Prepared by NUS SDS

Machines That Learn

Dive into the world of Machine Learning - from intuition to implementation

Team Members

Daeren

Y2 DSA

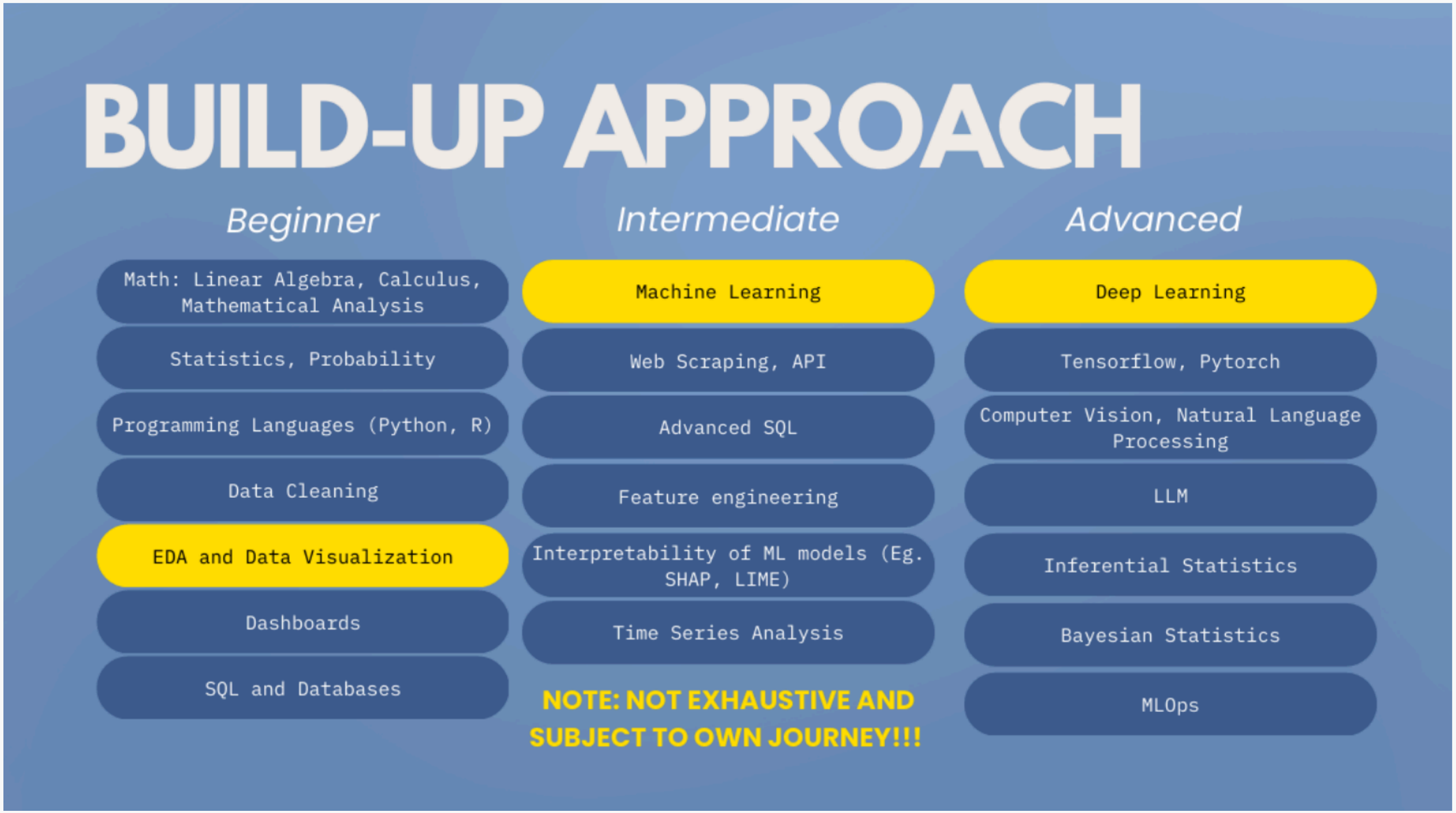
Clarence

Y2 DSE

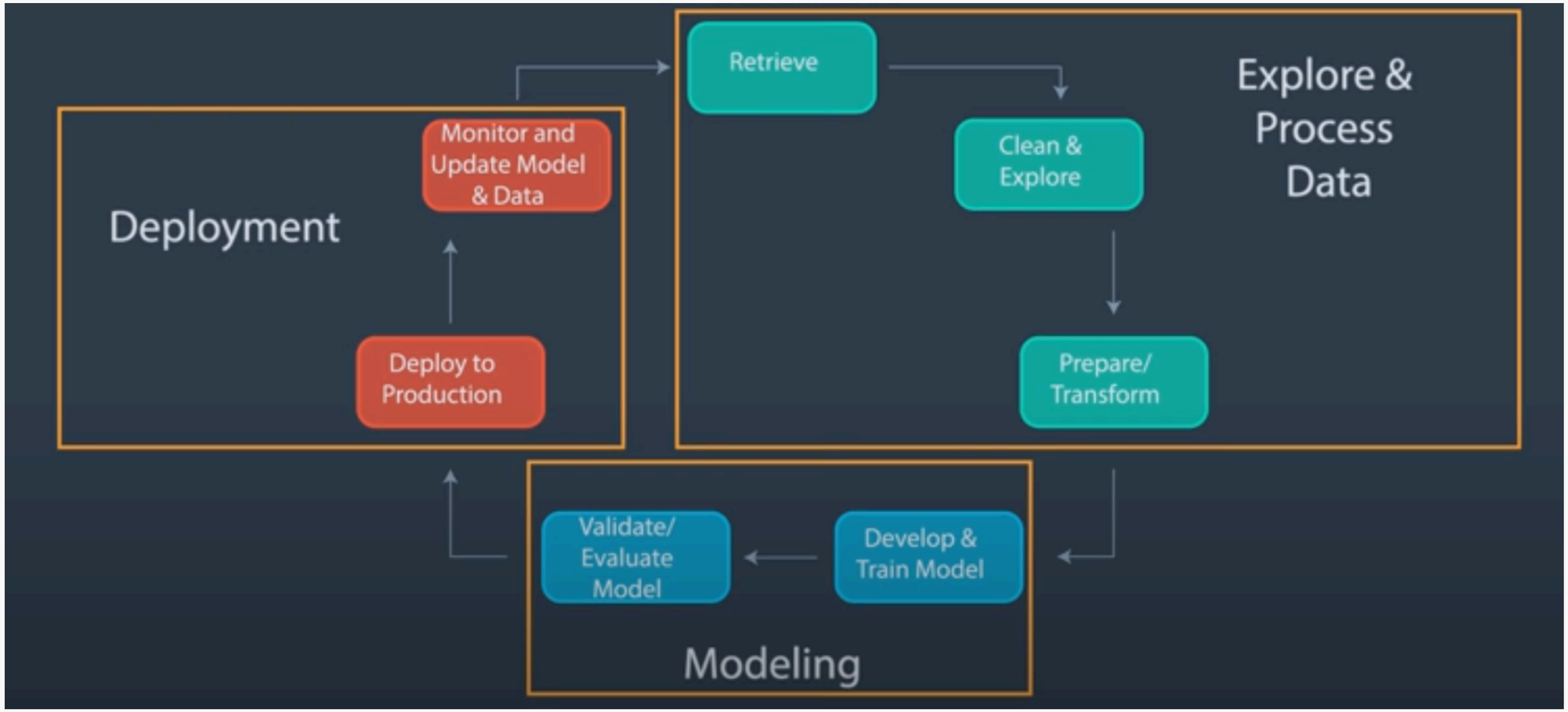
Xiang Ting

Y3 DSE

Roadmap


BUILD-UP APPROACH
Beginner
Math: Linear Algebra, Calculus, Mathematical Analysis
Statistics, Probability
Programming Languages (Python, R)
Data Cleaning
EDA and Data Visualization
Dashboards
SQL and Databases
Intermediate
Machine Learning
Web Scraping, API
Advanced SQL
Feature engineering
Interpretability of ML models (Eg. SHAP, LIME)
Time Series Analysis
NOTE: NOT EXHAUSTIVE AND SUBJECT TO OWN JOURNEY!!!
Advanced
Deep Learning
Tensorflow, Pytorch
Computer Vision, Natural Language Processing
LLM
Inferential Statistics
Bayesian Statistics
MLOps

Machine Learning Workflow



Main Focus Today!

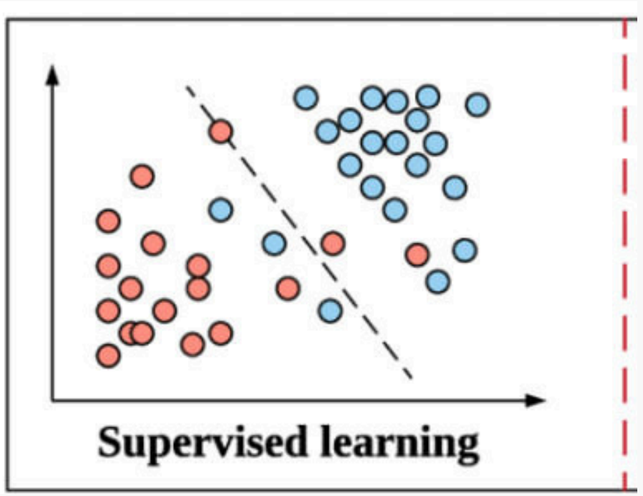
Intro to Machine Learning

Types of Machine Learning

I. *Supervised Learning*

- Definition: Model learns from labelled data (input-output pairs) to predict outputs for unseen inputs
- Examples:
  - Predicting house prices from features (regression)
  - Classifying emails as spam vs not spam (classification)
- Key algorithms: Linear Regression, Logistic Regression, Decision Trees, Neural Networks

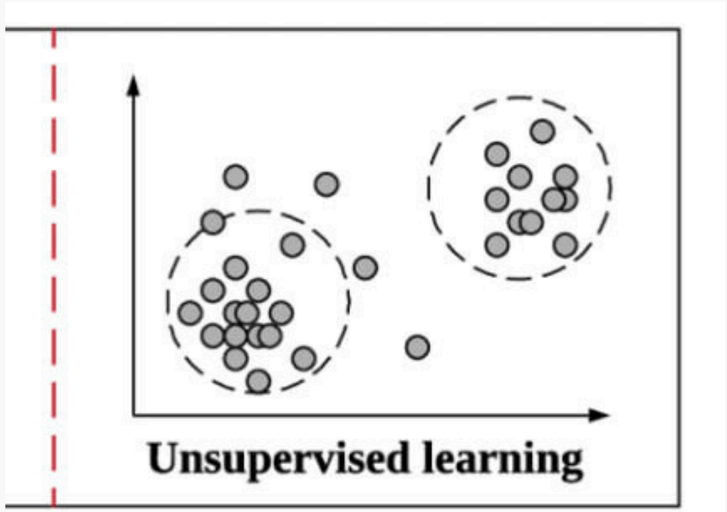
$$f : X \rightarrow Y$$



Types of Machine Learning

2. Unsupervised Learning

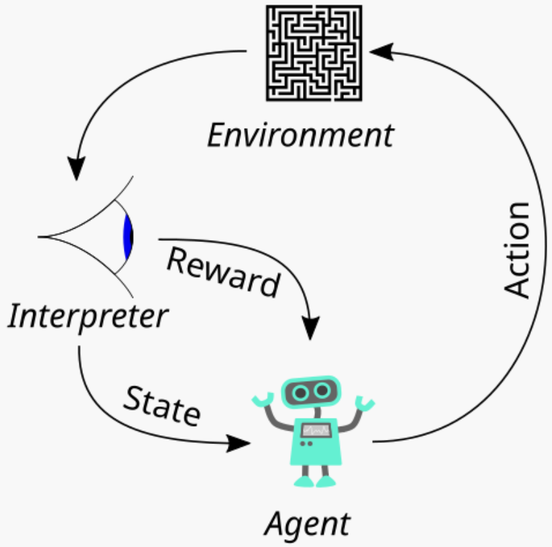
- Definition: data has inputs but no labels → model tries to discover hidden patterns
- Examples:
  - Customer segmentation (clustering)
  - Dimensionality reduction (PCA)
- Key algorithms: k-Means, Hierarchical clustering, PCA



Types of Machine Learning

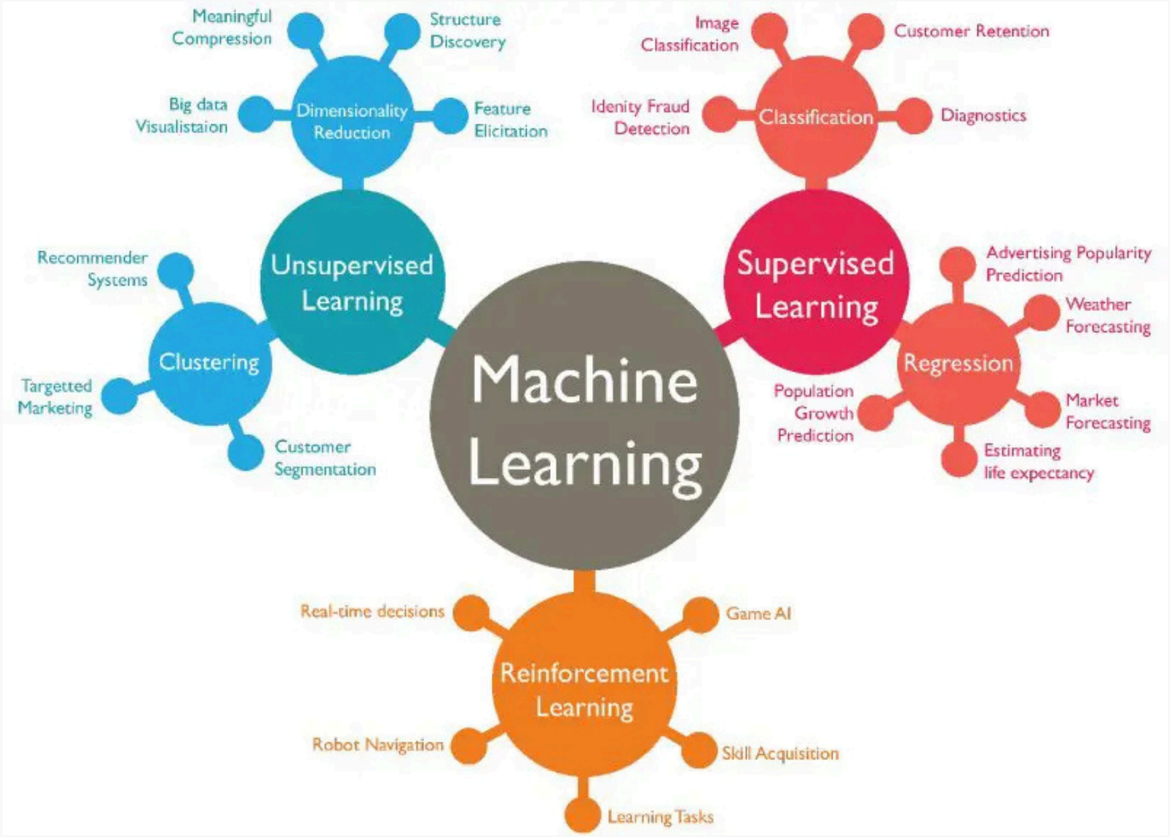
3. Reinforcement Learning

- Definition: learn by interacting with an environment, receiving rewards (positive feedback) or penalties (negative feedback)
- Examples:
  - Game playing (Chess)
  - Adaptive systems (personalized recommendations)
- Key algorithms: Q-learning, Deep Q-Networks, Policy Gradients



