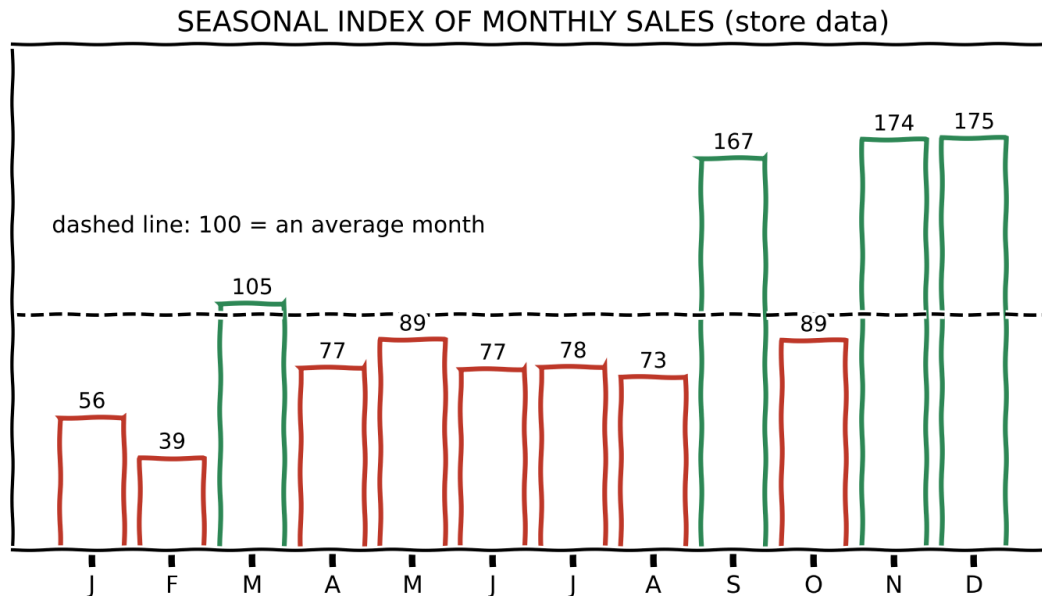
Monthly version (store data)

- With a trend, the plain average is unfair, because later years sit higher. Fix: divide each month by a 12-month centred moving average, then average the ratios per calendar month. statsmodels seasonal_decompose does this.

Month	Jan	Feb	Mar	Apr	May	Jun
Index	56	39	105	77	89	77

Month	Jul	Aug	Sep	Oct	Nov	Dec
Index	78	73	167	89	174	175

- Read: November and December sell about 74 percent above an average month. September is strong too. February is the weakest at 39.



10. Lab results (store data, Python)

- Data: 9994 order rows from 2014 to 2017. Add them per month to get 48 points.

Step	Result
ADF test on monthly sales	statistic -4.49, p = 0.0002, stationary
ACF	lag 12 stands out, so one-year season
ARIMA grid search, lowest AIC	(3, 2, 3), AIC 1065.76
ARIMA forecast	MAE 43745, MSE 2.38e9
SARIMA (1,1,1)(1,1,1,12)	MAE 10612, MSE 1.52e8
SARIMAX, same orders, flags relaxed	MAE 13031, MSE 2.37e8

- The lab's winner is SARIMA. A season is part of the data, so the model that knows $s = 12$ does better.

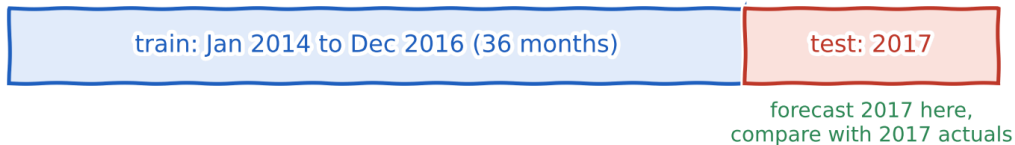
Read the lab carefully

- The lab fits on all 48 months, then forecasts the next 12, which are 2018. It scores those against the last 12 actual months, which are 2017. Different months, so the MAE is not a real accuracy.
- The SARIMA and SARIMAX plots draw the ARIMA forecast again, because the plot code uses `forecast_values`. Only the MAE lines use the right forecast.

