

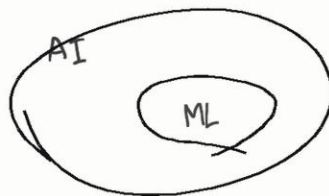
01 - intro to ML

what is 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
 - e.g. weather forecasting app: experience = historical weather data, task = predicting today's weather, perf. = rains / not
- real world: face recognition, spam filters in email, recommendations (Netflix), fraud detection (PayPal), gen AI (GPT)

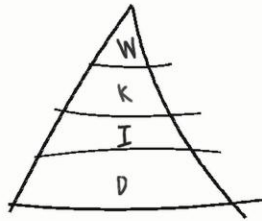
AI vs ML

- 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 that's processed / 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 / observations. most abundant, least valued (database of timestamps & product codes)
- I: data categorized, calculated / 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)

why it's important

- 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 way VS ML way

- 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
 - e.g. 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 of ML

supervised learning

- every example has the correct label (photos tagged "cat" / "dog")
- classification: predict a specific class label from predefined data. "which one is it?" / "what kind is it?"
- determining if a financial profile falls to bad / fair / 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
- regression: predict a continuous real number. "how much?" / "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 learning

- 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 / things, organize to distinct segments based on how similar they are
 - "which of these things belong together in a group?" e.g. "young parents" // "health eaters"
 - also: similar pictures, categorizing news
- association (connecting items, rules): finds specific triggers / links between items / events that freq. happen at the same time
 - "if X happens, what else is likely to happen?" e.g. if buy A, 70% chance buy B
 - $\approx$ basket analysis
- dimensionality reduction $\rightarrow$ squeezes many features down to fewer
 - e.g. dataset of students, each student 50 features (scores in 50 subjects)
 - most of the features move together (if good at calculus, also good at algebra 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 / 3D"

other learning types

type	answer key	example
supervised	every example has the correct label	photos tagged "cat" / "dog"
unsupervised	no labels, finds path itself	grouping customers by behaviour
semi-supervised	only some examples have labels	100 tagged photos + 10,000 untagged
reinforcement	~~no labels, only rewards or penalties~~	game AI gets +1 for winning, -1 for losing

- 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 association notation

- "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 we're predicting
- python libs: NumPy (arrays), SciPy (numerical algorithms), matplotlib (plots), pandas (tables), scikit-learn (ML algorithms). no notebook this week

02 - regression

what is regression?

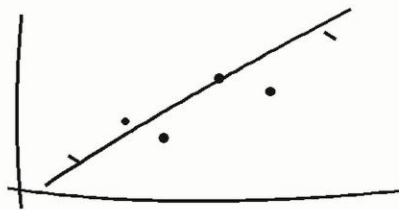
- Supervised. Predicts a continuous value from one or more variables.
- X = independent variables = features. Y = dependent variable = target.
 - Boston data: the 13 columns CRIM to LSTAT are X. House price is Y.
- Uses: wind speed from temperature and humidity. House price from rooms and land size. Stock price. Car MPG from weight and horse power.

simple linear Regression

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

$\hat{y}$ predicted variable. θ_0 intercept. θ_1 slope. x_1 the single predictor.

- Training = find the θ whose line fits the points best.



how good is the model?

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

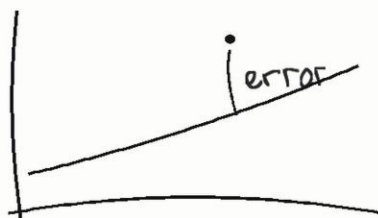
- Real $y = 16.5$, predicted $\hat{y} = 15$. Error = 1.5.

$$\text{MSE} = (1/n) \sum (y - \hat{y})^2$$

- Squaring: plus and minus errors cannot cancel, and big errors are punished more.

- Objective: find the θ that minimise the MSE.

- Also: MAE = mean of $|y - \hat{y}|$. R^2 = share of variance explained, 1 is perfect.



least squares by hand

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

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

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

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

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

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

- Sum = 0.30. MSE = 0.30 / 4 = 0.075.

multiple linear regression

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

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

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

- Slide example: $\theta = [-4.56, -0.09, -0.56, 5.47]$. Row 8: CRIM 0.21124, LSTAT 29.93, RM 5.631.

$$\text{price} = -4.56 - 0.019 - 16.46 + 30.80 = 9.76$$

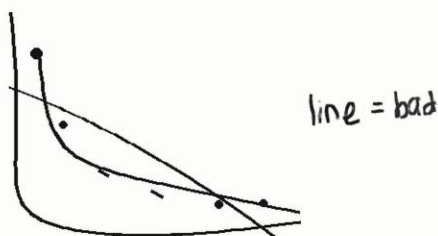
- Real price 16.5, so the error is 6.74.

- Signs: more crime or more LSTAT $\rightarrow$ lower price. More rooms $\rightarrow$ higher price.

non-linear regression

- Use it when a straight line cannot fit. Price vs crime drops fast, then flattens.

- Shapes: linear, quadratic, cubic, polynomial.



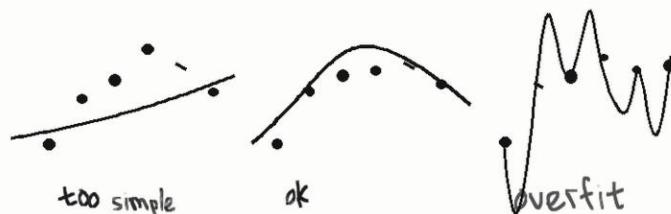
polynomial regression

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

- Trick: rename $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$. Now it is multiple linear regression, solved by least squares.

- Goal: minimise the sum of squared residuals.