Types of Machine Learning

Supervised / Unsupervised / Reinforcement = most common family tree

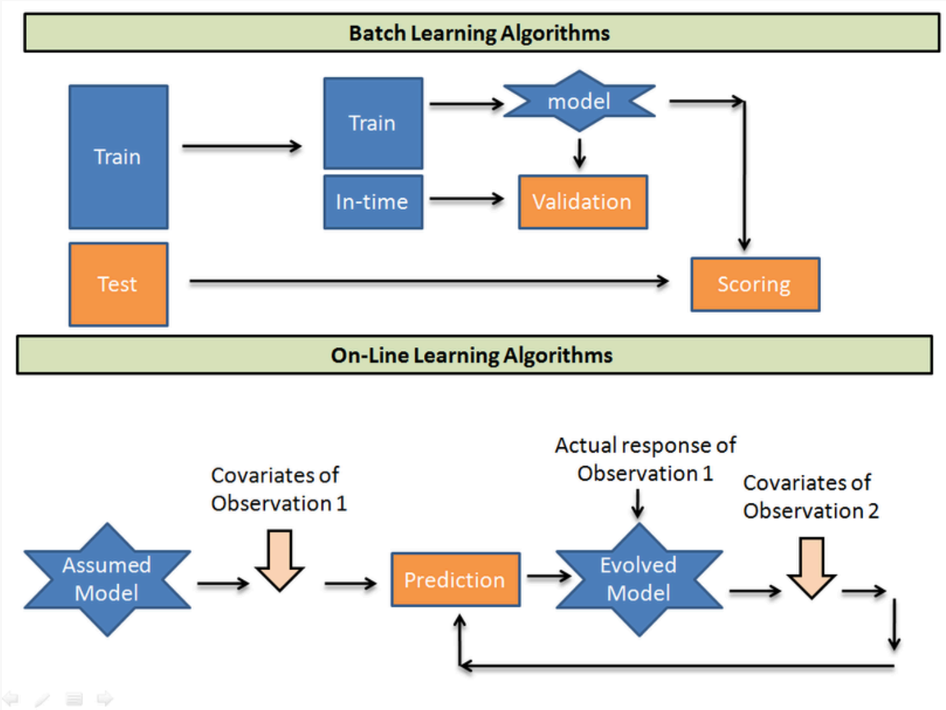


BUT: ML can also be classified by how models learn / generalise

Classification of ML Models

ML by Learning Processes

- Batch vs Online Learning := Whether they can learn incrementally on the fly
  - Batch → trained once on fixed data (Credit scoring at a bank)
  - Online → updates continuously as data streams in (Predicting news article recommendations)



Error Metrics

Regression vs Classification

Regression	Classification
• Predicts continuous values (e.g., house prices, height, exam scores)	• Assign input data into predefined categories accurately (e.g., approve/deny a loan, spam/not spam)

Different Metrics for Different Tasks

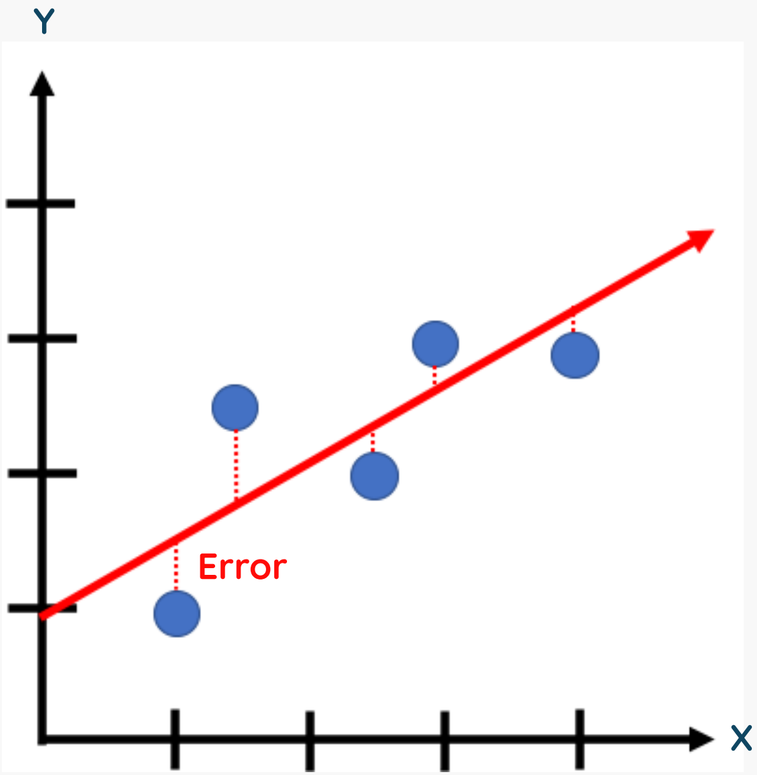
Regression	Classification
• Mean Squared Error (MSE) • Root Mean Squared Error (RMSE) • Mean Absolute Error (MAE)	• Sensitivity and Specificity • Accuracy and Misclassification Rate • Recall (Sensitivity) and Precision • F1-Score

Regression Metrics

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \theta_i)^2$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{1}^{N}(Y_i - \theta_i)^2}$$

$$MAE = \frac{1}{N}\sum_{1}^{N}|Y_i - \theta_i|$$



N → number of data points that you are predicting

Y → Actual value of the data point

θ → Your predicted Value

(Y - θ**)** → Also known as residuals or error

Regression Metrics

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \theta_i)^2$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{1}^{N}(Y_i - \theta_i)^2}$$

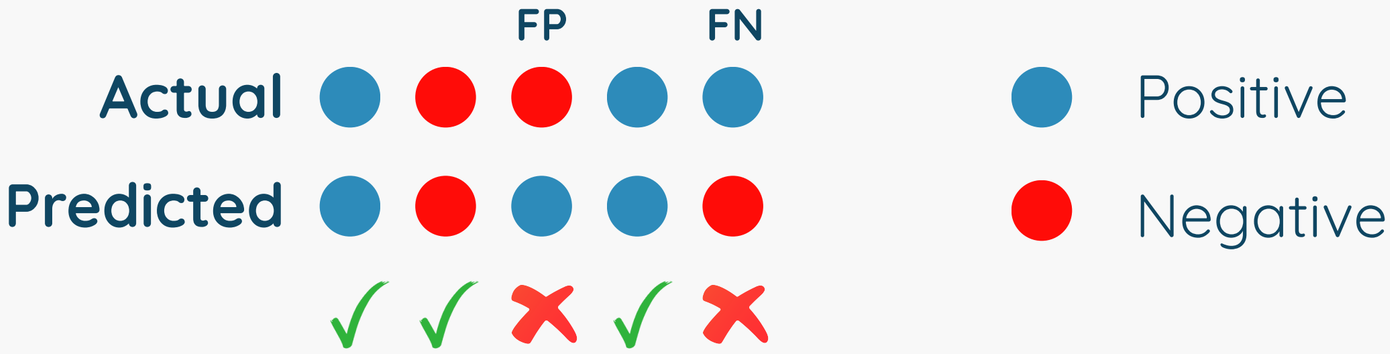
$$MAE = \frac{1}{N}\sum_{1}^{N}|X_i - \theta_i|$$

MSE	• You want to penalise large errors and outliers heavily. Squaring large errors makes them larger! • Easily Differentiable → Used in Gradient Descent Algorithm (Stay tuned for SLR slide!)
RMSE	• You want to penalise large errors and outliers heavily • You want a more interpretable scale • Imagine you are predicting distances. Using MSE gives errors in squared units (eg. km^2), while RMSE converts it back to the original units (km)
MAE	• You care about the average size of errors more than punishing large errors. • If the data set has many outliers, MAE presents a robust metric to use. • More difficult to differentiate due to the absolute notation.

Confusion Matrix

	Actual Positive	Actual Negative
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Confusion Matrix



	<u>Actual Positive</u> = 3	<u>Actual Negative</u> = 2
<u>Predicted Positive</u> = 3	True Positive = 2	False Positive = 1
<u>Predicted Negative</u> = 2	False Negative = 1	True Negative =1

Sensitivity and Specificity

$$Sensitivity = \frac{TP}{TP + FN} = \frac{TP}{ActuallyPostive}$$

$$Specificity = \frac{TN}{TN + FP} = \frac{TN}{ActuallyNegative}$$

	Actual Positive	Actual Negative
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Use Cases Depend on Context

Sensitivity: Important when a high number of false negative is costly

- Example: Medical Diagnostic Testing
- Identifying malignant regions as benign is extremely costly in cancer diagnosis & treatment

Specificity: Important when assessing the proportion of negatives correctly classified

- Example: Drug test for employees/ athletes
- You want to maximise the proportion of negative (clean) individuals correctly classified

Recall and Precision

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{ActuallyPostive}$$

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{PredictedPostive}$$

	Actual Positive	Actual Negative
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Use Cases Depend on Context

Recall: Important when a high number of false negative is costly

- Recall represents the same concept as Sensitivity
- Recall is commonly referred to in ML, while Sensitivity is used in Medicine and Biology

Precision: Important when a high number of false positive is costly

- Email Spam Filtering
- Classifying non-spam email as spam is costly as you might miss out on important emails

F1-Score

What if I want a blended mix of both Precision and Recall?

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In most real-world contexts, we will want to effectively avoid False Positives (Precision) while also avoiding False Negatives (Recall).

F1-score is a harmonic mean of Precision and Recall

- Punishes extreme values of Precision and Recall
- High F1-score means you have done well in avoiding False Positives and False Negatives.

Use Cases:

- **Example:** Object Detection in Computer Vision
- Computer Vision: Accurate identify objects using Machine Learning (Deep Learning).
  - A good model should have a low rate of classifying an apple as not an apple (False Negative)
  - A good model should have a low rate of classifying a banana as an apple (False Positive)

F1-Score

What if I want a blended mix of both Precision and Recall?

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In most real-world contexts, we will want to effectively avoid False Positives (Precision) while also avoiding False Negatives (Recall).

Worked Example

	Actual Positive =10	**Actual Negative** =90
Predicted Positive = 5	True Positive = 4	False Positive = 1
Predicted Negative = 95	False Negative = 6	True Negative = 89

$$Recall = \frac{4}{10} = 0.4$$

$$Precision = \frac{4}{5} = 0.8$$

$$F1 = \frac{2 \times 0.4 \times 0.8}{0.4 + 0.8} = 0.53\,(2d.\,p)$$

F1-Score

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Why not just use F1-score for everything?

- Model that uses F1-score as a metric tries to balance precision and recall
- The cost of False Positives and the cost of False Negatives are treated equally

Bring back the email spam classification example:

- False positives → Classifying a non-spam as spam (Very Costly)
- False negatives → Classifying a spam as non-spam (Less Costly)
- False positives are more costly (furious customer) than False negatives (annoyed customer)

What happens if I use F1-score?

- Start with a model that has high precision and low recall
- F1-score is a harmonic mean between recall and precision.
- To increase recall, the model will start to label more emails as spam
- No model is perfect → it will catch more non-spam as spam → Precision suffers
- False Positives are more costly! You get more furious customers

Accuracy and Misclassification Rate

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Misclassification = \frac{FP + FN}{TP + TN + FP + FN} = 1 - Accuracy$$

I have a dataset of 95% Negatives and 5% Positives
If I predict everything to be Negative
Accuracy = 0.95
Is that really good?
Recall = 0 | Precision = undefined

	Actual Positive	Actual Negative
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

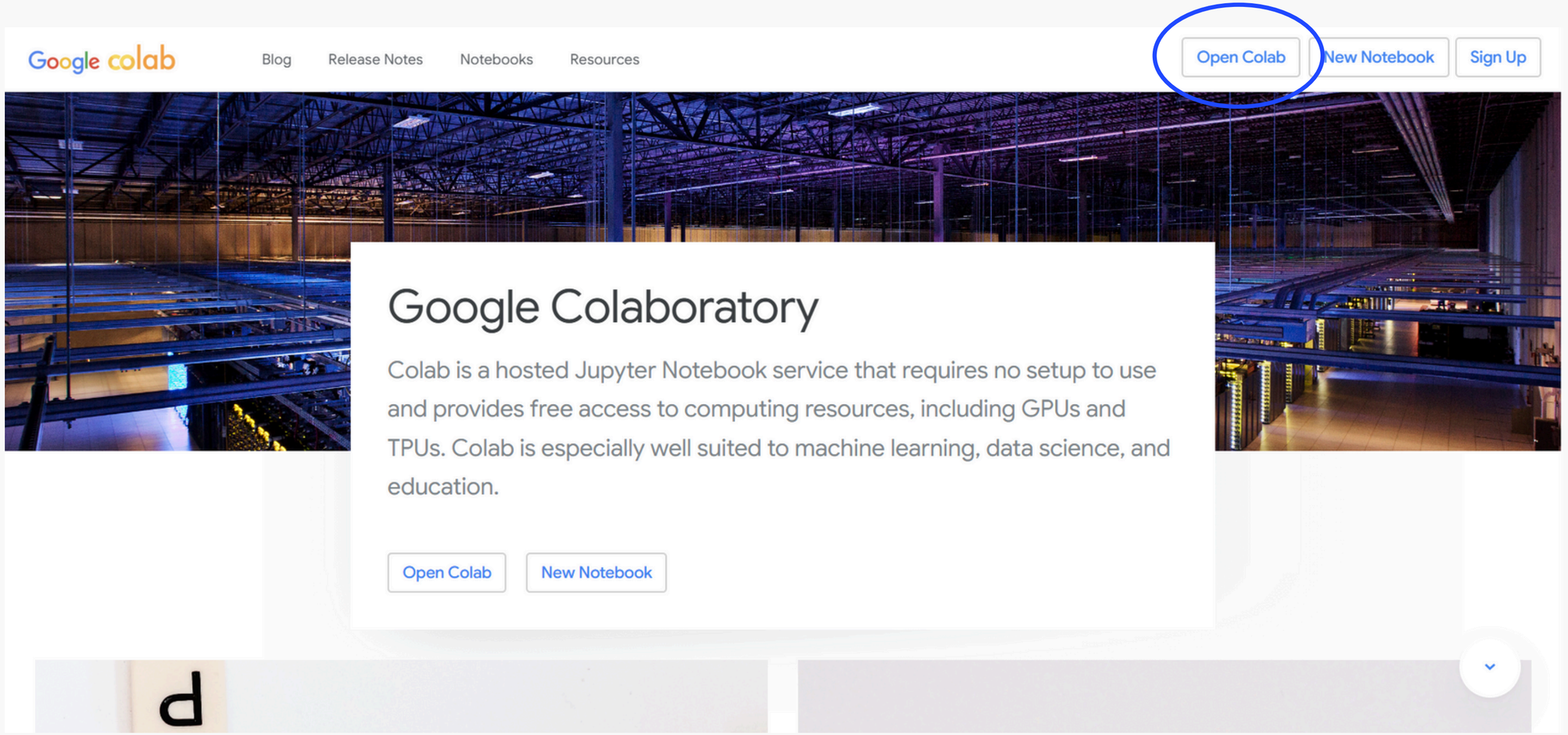
1. Accuracy measures the proportion of correctly classified instances.
2. Misclassification Rate measures the proportion of incorrectly classified instances.