- The lab's SARIMAX has no outside variable. It only switches off two checks, so it is SARIMA with looser rules.
- ADF says stationary, yet the grid picks $d = 2$, and the fit warns that it did not converge. Trust the AIC pick less than the score.

HOLD OUT THE LAST 12 MONTHS, THEN FORECAST THOSE SAME MONTHS

RIGHT



THE LAB



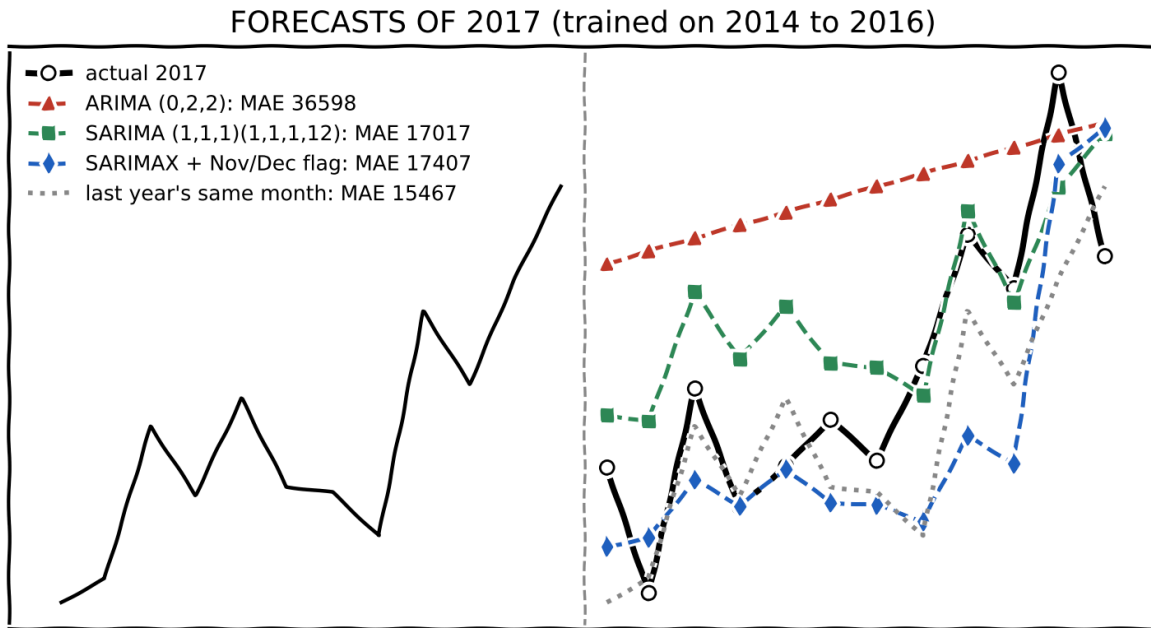
It scores the 2018 forecast against the 2017 actuals. Different months, so the MAE is not an accuracy.

Fair test: hold out 2017

- Train on 2014 to 2016 (36 months), forecast 2017, compare with 2017.

Model	MAE
Repeat last year's same month	15468
ARIMA (0,2,2), the grid pick	36598
SARIMA (1,1,1)(1,1,1,12)	17017
Season only, no flag	18221
Same plus Nov/Dec flag (X)	17408

- The ARIMA line is flat-ish and misses the seasonal swings, so it scores worst.
- The Nov/Dec flag lowers MAE from 18221 to 17408. The flag adds about 79246 to sales in those months.
- Repeating last year beats SARIMA here. With only 36 training months, the seasonal terms have little to learn from. Always compare against this simple baseline.



11. Lab code, step by step

- The code uses the lab's libraries: pandas, numpy, matplotlib, statsmodels, scikit-learn. It holds out 2017 for a fair score.