- Overfitting: degree too high. The model memorises the training points and fails on new ones. Train error falls, test error rises.



actual non-linear

- Non-linear in θ . Cannot be renamed into a line. Solved by iterating, no closed formula.

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

$$\hat{y} = \log(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$\hat{y} = \theta_0 / (1 + \theta_1 (x - \theta_2)) \text{ S-curve}$$

Regularization

- Prevents overfitting by penalising large coefficients.
- Ridge: squared penalty. Shrinks all coefficients. Use when many features are highly correlated.
- Lasso: forces some coefficients to exactly 0, so it selects features. Use when only a few features matter.
 Ridge: $MSE + \alpha(\theta_1^2 + \theta_2^2 + \dots)$
 Lasso: $MSE + \alpha(|\theta_1| + |\theta_2| + \dots)$
- α = penalty strength. $\alpha = 0$ is plain regression. Pick α with GridSearchCV.
- CAREFUL: the slide says Ridge (L1) and Lasso (L2). The standard is Ridge = L2, Lasso = L1.

lab code (Boston)

- This is the notebook's 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.

1. Load the data

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

2. Look at the data

```
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.

3. Split into train and test

```
# 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.

4. Simple linear regression (1 feature)

```
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` 6.92, `MSE` 63.20.

6. Multiple linear regression (3 features)

```

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

```

- Lab output: `coefs` -0.094, -0.605, 5.07. `Intercept` -1.62. "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.

6. Polynomial regression

```

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  $\theta$ , 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).

```

7. 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 -6.55. 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_.

8. 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 7.39, MSE 98.32.
- CAREFUL: same mistake. The search picked $1e-15$, the lab trained with alpha = 1.
- CAREFUL: -81.90 against Ridge's -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.

03 - time series

what is a 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.
- Univariate: time is the only independent variable. Monthly sales alone.
- Multivariate: more variables. Sales plus weather.
- Uses: forecasting (next month's sales), anomaly detection (odd network activity), pattern analysis (weekend peaks).

autocorrelation

- How a series relates to its own past. Lag $k = k$ steps back.
- ACF at lag k = correlation between $y(t)$ and $y(t-k)$.
- A tall bar at lag 12 = a yearly season.
- PACF = the same, with the shorter lags removed.
- By hand: 12, 16, 14, 18, 20, 19, 23. Mean 17.29.
 - Deviations: -5.29, -2.29, -3.29, 0.71, 2.71, 1.71, 5.71.
 - Top = sum of $\text{dev}(t) \times \text{dev}(t-1) = 33.63$. Bottom = sum of $\text{dev}^2 = 87.43$.
 - $\text{ACF}(1) = 33.63 / 87.43 = 0.386$.

trend vs seasonality vs cycle

- Trend: long-term rise or fall. Sales grow as the customer base grows.
- Seasonality: repeats at a fixed gap. Electricity up every evening. Ice cream up every summer.
- Cycle: rises and falls, gap not fixed. Economic booms and recessions.
- Test: can I say how many steps until it repeats? Yes = season. No = cycle.



stationarity

- The statistics stay stable over time.
- 3 conditions: constant mean. Constant variance. Covariance depends on the lag, not the time.
 - Use that climbs every month: not stationary. Use that wobbles around one average: stationary.
- Fix a trend by differencing: $y(t) - y(t-1)$. Once is $d = 1$.

12, 15, 14, 18, 20, 19, 23 $\rightarrow +3, -1, +4, +2, -1, +4$.
 ADF test $p < 0.05$ means stationary.



ARIMA(p, d, q)

- Autoregressive Integrated Moving Average. Forecasts from past values and past errors.
- p = autoregressive order: number of lags (past values).
- d = differencing: times differenced to become stationary. Usually 1.
- q = moving average order: number of past errors.
 - ARIMA(1,1,1): 1 past value, differenced once, 1 past error.
- The lab tries many orders and keeps the lowest AIC (fit, with a penalty for extra parameters).
- Steps: visualise, ACF and PACF plots, seasonal check, stationary check, pick p, d, q .

SARIMA and SARIMAX

- SARIMA(p, d, q)(P, D, Q, s): ARIMA plus a seasonal copy. s = season length, 12 for monthly data.
 - P seasonal AR order. D seasonal differencing. Q seasonal MA order.
 - Seasonal differencing: $y(t) - y(t-12)$.
- SARIMAX: SARIMA plus X , outside (exogenous) variables. A holiday flag.
 - S seasonal. AR past values. I differencing. MA past errors. X external variables.

seasonal index

- How strong a period is against an average period. 100 = average.
- Index = period average / overall mean $\times 100$.
- Sales Y1: 20, 30, 25, 45. Y2: 24, 35, 28, 52. Y3: 28, 40, 31, 59.

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

- Overall mean 34.75. Deseasonalise Q4 of Y3: 59 / 1.4964 = 39.43.

lab code (store sales)

- This is the notebook's own code. Lines marked CAREFUL are mistakes or traps in it.

1. Load the data

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

# 9994 rows, one per item ordered
url
    = "https://raw.githubusercontent.com/trdsaputri/sample/refs/heads/main/store_data"
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().

2. Daily, then 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, 5), 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' = month end. Resample needs a date index.

3. 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.

4. ARIMA: try every order, keep the lowest AIC

```
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.  $4 \times 3 \times 4 = 48$ 
pdq = list(itertools.product(p, d, q))
```

```
best_aic = np.inf
best_order = None
best_model = None
```

```
for order in pdq:
    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)
```

- Lab output: best order (3, 2, 3), AIC 1066.76, 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.

6. Plot and 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:].

6. 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 1.52e8. 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.

7. 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.