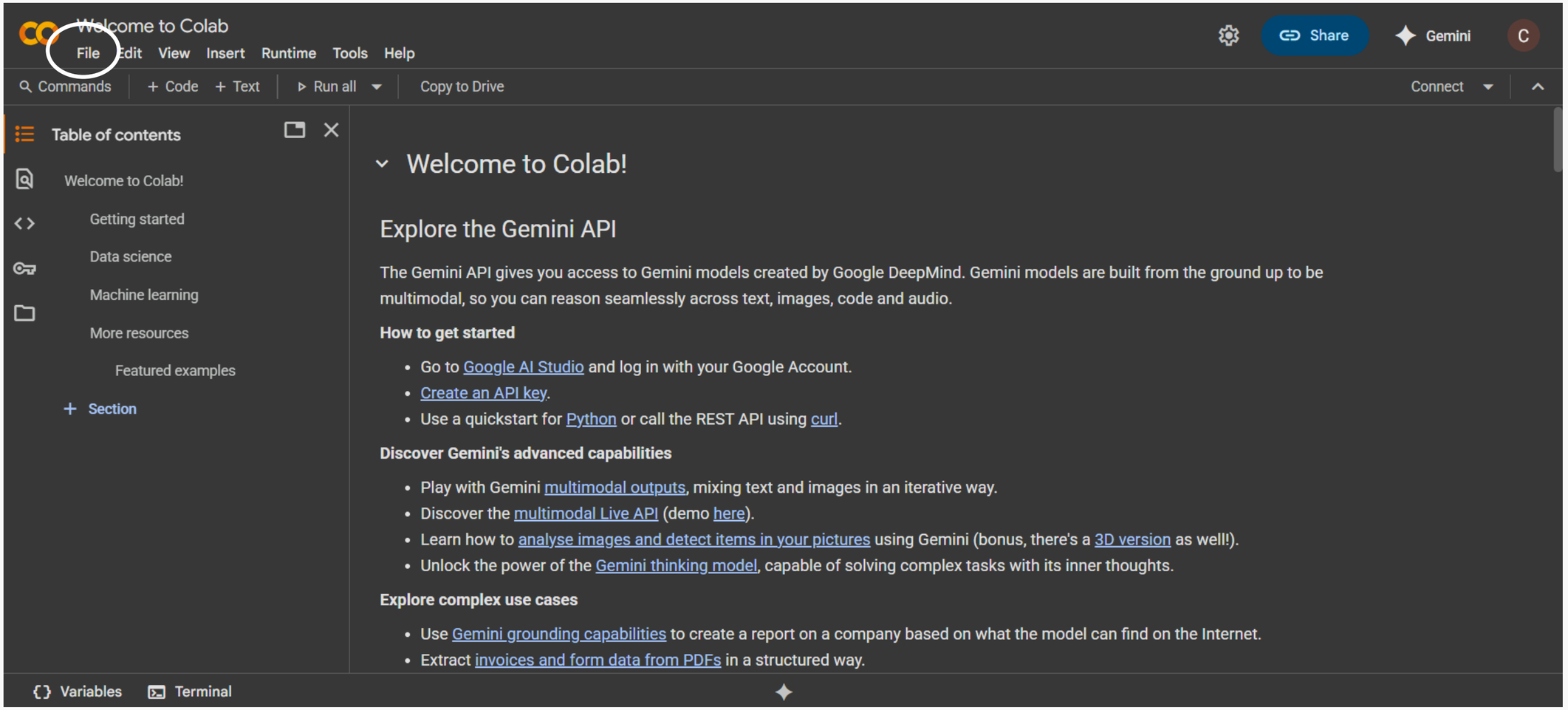
Caveat: Using Accuracy as a sole metric to judge performance may be misleading

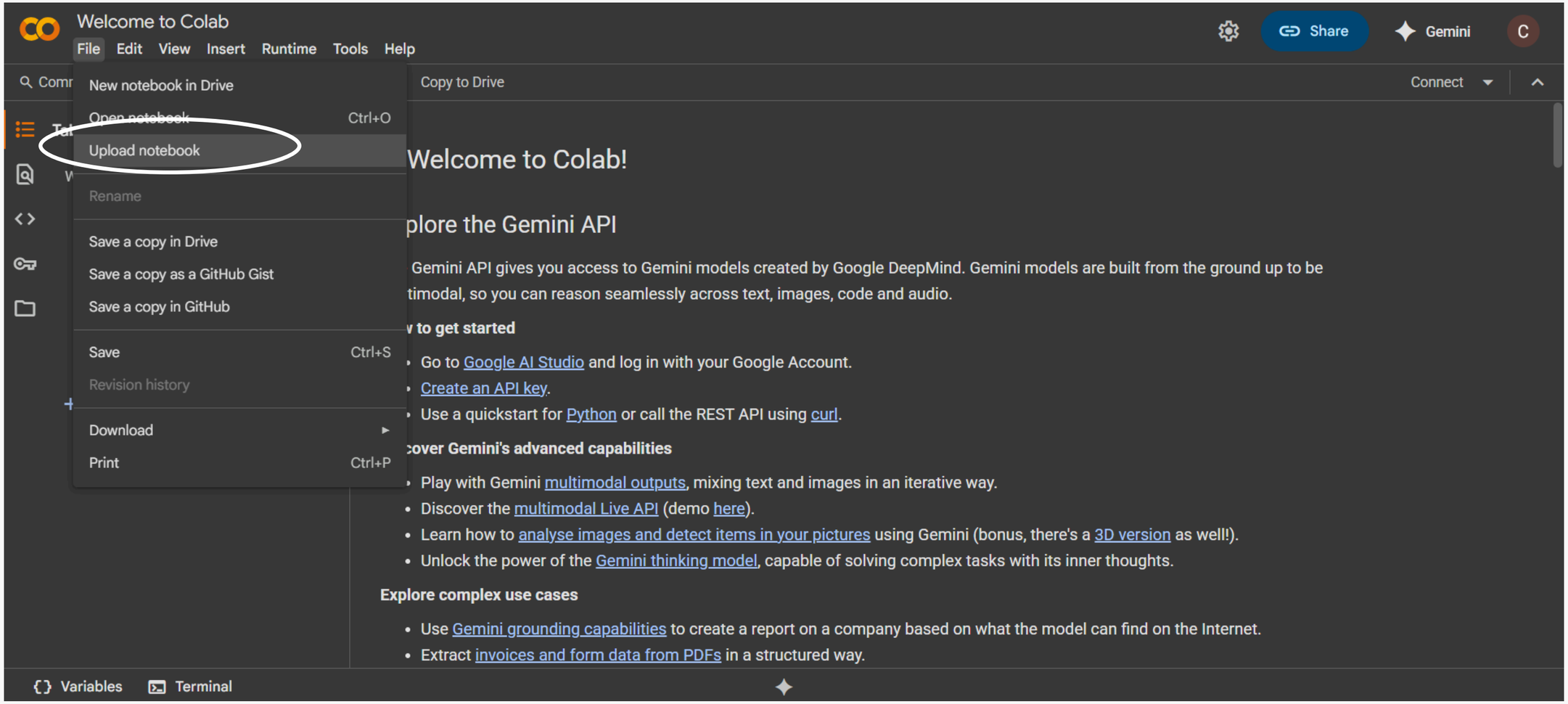
Supervised Learning

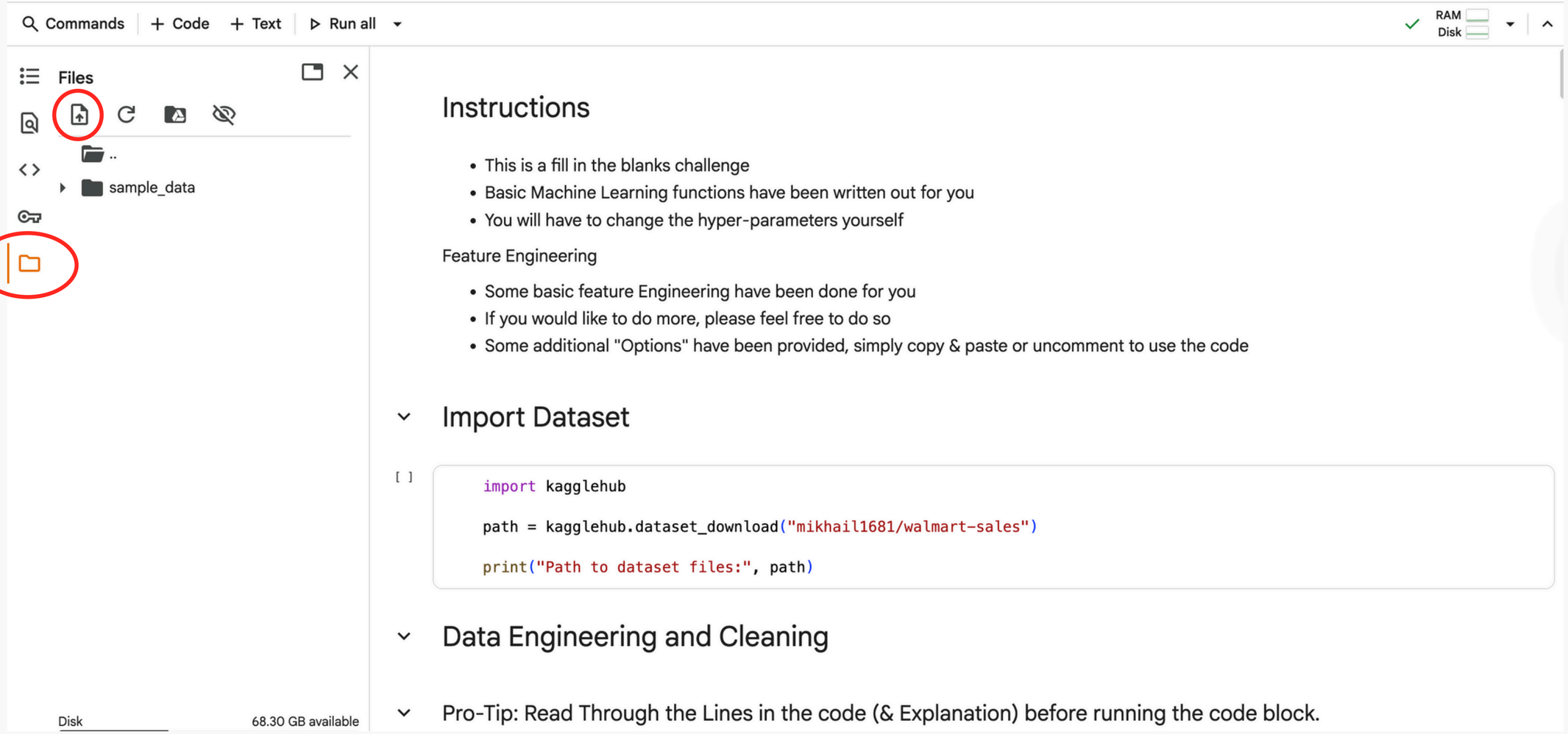
Google Colab Walkthrough

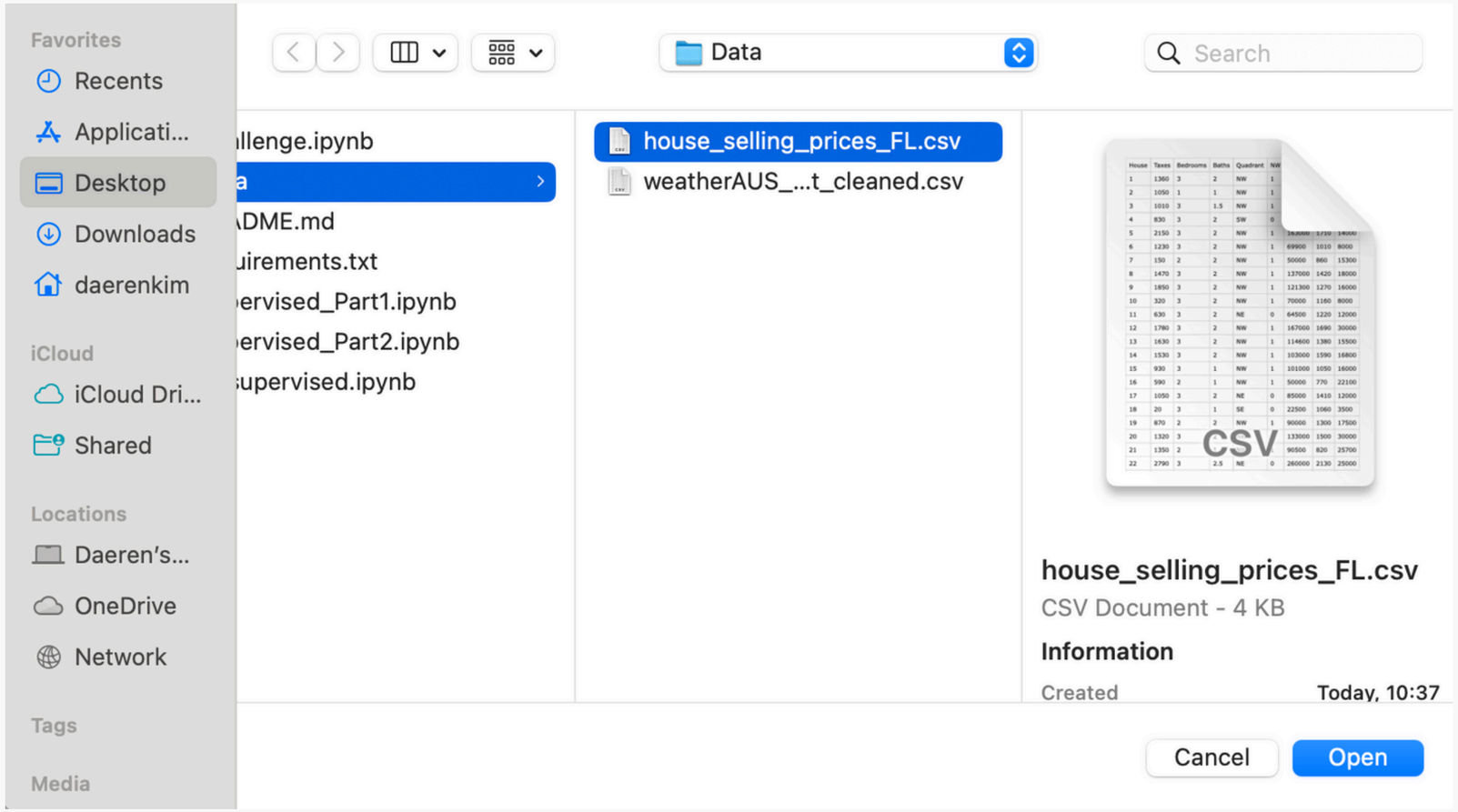
colab.google


Google colab
Blog
Release Notes
Notebooks
Resources
Open Colab
New Notebook
Sign Up
Google Colaboratory
Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
Open Colab
New Notebook

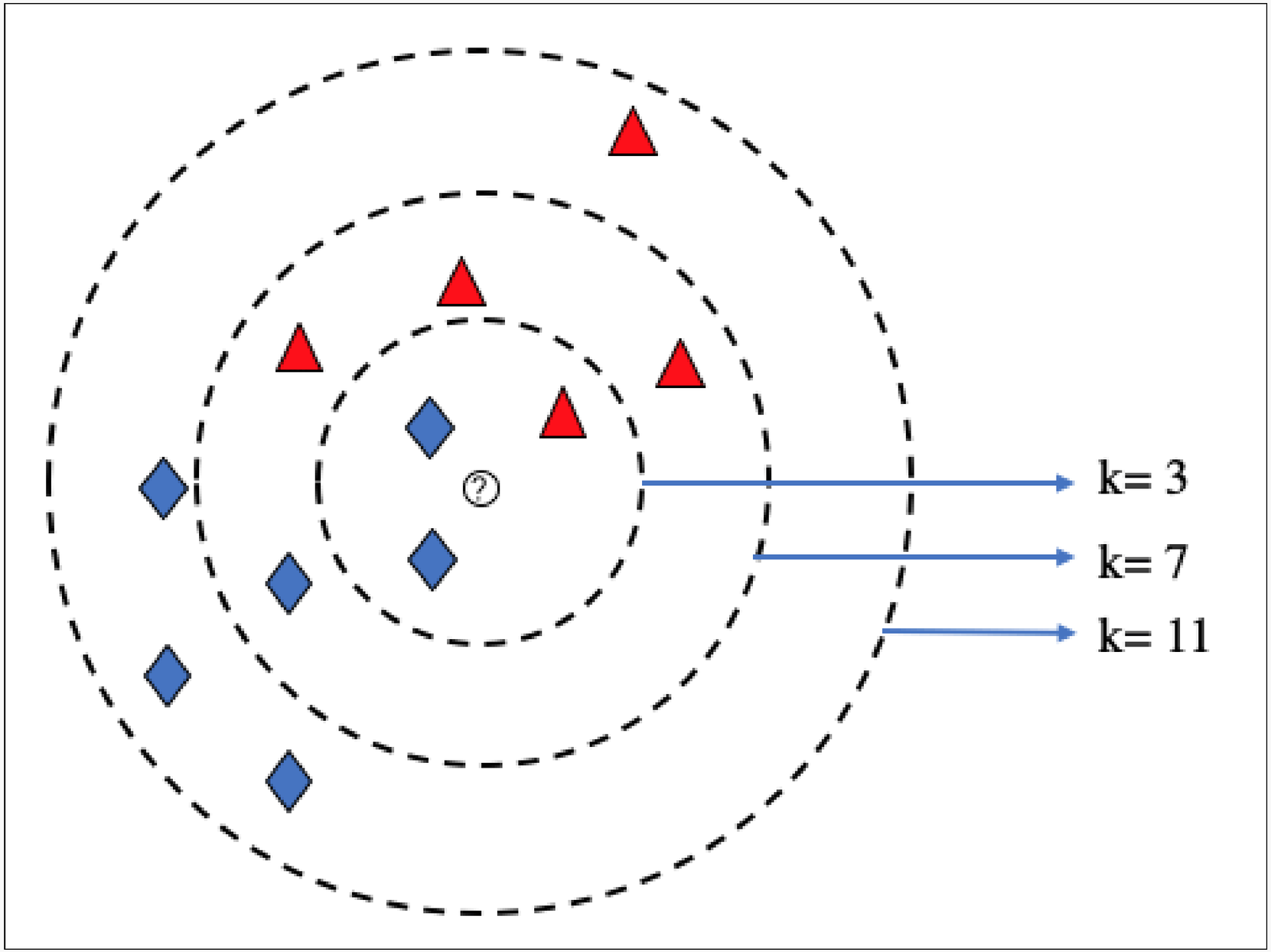

Welcome to Colab
File Edit View Insert Runtime Tools Help
Share
Gemini
C
Commands
+ Code
+ Text
Run all
Copy to Drive
Connect
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More resources
Featured examples
Section
Welcome to Colab!
Explore the Gemini API
The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code and audio.
How to get started
Go to Google AI Studio and log in with your Google Account.
Create an API key.
Use a quickstart for Python or call the REST API using curl.
Discover Gemini's advanced capabilities
Play with Gemini multimodal outputs, mixing text and images in an iterative way.
Discover the multimodal Live API (demo here).
Learn how to analyse images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
Unlock the power of the Gemini thinking model, capable of solving complex tasks with its inner thoughts.
Explore complex use cases
Use Gemini grounding capabilities to create a report on a company based on what the model can find on the Internet.
Extract invoices and form data from PDFs in a structured way.
Variables
Terminal