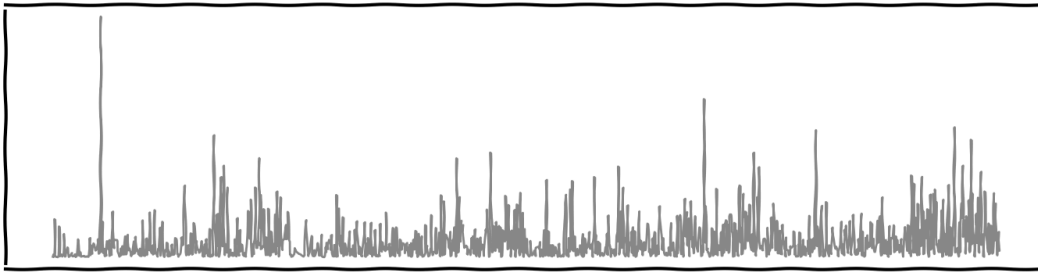
Load and add up per month

```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
url = ("https://raw.githubusercontent.com/trdsaputri/sample/"
      "refs/heads/main/store_dataset.csv")
data_df = pd.read_csv(url)          # 9994 orders (one row per item)
# keep 2 columns, turn the date text into real dates
sales_df = data_df[['Order Date', 'Sales']].copy()
sales_df['Order Date'] = pd.to_datetime(sales_df['Order Date'])
# many orders per day: add them up per day, in date order
daily_sales = (sales_df.groupby('Order Date')['Sales']
               .sum().sort_index())
# daily is jumpy, so add up per month ('ME' = month end)
df1 = sales_df.set_index('Order Date')
monthly_sales = df1.resample('ME').sum()    # 48 months
monthly_sales['Sales'].plot(title="Monthly Sales")
plt.show()
```

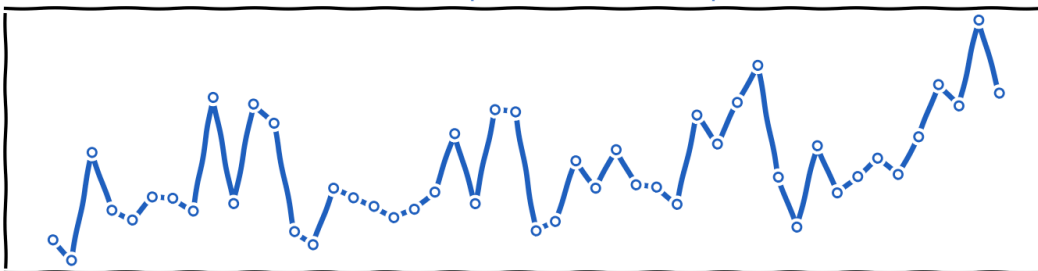
- `read_csv` loads the file. `to_datetime` turns text dates into real dates, which `resample` needs.
- `groupby` then `sum` adds many orders on one day. The daily plot is jumpy, so we `resample` to months ('ME' = month end).

- Result: 48 rows, 2014-01-31 to 2017-12-31. First month 14236.9.

DAILY SALES: 1237 order days, jumpy



MONTHLY SALES: 48 points, trend and peaks show



Check stationarity, ACF, PACF

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# ADF test: p < 0.05 means "looks stationary"
p_value = adfuller(monthly_sales['Sales'], autolag='AIC')[1]
print(p_value)
print('stationary' if p_value < 0.05 else 'not stationary')
plot_acf(monthly_sales)      # correlation of y_t with y_{t-k}
plot_pacf(monthly_sales)     # same, with shorter lags removed
plt.show()
```

- adfuller returns a tuple. Index 1 is the p-value. Here 0.0002, so stationary.
- plot_acf shows correlation with each past lag. plot_pacf shows each lag alone. Both show the lag 12 bar.

ARIMA: try every order, keep the lowest AIC