Welcome to Colab
File Edit View Insert Runtime Tools Help
Share
Gemini
C
New notebook in Drive
Open notebook Ctrl+O
Upload notebook
Rename
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Revision history
Download
Print Ctrl+P
Copy to Drive
Connect
Welcome to Colab!
plore the Gemini API
Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be
timodal, so you can reason seamlessly across text, images, code and audio.
to get started
Go to Google AI Studio and log in with your Google Account.
Create an API key.
Use a quickstart for Python or call the REST API using curl.
over Gemini's advanced capabilities
Play with Gemini multimodal outputs, mixing text and images in an iterative way.
Discover the multimodal Live API (demo here).
Learn how to analyse images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
Unlock the power of the Gemini thinking model, capable of solving complex tasks with its inner thoughts.
Explore complex use cases
Use Gemini grounding capabilities to create a report on a company based on what the model can find on the Internet.
Extract invoices and form data from PDFs in a structured way.
Variables
Terminal


Commands
+ Code
+ Text
Run all
RAM
Disk
Files
..
sample_data
Disk
68.30 GB available
Instructions
This is a fill in the blanks challenge
Basic Machine Learning functions have been written out for you
You will have to change the hyper-parameters yourself
Feature Engineering
Some basic feature Engineering have been done for you
If you would like to do more, please feel free to do so
Some additional "Options" have been provided, simply copy & paste or uncomment to use the code
Import Dataset
[ ]
import kagglehub
path = kagglehub.dataset_download("mikhail1681/walmart-sales")
print("Path to dataset files:", path)
Data Engineering and Cleaning
Pro-Tip: Read Through the Lines in the code (& Explanation) before running the code block.


Favorites
Recents
Applicati...
Desktop
Downloads
daerenkim
iCloud
iCloud Dri...
Shared
Locations
Daeren's...
OneDrive
Network
Tags
Media
Data
Search
llenge.ipynb
a
DME.md
uirements.txt
ervised_Part1.ipynb
ervised_Part2.ipynb
upervised.ipynb
house_selling_prices_FL.csv
weatherAUS_...t_cleaned.csv
CSV
house_selling_prices_FL.csv
CSV Document - 4 KB
Information
Created
Today, 10:37
Cancel
Open

K Nearest Neighbour


?
k= 3
k= 7
k= 11

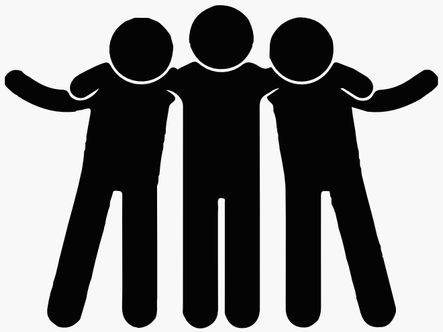
What is KNN?

machine learning method that makes predictions by looking at the k closest data points to the one you're interested in

---

classification → *it chooses the majority label among the k neighbors*

regression → *it takes the average value of the k neighbors*



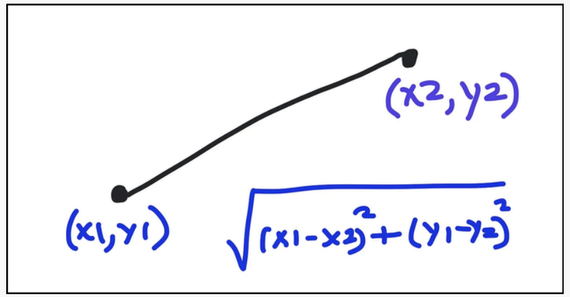
Analogy: Just like how we're influenced by the people around us, KNN's predictions are shaped by nearby data points.

What is KNN?

KNN for Classification

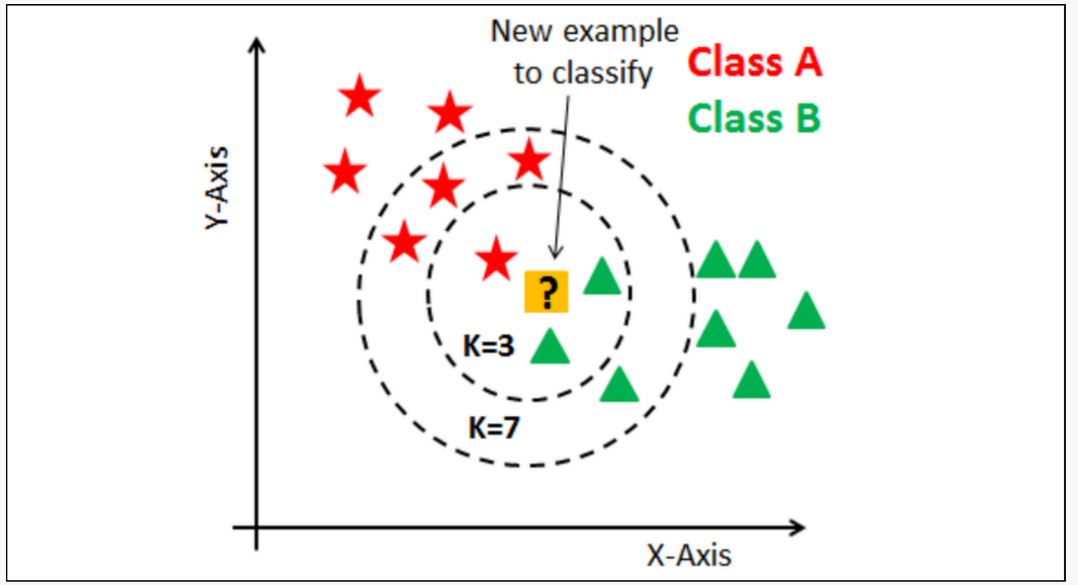
1) Calculate distance (e.g. Euclidean distance)

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$



2) Find k closest neighbors

3) Vote for labels (majority vote)



Avoid even k values to prevent ties

What is KNN?

KNN for Regression

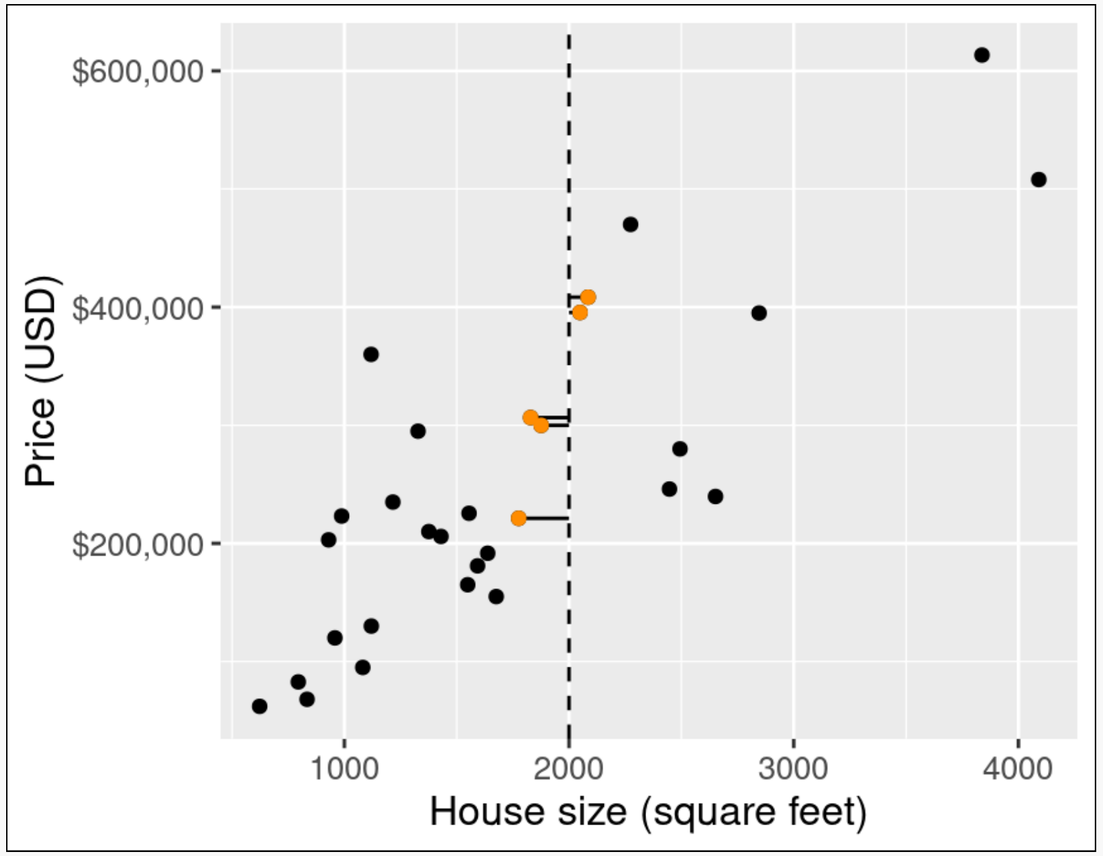
1) Calculate distance (e.g. Euclidean distance)

2) Find k closest neighbors

3) Take the average of the k nearest neighbours

If you want to predict house price, and the 5 nearest houses are priced at 410k, 395k, 305k, 300k and 220k, then KNN regression predicts:

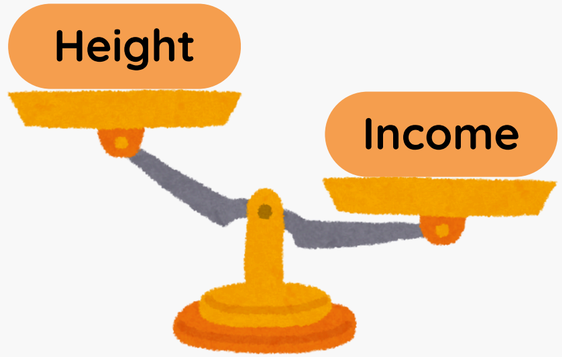
$$\hat{y} = \frac{410 + 395 + 305 + 300 + 220}{5} = 326\text{k}$$



What is KNN?

Since KNN is distance-based, it is important to scale the input features

What is scaling & why?



Suppose we have 2 features:

- Height (in cm): ranges from 150–200
- Income (in $): ranges from 30,000–200,000

If you calculate Euclidean distance directly:

$$\text{distance} = \sqrt{(h_1 - h_2)^2 + (i_1 - i_2)^2}$$

The differences in income will overwhelmingly dominate the distance, making height almost irrelevant

How to scale?

1) Normalisation

$$\text{normalised\_x} = \frac{x - \min(x)}{\max(x) - \min(x))}$$

rescales values so it's between **[0, 1]**

2) Standardisation

$$\text{standardised\_x} = \frac{x - \text{mean}(x)}{\text{sd}(x)}$$

transforms data such that the **mean = 0 and variance = 1**

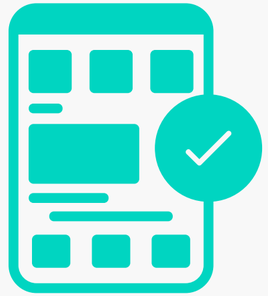
*** Note that scaling only works on numeric features**

Train, Validate, Test

Code Implementation

Training: The model studies and learns patterns, weights, or decision boundaries from data

Validation: The model practices and adjusts: tunes hyperparameters (like choosing k in KNN) without touching the test set, or even choose the ‘best’ model



Test: Used only once at the end to evaluate the final performance of the model on unseen data

Train, Validate, Test

Code Implementation

```
# Split dataset into training and testing sets
# test_size = 0.2 → 20% of data used for testing
# random_state ensures reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

Why split into train and test set?

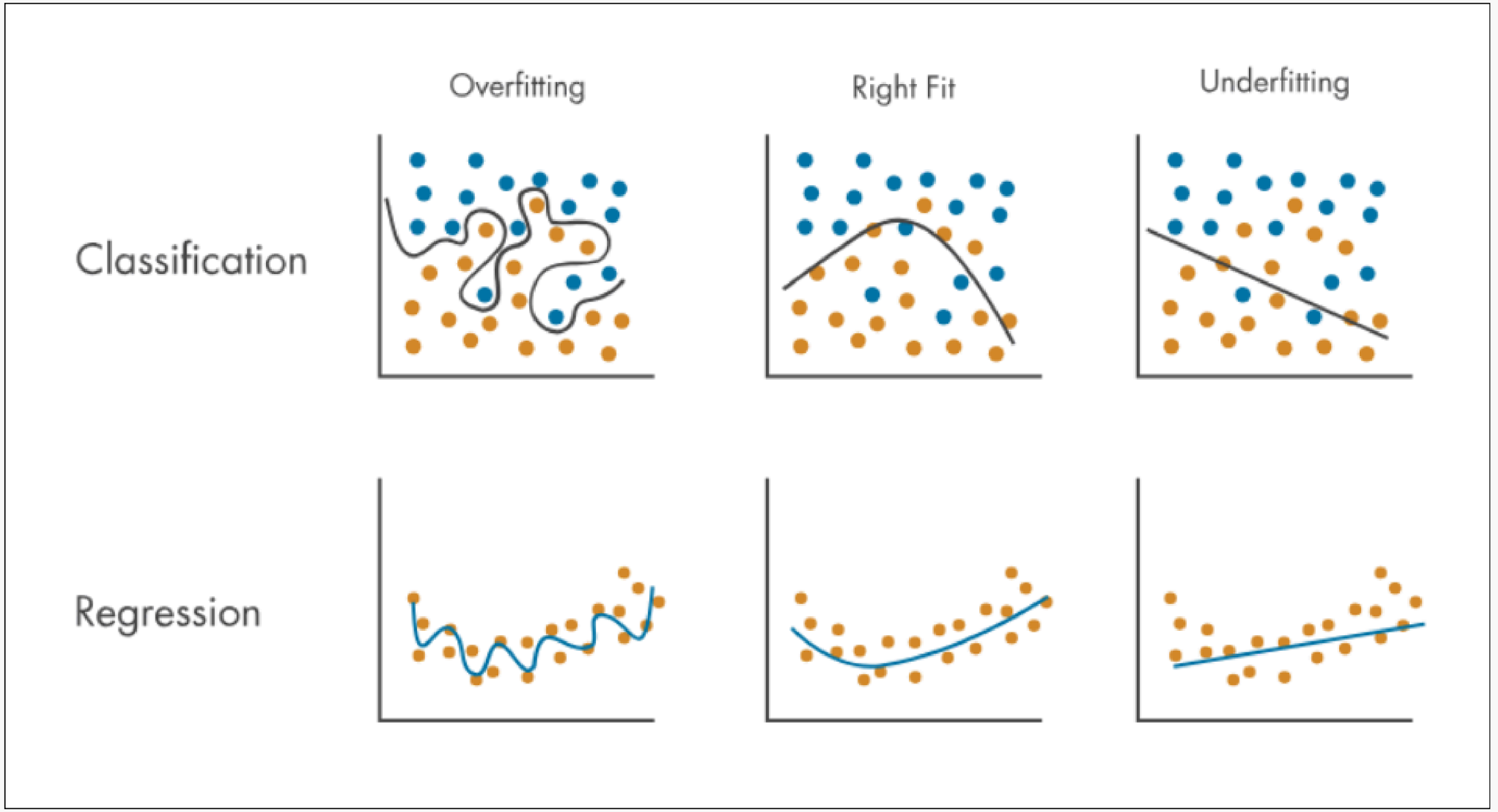
***Train set**: learn patterns*

***Test set**: checks how well those patterns generalize*

This helps detect:

***Overfitting**: the model memorizes training data but fails on new data (performs poorly on test set)*

***Underfitting**: the model is too simple, missing key patterns and performing poorly on both training and test data.*


Overfitting
Right Fit
Underfitting
Classification
Regression

What is KNN?

Code Implemetation

1) Select features (x values)

```
# Define features & target
# Features (independent variables)
X = data[['MinTemp', 'MaxTemp', 'Rainfall', 'Sunshine', 'WindGustSpeed', 'Humidity3pm', 'Pressure3pm']]

# Target (dependent variable)
y = data['RainTomorrow']
```

2) Split into train & test set

```
# Split dataset into training and testing sets
# test_size = 0.2 → 20% of data used for testing
# random_state ensures reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

What is KNN?

Code Implementation

3) Scaling the features (x values)

```
# Scale features to have mean=0 and std=1
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4) Form the KNN model using the scaled x feature, where k = 5 in this case

```
# Create KNN classifier (k=5 nearest neighbors)
knn_model = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn_model.fit(X_train_scaled, y_train)
```

What is KNN?

Code implementationation

5) Test on test set

```
# Predict on test set
y_pred = knn_model.predict(X_test_scaled)
```

What is KNN?

Code Implemetation

6) Predict new points

Extract raw probabilities for each class and the WINNING class gets assigned

E.g. Rain: 0.85, No rain: 0.15
Final decision: Rain

```python
# Example test points for predicting RainTomorrow
test_days = pd.DataFrame({
    'MinTemp': [15, 8, 22, 12, 18],           # minimum temperature in °C
    'MaxTemp': [25, 18, 30, 20, 28],          # maximum temperature in °C
    'Rainfall': [0.0, 5.0, 0.0, 10.0, 2.5],   # rainfall in mm
    'Sunshine': [8.0, 3.5, 10.0, 2.0, 6.5],   # hours of sunshine
    'WindGustSpeed': [25, 40, 15, 30, 20],    # km/h
    'Humidity3pm': [45, 80, 35, 90, 60],      # %
    'Pressure3pm': [1020, 1005, 1015, 1008, 1018] # hPa
})

# Scale the new test points (as defined above under Logistic Regression)
test_days_scaled = scaler.transform(test_days)

# Predict class (0=No rain, 1=Rain)
pred_class = knn_model.predict(test_days_scaled)

# Predict probability
pred_prob = knn_model.predict_proba(test_days_scaled)

# Print results
for i, (cls, prob) in enumerate(zip(pred_class, pred_prob)):
    print(f"Day {i+1}: Predicted RainTomorrow = {cls}, Probability of rain = {prob[1]:.2f}")
```

General Flow

Code implementation

1) **Select features and target**: Choose your input (X) and output (y) variables for the model

2) **Split dataset**: Divide data into **training, validation, and test sets** for **learning, tuning, and evaluation**

3) **Scale features**: Normalize or standardize numeric features so distance-based models perform correctly

4) **Train model**: Fit the KNN model using the training set to learn patterns

5) **Validate model**: Test on the validation set to tune parameters like k for best performance

6) **Retrain (optional)**: Refit the model on **both training and validation** sets using the chosen parameters

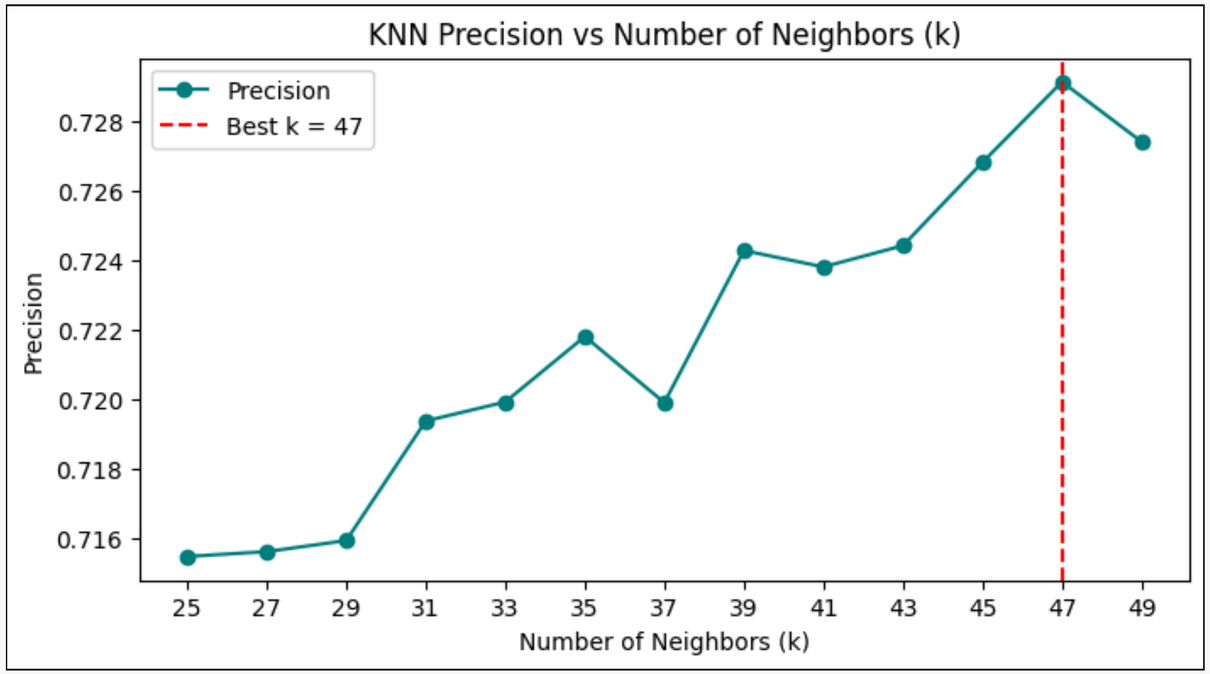
7) **Test model**: Evaluate the final model on unseen test data for unbiased performance

8) **Predict new data**: Use the trained model to make predictions on future or real-world inputs

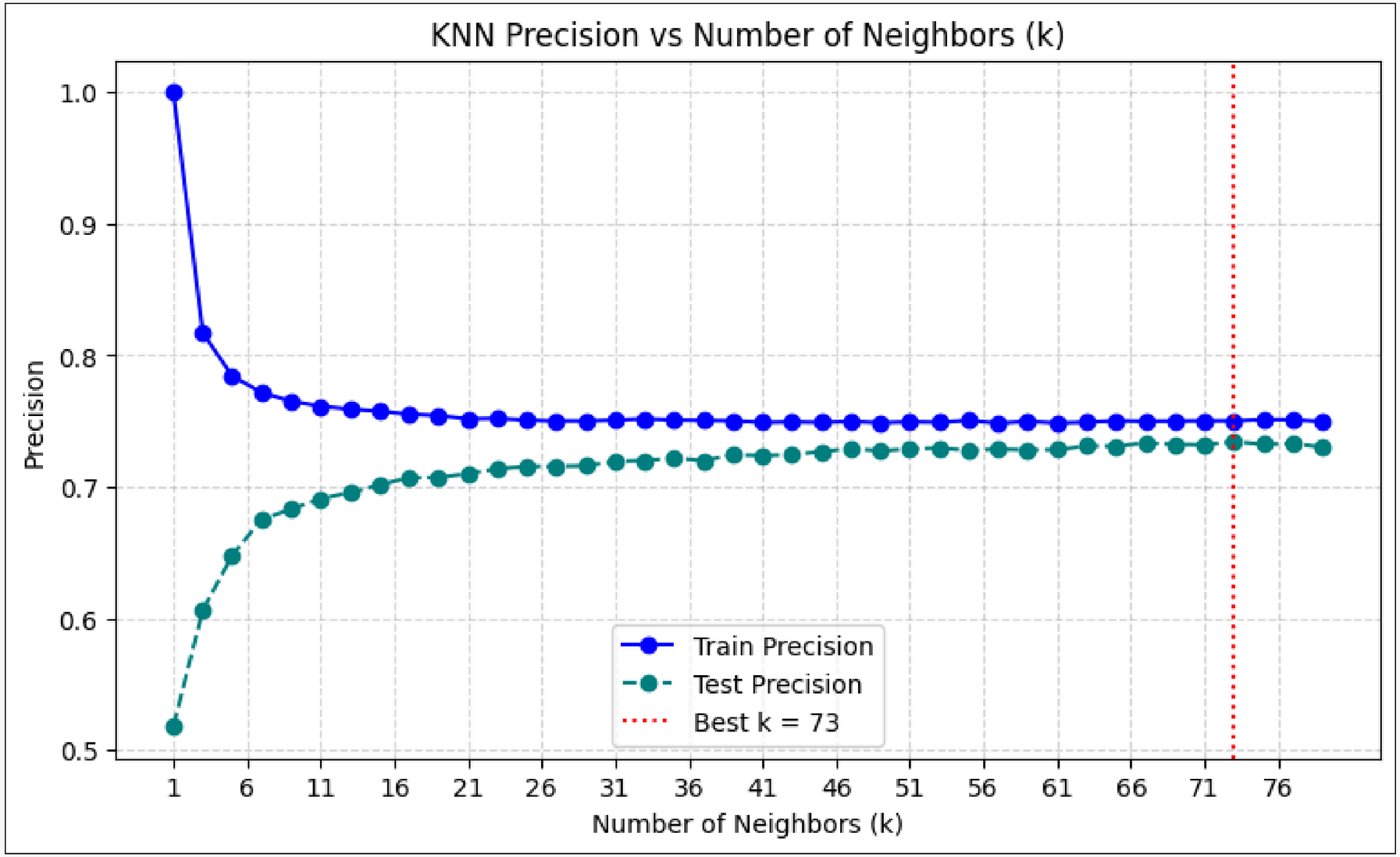
What is KNN?

Code Implemetation

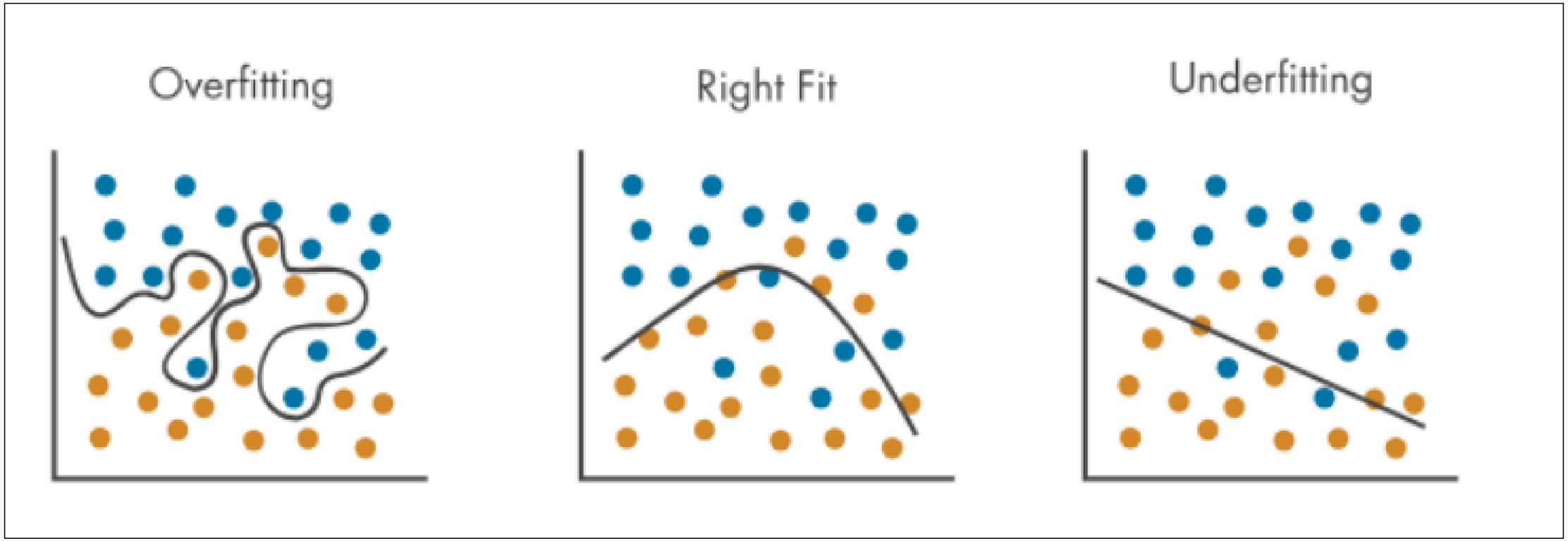
```
precisions = []
k_values = list(range(25, 51, 2))  # odd k values

# Loop through odd k values and record precision
for k in k_values:
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train_scaled, y_train)
    y_pred = knn_model.predict(X_test_scaled)