```
import itertools
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
sales = monthly_sales['Sales']
# hold out the last 12 months (2017) as the test set
train, test = sales.iloc[:-12], sales.iloc[-12:]
# try every (p, d, q), keep the lowest AIC (lower = better fit)
best_aic, best_order = np.inf, None
for order in itertools.product(range(4), range(3), range(4)):
    try:
        aic = ARIMA(train, order=order).fit().aic
    except Exception:
        continue
    if aic < best_aic:
        best_aic, best_order = aic, order
print(best_order, best_aic)
fc = ARIMA(train, order=best_order).fit().forecast(steps=12)
print(mae(test, fc))    # MAE
print(mse(test, fc))    # MSE
```

- `product(range(4), range(3), range(4))` makes all 48 combinations of p, d, q , like a grid search.
- `.fit()` estimates the model. `.aic` is the score to minimise. `try/except` skips combinations that fail.
- `forecast(steps=12)` predicts the next 12 months. `mae` and `mse` compare with the held-out 2017.
- Output on the 2014 to 2016 training data: (0, 2, 2), AIC 791.77, MAE 36598, MSE $1.58e9$.

SARIMA

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
p, d, q = 1, 1, 1
P, D, Q, s = 1, 1, 1, 12      # s = 12 months in one season
model = SARIMAX(train, order=(p, d, q),
                 seasonal_order=(P, D, Q, s))
fc = model.fit().forecast(steps=12)
print(mae(test, fc))    # MAE
print(mse(test, fc))    # MSE
```

- order is (p, d, q) . seasonal_order is (P, D, Q, s) . $s = 12$ because one season is 12 months.
- Output: MAE 17017, MSE $3.87e8$.

SARIMAX with an outside variable

```
# X: a flag known in advance, 1 for November and December
months = sales.index.month
holiday = pd.DataFrame({'holiday': np.isin(months, [11, 12]) * 1},
                        index=sales.index)
base = dict(order=(1, 0, 0), trend='c',
            seasonal_order=(1, 0, 0, 12))
plain = SARIMAX(train, **base).fit(dispatch=False)
with_x = SARIMAX(train, exog=holiday.loc[train.index],
                 **base).fit(dispatch=False)
f1 = plain.forecast(steps=12)
f2 = with_x.forecast(steps=12, exog=holiday.loc[test.index])
print(mae(test, f1))    # without X
print(mae(test, f2))    # with X
print(with_x.params['holiday']) # extra Nov/Dec sales
```

- holiday is 1 in November and December, else 0. It is known ahead, so it can also be given for the forecast months.
- exog gives the flag for training. The forecast needs exog for the 12 new months too.
- trend='c' adds a constant level. Output: MAE 18221 without the flag, 17408 with it, flag effect 79246.

Seasonal index

```
from statsmodels.tsa.seasonal import seasonal_decompose
dec = seasonal_decompose(sales, model='multiplicative', period=12)
# the 12 factors repeat every year; x 100 gives the seasonal index
month = dec.seasonal.index.month
index = dec.seasonal.groupby(month).first() * 100
print(index.round(0))
```

- seasonal_decompose splits the series into trend, season and leftover. model='multiplicative' means they multiply, so the season is a ratio.
- dec.seasonal repeats every 12 months. groupby month then first takes one value per month. Multiply by 100 for the index.
- Output: 56, 39, 105, 77, 89, 77, 78, 73, 167, 89, 174, 175.

W3 quick recall

- Time series = ordered by time. Never shuffle. Split by time.
- ACF at lag k = correlation of y_t with y_{t-k} . Lag 12 high means a yearly season.
- Stationary = constant mean, constant variance, covariance depends on lag only. Difference to fix. ADF $p < 0.05$ means stationary.
- ARIMA(p,d,q): p = past values, d = differences, q = past errors. SARIMA adds (P,D,Q,s). SARIMAX adds outside variables.
- Seasonal index = period average / overall average $\times 100$. 100 is average.

- Lab: SARIMA beat ARIMA. Score on months the model never saw.

Overall recall

- W1: ML learns rules from data. Tasks: association, classification, regression, clustering.
- W2: Regression predicts numbers. Fit by minimising squared error. Curves via polynomial features. Regularise to avoid overfitting.
- W3: Time series is data in time order. Check stationarity and difference if needed. ARIMA has p, d, q. SARIMA adds the season gap s. Seasonal index = period average / overall average $\times 100$. Split by time, never shuffle.