    # Compute precision (with "Yes" as positive class)
    prec = precision_score(y_test, y_pred, pos_label='Yes')
    precisions.append(prec)
```



GridSearchCV(), RandomizedSearchCV()



small k → overfits (too sensitive to individual data points, fails to identify patterns)
large k → underfits (oversimplifies the model by averaging predictions over a large number of neighbours)


Overfitting
Right Fit
Underfitting

Linear Regression

Linear Regression

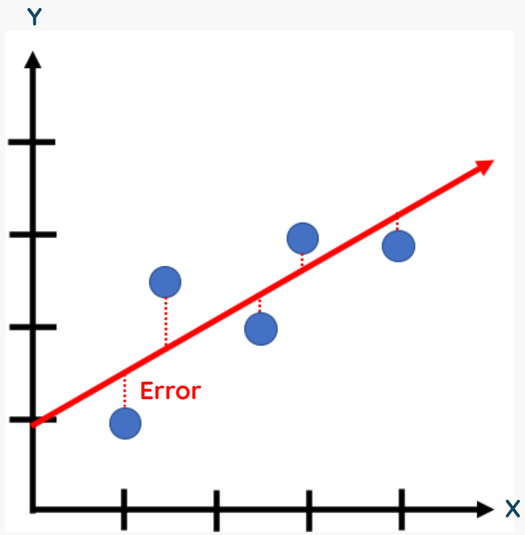
Main Idea

Finding the Best Fit

Simple linear regression finds the line that best represents the data by minimizing the sum of squared errors (Mean Squared Error)

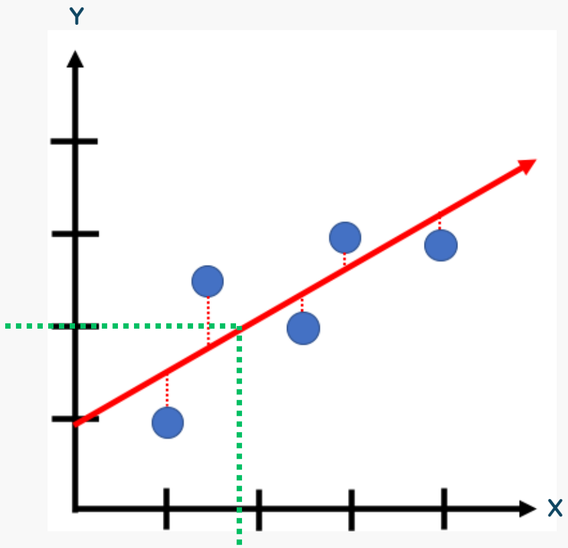
Mean Error Squared

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



Making Predictions

This fitted line is then used to predict outputs for new inputs by plugging them into the equation.



Linear Regression

Gradient Descent

Equation

The line is **$y = w_0 + w_1x$**, where w_1 is the gradient and w_0 is the intercept. Our goal is to find the **'best'** w_0 & w_1

Possible Gradient Descent Algorithm:

1. Choose initial values for w_0 & w_1 (either randomly or set to 0)
2. Pick a small learning rate α to control step size
3. Calculate the partial derivatives of the cost function w.r.t w_0 & w_1
4. Update w_0 & w_1
5. Repeat 3 & 4 until the cost function is minimised

Cost Function (Mean Squared Error)

$$J(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Where:

- $J(w_0, w_1)$ = cost / error
- $\hat{y}_i$ = predicted output for data point i
- y_i = actual output for data point i
- N = number of data points
- w_0 = intercept
- w_1 = gradient (slope)

$$J(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}\left((w_0 + w_1x_i) - y_i\right)^2$$

Partial Derivatives

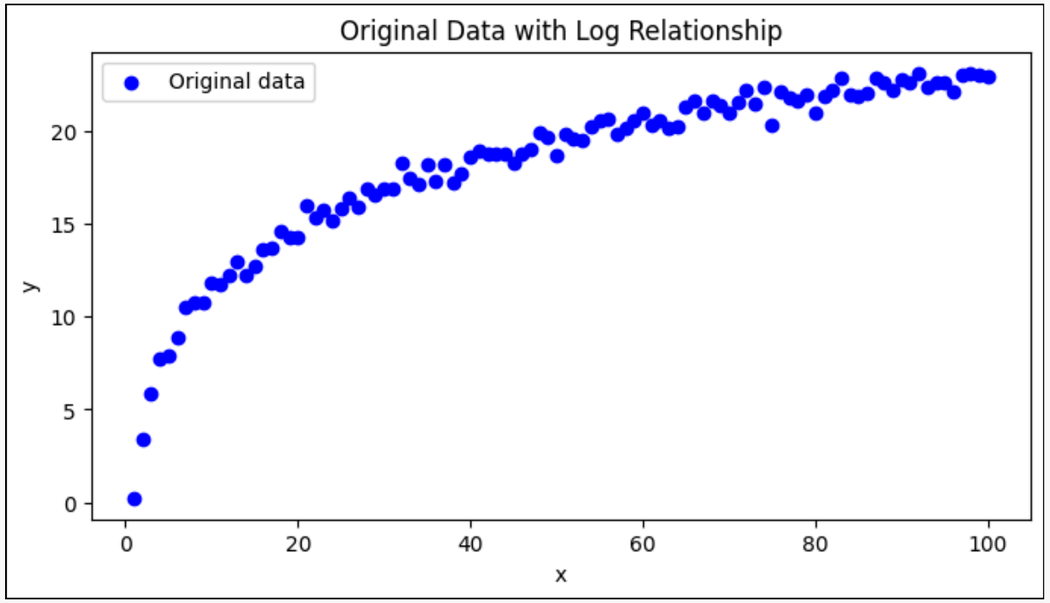
$$w_0 := w_0 - \alpha\frac{\partial J}{\partial w_0}, \quad w_1 := w_1 - \alpha\frac{\partial J}{\partial w_1}$$

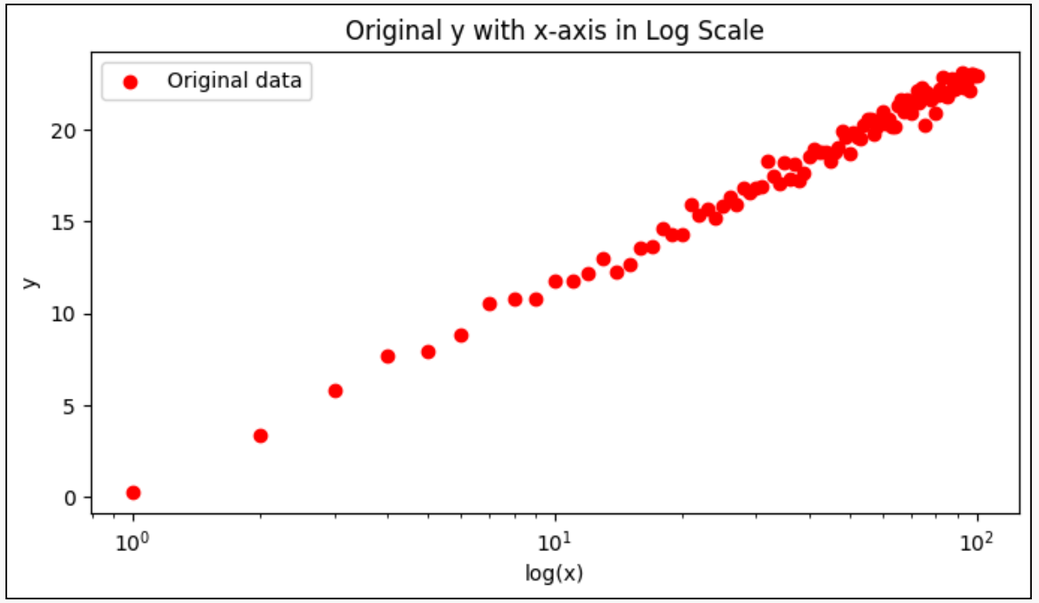
Where:

- w_0, w_1 = parameters to update
- α = learning rate (step size)
- $\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}$ = gradients of the cost function w.r.t. w_0 and w_1

Linear Regression

Linear regression assumes straight lines... but what if reality isn't?





Choosing the right transformation can make a non-linear relationship work with linear regression

Linear Regression

Linear regression assumes straight lines... but what if reality isn't?

Common Transformations

1) Log Transformation: log(y) and/or log(x)

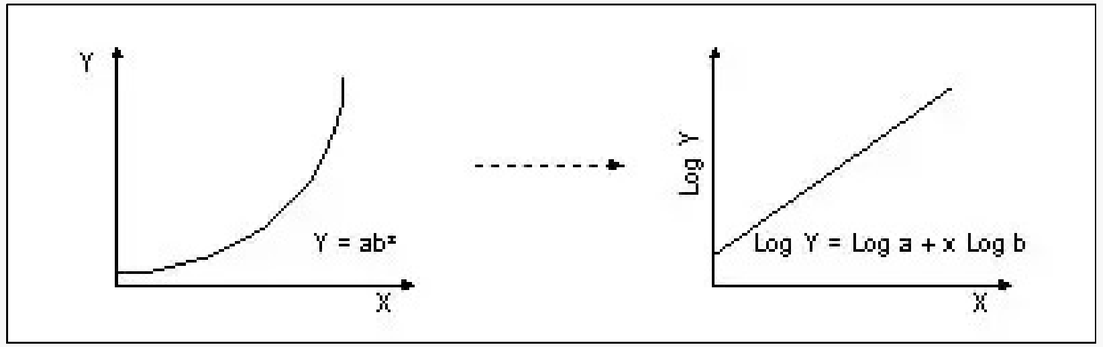
- Used when y and x has an exponential relationship

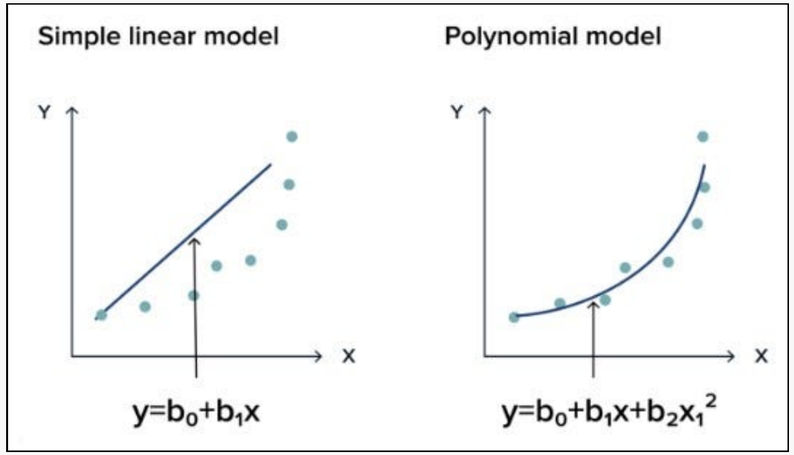
2) Root Transformation: √ x

- Linearise relationship (esp. if it seems like $Y \propto X^2$)

3) Polynomial Transformation: x^2, x^3

- Used when relationship is non-linear but continuous





Linear Regression

Code Implementation

```
# Load CSV file
data = pd.read_csv('data/house_selling_prices_FL.csv')

# View the first 5 rows
print(data.head())
```

```
# Load CSV file
data = pd.read_csv('house_selling_prices_FL.csv')

# View the first 5 rows
print(data.head())
```

Linear Regression

Code Implementation

```python
# Select input (X) and output (y)
X = data[['size']]  # input
y = data['price']   # output

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)
```

```python
# Predict on training data for plotting
y_pred_train = model.predict(X)

# Plot actual data and regression line
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred_train, color='red', label='Regression Line')
plt.xlabel('Size (sq meters)')
plt.ylabel('Price ($)')
plt.title('Linear Regression: Size vs Price')
plt.legend()
plt.show()

# Print slope and intercept
print(f"Slope (w1): {model.coef_[0]:.2f}")
print(f"Intercept (w0): {model.intercept_:.2f}")
```

Linear Regression

More Linear Regressions

Linear Regression

Regression Type	Description	Equation/Loss Function
Simple Linear Regression	One independent (x) and one dependent (y) variable	$\hat{y} = wx + b$ $Loss = \sum \frac{(y - \hat{y})^2}{n}$
Polynomial Linear Regression	Polynomial features ($x, x^2, \ldots, x^n$) and one dependent (y) variable	$\hat{y} = w_1 x + w_2 x^2 + \cdots + b$ $Loss = \sum \frac{(y - \hat{y})^2}{n}$
Multiple Linear Regression	Arbitrary features ($x_1, x_2, \ldots, x_n$) and one dependent (y) variable	$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + b$ $Loss = \sum \frac{(y - \hat{y})^2}{n}$

Logistic Regression

Logistic Regression

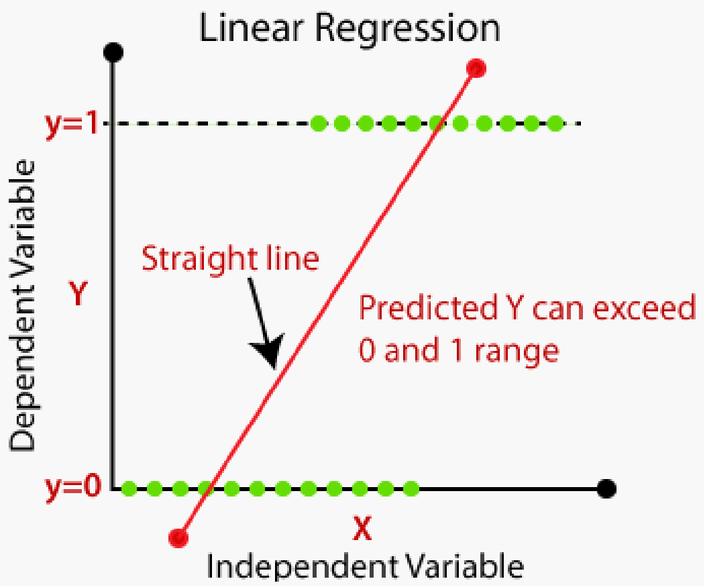
Main Idea

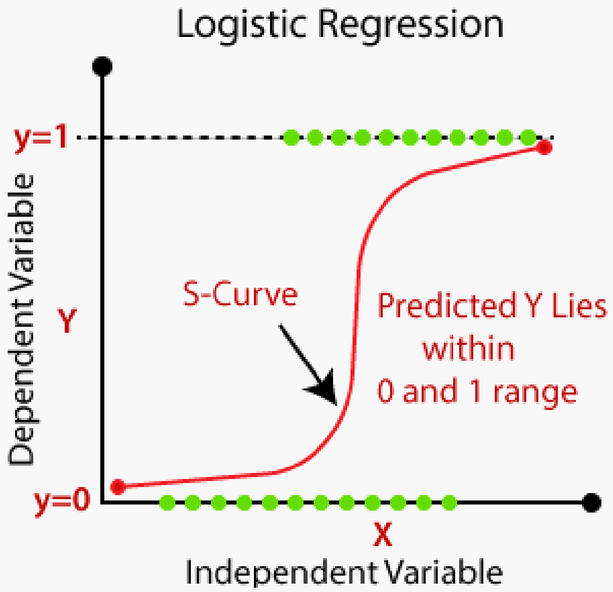
What is it?

Classification algorithm used to predict the probability of a binary outcome

Making Predictions

Predicts the probability of a class label (eg. cancel or no cancel), then assigns the label based on a threshold





Logistic Regression

Construction

1) Apply the regression function

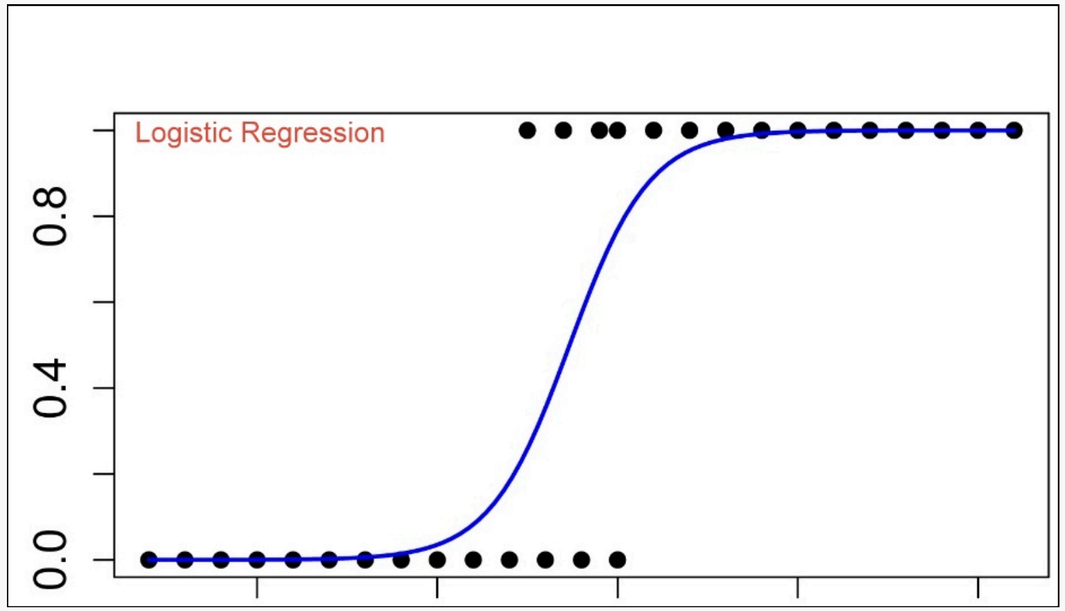
$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

2) Pass results into the sigmoid function

Find the probability (0 to 1) $\phi(z) = \frac{1}{1 + e^{-z}}$

3) Predict based on threshold

Those with estimated probability ≥ threshold belongs to the “positive class” (label = 1), otherwise the “negative class” (label = 0)



Sigmoid function ensures any input value (−∞ to +∞) will be mapped between 0 and 1

Logistic Regression

Code Implementation

```
# Load CSV file
data = pd.read_csv('data/weatherAUS_rainfall_prediction_dataset_cleaned.csv')

# View the first 5 rows
print(data.head())
```


```
# Load CSV file
data = pd.read_csv('weatherAUS_rainfall_prediction_dataset_cleaned.csv')

# View the first 5 rows
print(data.head())
```

Logistic Regression

Code Implementation

```
# Split dataset into training and testing sets
# test_size = 0.2 → 20% of data used for testing
# random_state ensures reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

Data splitted into train and test set to prevent overfitting and ensure better overall result!!

```
# Initialize logistic regression model
model = LogisticRegression()

# Train the model using training data
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
```

Logistic Regression

Code Implementation

```
# Split dataset into training and testing sets
# test_size = 0.2 → 20% of data used for testing
# random_state ensures reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize logistic regression model
model = LogisticRegression(max_iter=1000)

# Train the model using training data
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", round(accuracy, 2))
```

```
Accuracy: 0.84
```

Logistic Regression

Code Implementation

Predicting Sample Points with Logistic Regression

We will use **Logistic Regression** to predict the **same sample points** that were used in the **KNN example**.

```
# Example test points for predicting RainTomorrow
test_days = pd.DataFrame({
    'MinTemp': [15, 8, 22, 12, 18],           # minimum temperature in °C
    'MaxTemp': [25, 18, 30, 20, 28],          # maximum temperature in °C
    'Rainfall': [0.0, 5.0, 0.0, 10.0, 2.5],   # rainfall in mm
    'Sunshine': [8.0, 3.5, 10.0, 2.0, 6.5],   # hours of sunshine
    'WindGustSpeed': [25, 40, 15, 30, 20],    # km/h
    'Humidity3pm': [45, 80, 35, 90, 60],      # %
    'Pressure3pm': [1020, 1005, 1015, 1008, 1018] # hPa
})

# Predict class (0 = no disease, 1 = disease)
pred_class = model.predict(test_days)

# Predict probability
pred_prob = model.predict_proba(test_days)

# Print nicely
for i, (cls, prob) in enumerate(zip(pred_class, pred_prob)):
    print(f"Day {i+1}: Predicted RainTomorrow = {cls}, Probability of rain = {prob[1]:.2f}")
```

✓ 0.0s

```
Day 1: Predicted RainTomorrow = No, Probability of rain = 0.05
Day 2: Predicted RainTomorrow = Yes, Probability of rain = 0.65
Day 3: Predicted RainTomorrow = No, Probability of rain = 0.01
Day 4: Predicted RainTomorrow = Yes, Probability of rain = 0.76
Day 5: Predicted RainTomorrow = No, Probability of rain = 0.13
```

Logistic Regression

More Regressions

	Regression Type	Description	Equation/Loss Function
Linear Regression	Simple Linear Regression	One independent (x) and one dependent (y) variable	$\hat{y} = wx + b$ $Loss = \sum \frac{(y-\hat{y})^2}{n}$
	Polynomial Linear Regression	Polynomial features ($x, x^2, \ldots, x^n$) and one dependent (y) variable	$\hat{y} = w_1 x + w_2 x^2 + \cdots + b$ $Loss = \sum \frac{(y-\hat{y})^2}{n}$
	Multiple Linear Regression	Arbitrary features ($x_1, x_2, \ldots, x_n$) and one dependent (y) variable	$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + b$ $Loss = \sum \frac{(y-\hat{y})^2}{n}$
Regularized Regression	Ridge Regression	Linear Regression with L2 Regularization	$Loss = \sum \frac{(y-\hat{y})^2}{n} + \lambda \sum_{i=1}^{n} w_i^2$
	Lasso Regression	Linear Regression with L1 Regularization	$Loss = \sum \frac{(y-\hat{y})^2}{n} + \lambda \sum_{i=1}^{n} \lvert w_i \rvert$
	Elastic Net	Linear Regression with BOTH L1 and L2 Regularization	$Loss = \sum \frac{(y-\hat{y})^2}{n} + \lambda((1-\alpha) * \underbrace{\sum w_i^2}_{L2} + \alpha * \underbrace{\sum \lvert w_i \rvert}_{L1})$
Categorical Probability	Logistic Regression	One (or more) independent variable(s) to predict BINARY outcome probability	$P(X) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + \cdots + b)}}$
	Multinomial Logistic Regression *(or Softmax Regression)*	One (or more) independent variable(s) to predict MULTIPLE categorical probabilities	$P(Y = k \mid X) = \frac{e^{score_k}}{\sum_{j=1}^{K} e^{score_j}}$

Applications

Linear Regression

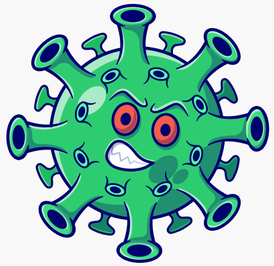
- *Predict housing prices based on size, location, number of bedrooms*

- *Estimate employee salary based on experience, education, and role*
- *Predict future sales of a product from historical data*

VS

Logistic Regression

- *Classify whether a person will default on a loan (yes/no)*
- *Classify emails as spam or not spam*
- *Predict the presence/absence of a disease (e.g., diabetes)*



Support Vector Machine

What is SVM?

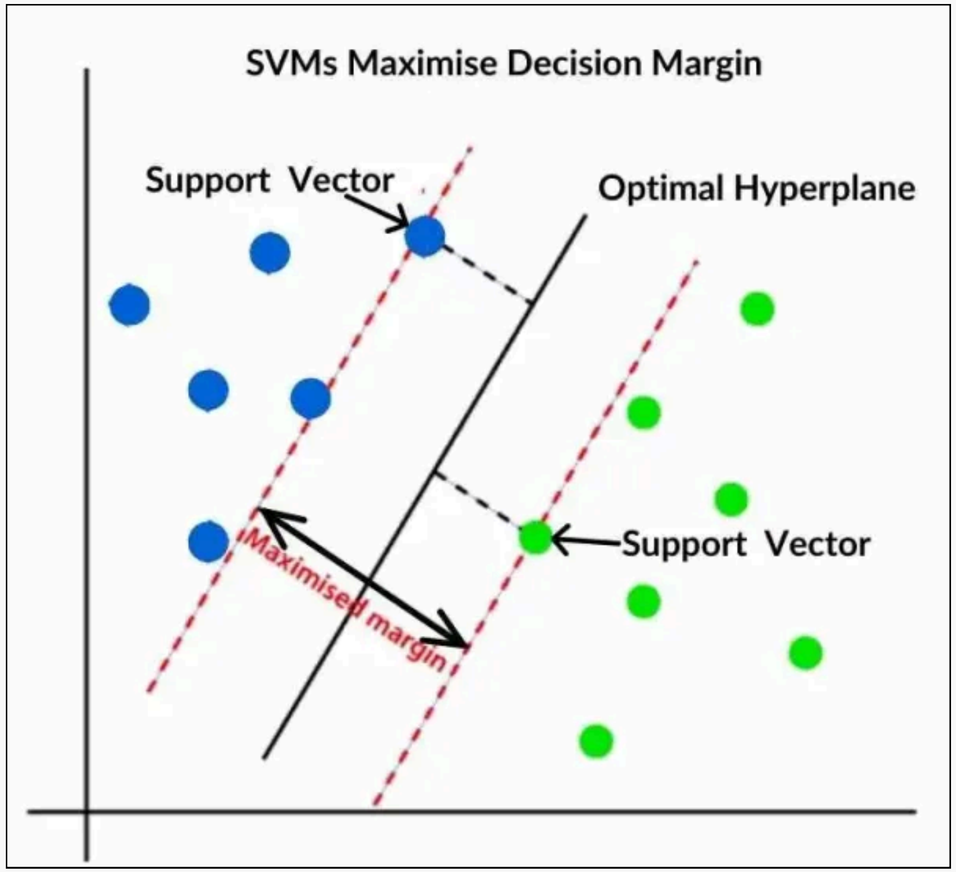
machine learning method that finds the best boundary (a line in 2D, a plane in 3D, or a hyperplane in higher dimensions) that separates different classes of data

What is the ‘best’ boundary?

one that maximizes the margin — the distance between the boundary and the closest data points from each class

Why?

These closest points, called support vectors, define this margin and ensure new points can be classified with more confidence



What is SVM?

the mathematical approach

Decision Boundary (Hyperplane):

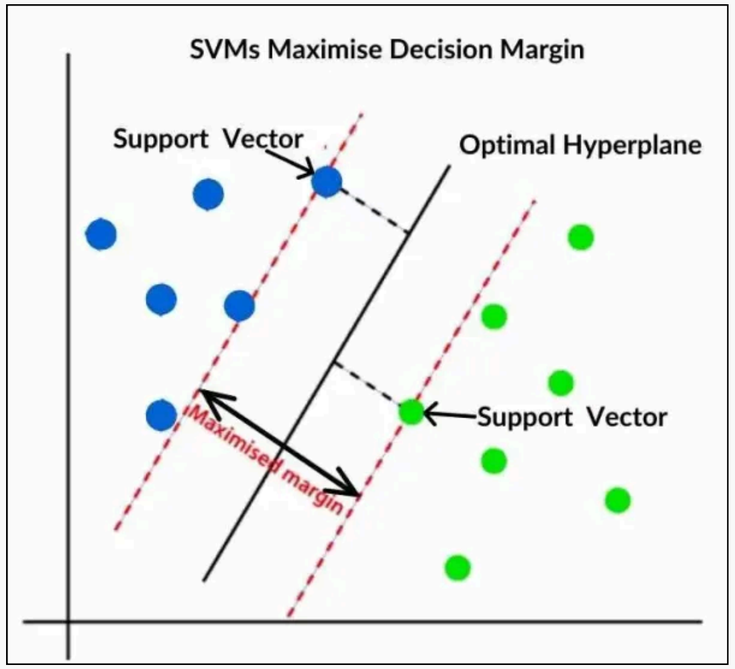
line separating the points

$$f(x) = w^T x + b = 0$$

Margin

gap between the
boundary & support vector

$$\text{Margin} = \frac{|w^T x_i + b|}{\|w\|}$$



Optimization problem

Find w & b that maximize the margin
while classifying all points correctly

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t. } y_i(w^T x_i + b) \geq 1 \quad \forall i$$

What is SVM?

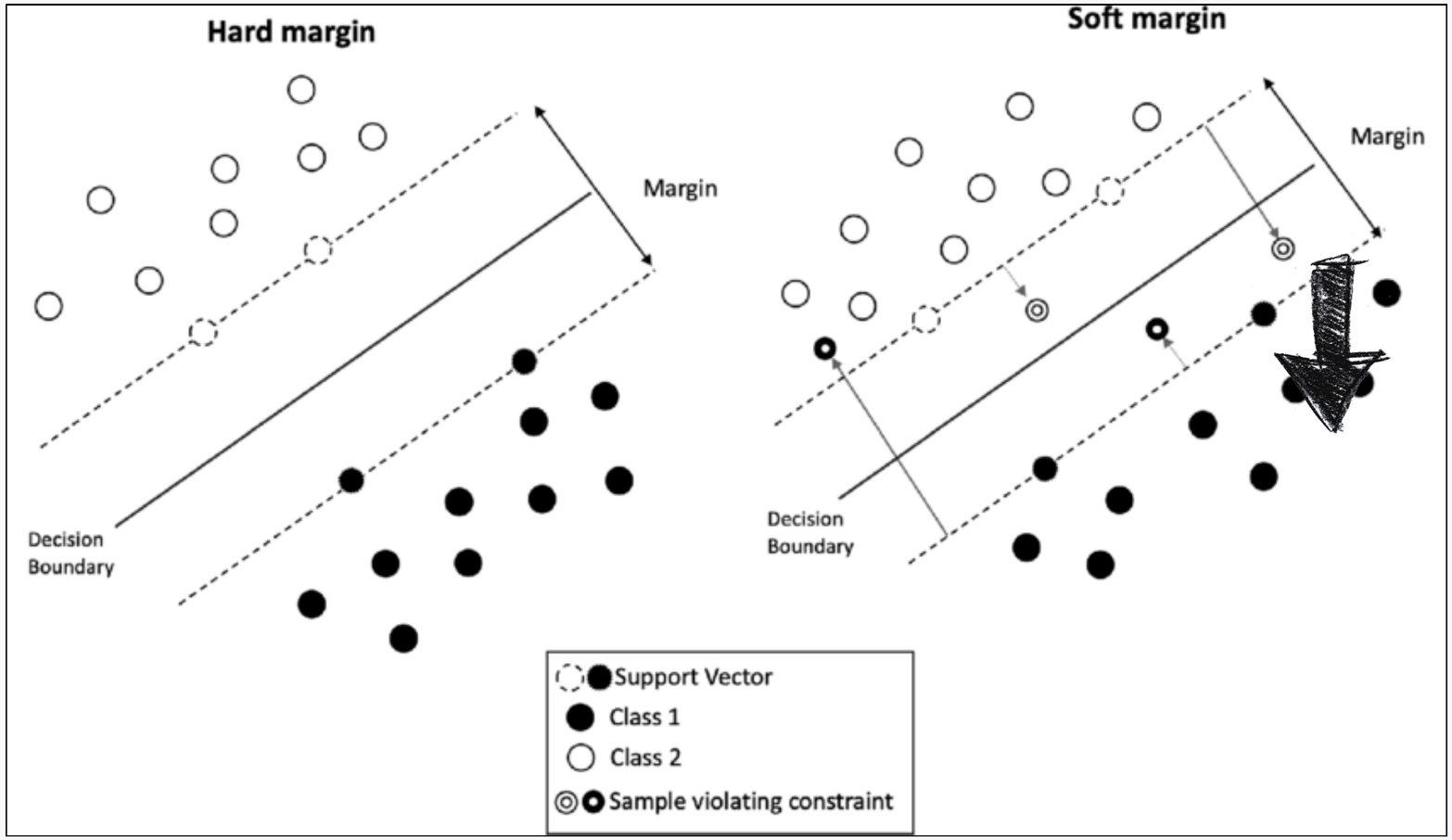
Hard Margin SVM

Hard-margin SVM

a perfect linear boundary without any misclassifications

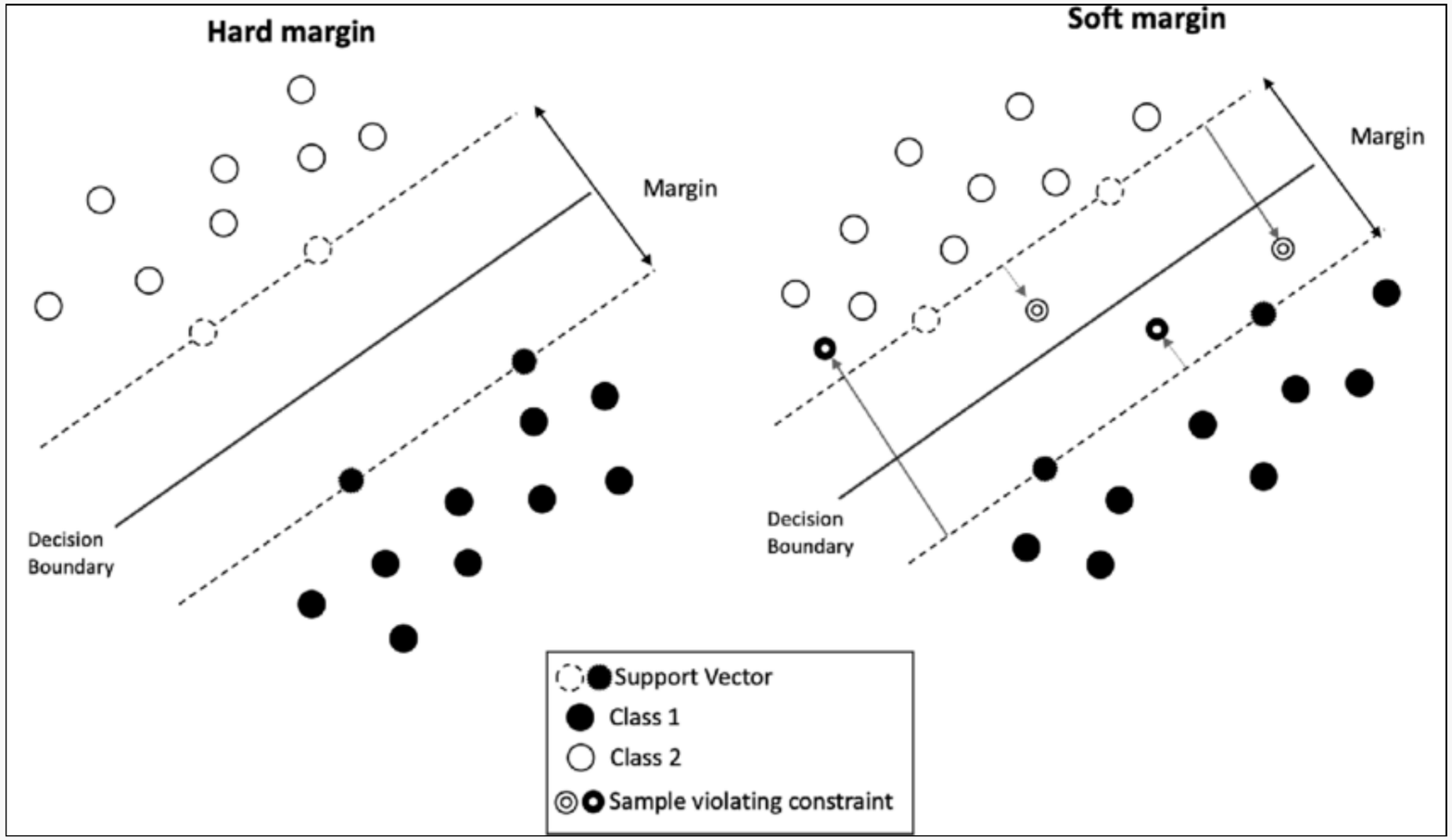
works only for perfectly linearly separable data

much simpler to form



What is SVM?

Soft Margin SVM



Soft-margin SVM

some misclassifications or margin violations

create a more robust and generalizable model for real-world, non-linearly separable data

more usable in real world
noisy data as it helps
prevent overfitting

What is SVM?

Hard vs Soft Margin

Hard-margin SVM

Example usage:

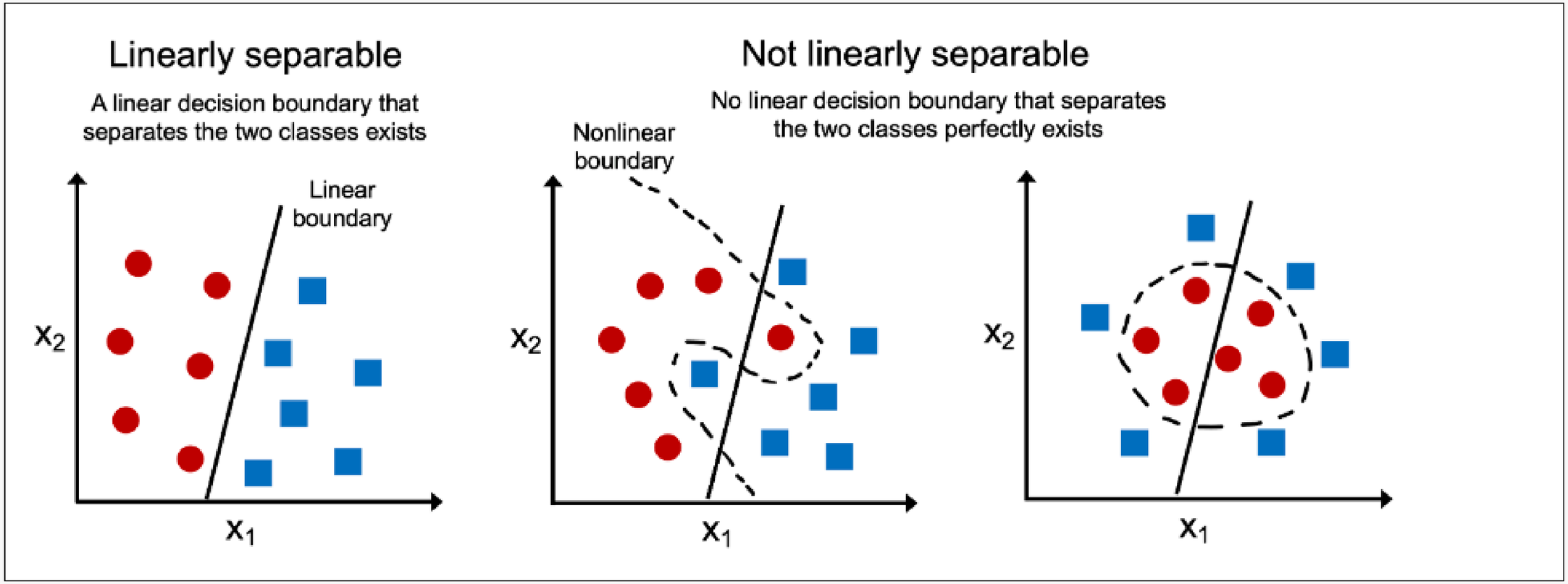
- manufacturing defect detection with very controlled sensors
- lab experiments

prediction must be exact and there is no room for error

Soft-margin SVM

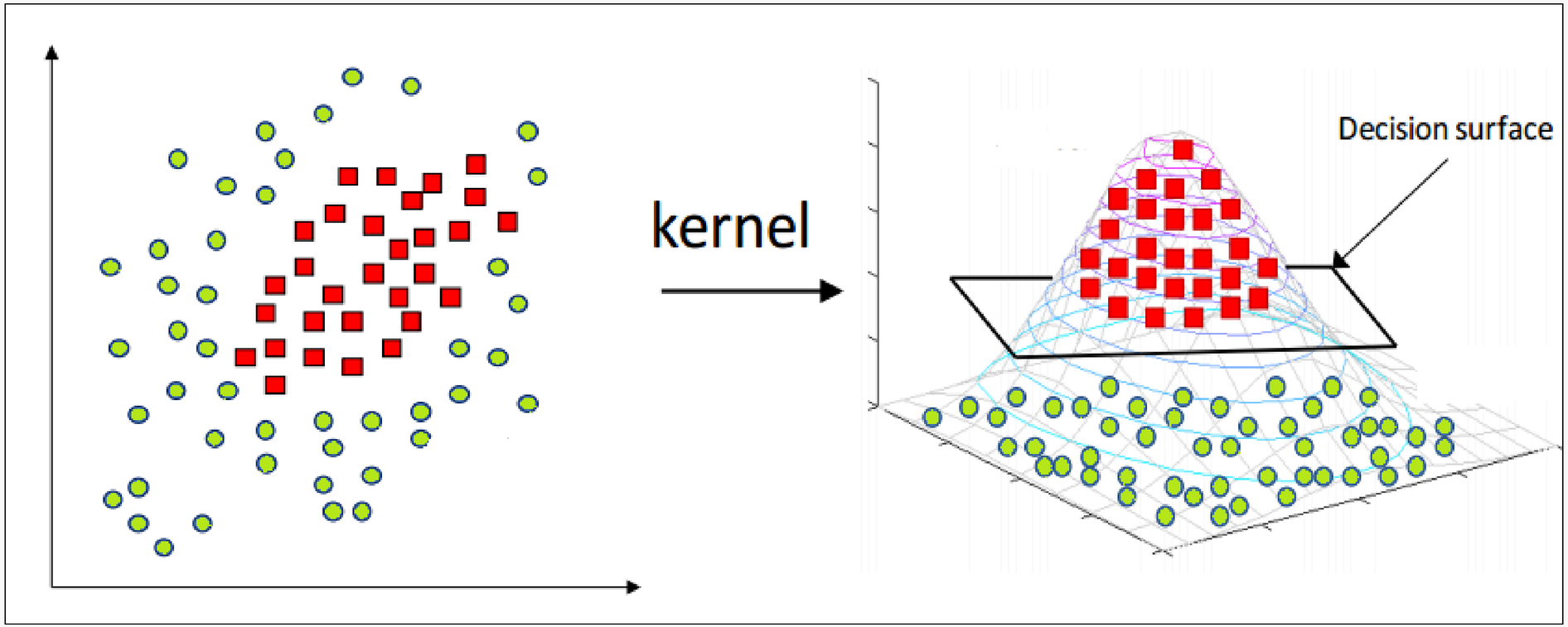
Example usage:

- spam email detection
- weather prediction
- medical diagnosis


Linearly separable
A linear decision boundary that separates the two classes exists
Linear boundary
x_2
x_1
Not linearly separable
No linear decision boundary that separates the two classes perfectly exists
Nonlinear boundary
x_2
x_1
x_2
x_1

Kernel SVM

Utilizes kernel functions to map the non-linearly separable input data points into a higher-dimensional space where the two classes can now be linearly separated



Kernel SVM

Why kernel SVM when we have feature transformation?

Feature transformation

- **explicitly transforms** original features into a higher-dimensional space
- very **expensive** as the number of new features grows
- **manually** decide how to transform features

Kernel SVM

- **implicitly maps** the data into a higher-dimensional space **without generating new features (Kernel Trick)**
- can **model complex, non-linear boundaries**
- choose a kernel (e.g., RBF, polynomial) and **let the SVM handle the mapping**

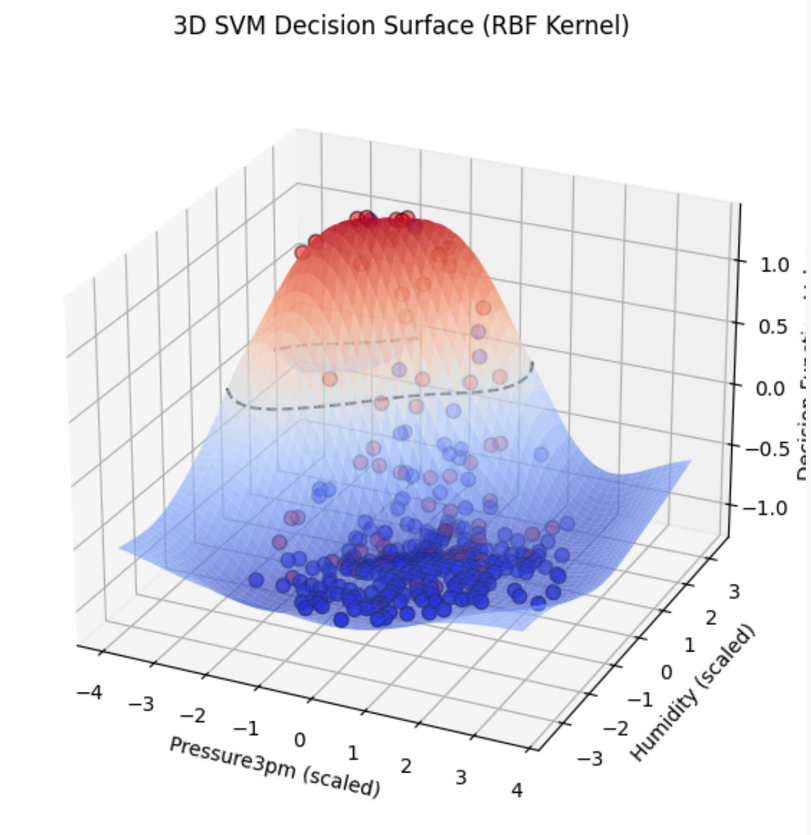
What is SVM?

Code implementation

RBF:
Calculates similarity score between points and makes points that are close (similar) in feature space “stick together” and those far apart stay separate

```
# Create SVM classifier with RBF kernel
# kernel='rbf' is the default and works well for non-linear boundaries.
svm_model = SVC(kernel='rbf', probability=True, random_state=42)

# Train the model
svm_model.fit(X_train_scaled, y_train)
```

```
# Standardize features (mean=0, std=1)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



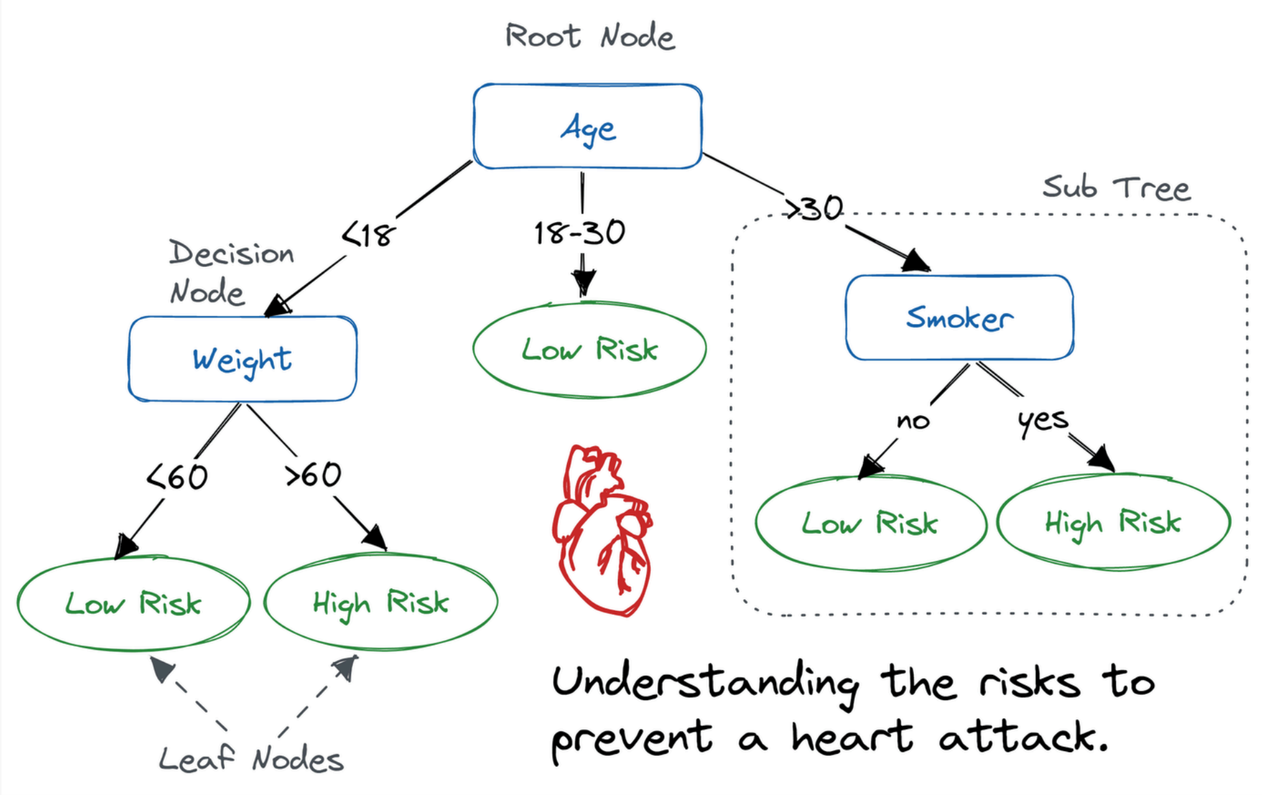
KNN vs Logistic vs SVM

	KNN	Logistic	SVM
Decision Boundary	Non Linear (depends on k)	Linear	Linear or non-linear (kernel)
Training	None	Fast	Fast for linear, Slow for kernel
Prediction	Slow (computes distances to all points)	Fast	Fast (just evaluate hyperplane)
Best use case	Low-dimensional datasets with clear clusters	Binary/Multi classification (one-vs-rest)	High-dimensional data
Weakness	Slow for large datasets Sensitive to irrelevant/noisy features	Poor with non-linear decision boundaries Cannot handle complex interactions without feature engineering	Sensitive to choice of kernel & parameters

Decision Tree

What is Decision Tree?

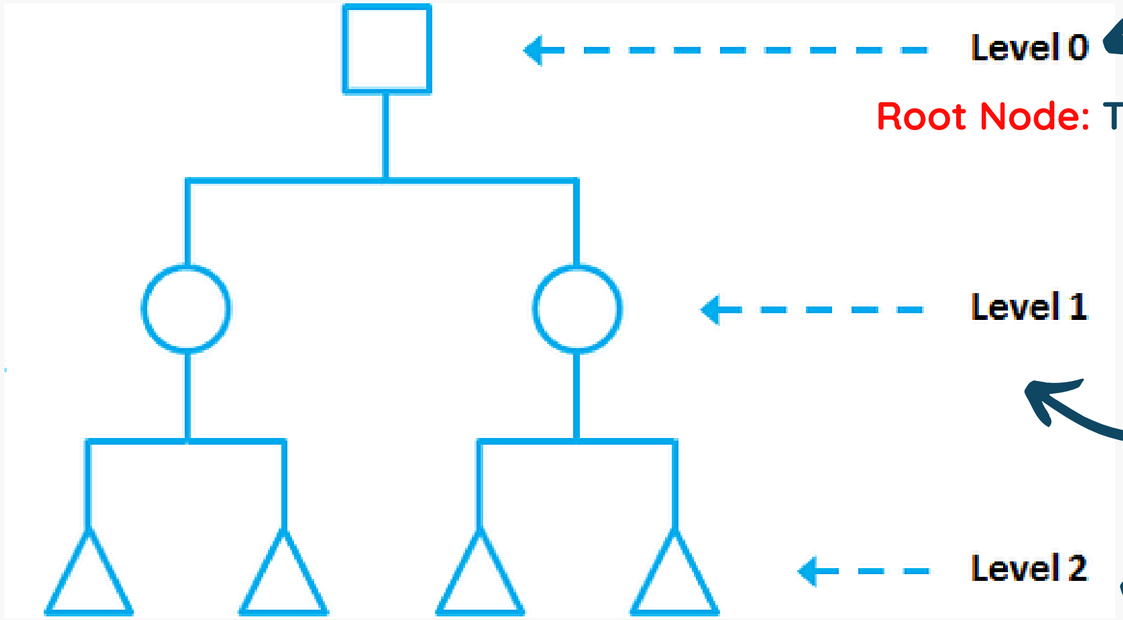
predicts outcomes by asking a series of yes/no questions about the data — just like a flowchart



Try it out yourself:

Would a 34 year old who smokes be at a high / low risk of a heart attack?

What is Decision Tree?



Root Node: The topmost node of the tree which performs the first split on the most important feature.

Internal Nodes: Nodes between the root and leaves, each one represents another condition or decision. They take a subset of data and split it further based on another feature to reduce impurity.

Leaf Nodes: The end points of the tree, no further splitting happens here. They output the final prediction or class label, e.g. “Rain = Yes”.

Lastly, the branches are the lines connecting nodes. Each branch represents the outcome of a condition (e.g., “Yes” or “No” path).

What is Decision Tree?

Decision Tree for Classification

1. Split data on the feature that best separates classes (Gini / Entropy). In practice, the algorithm:

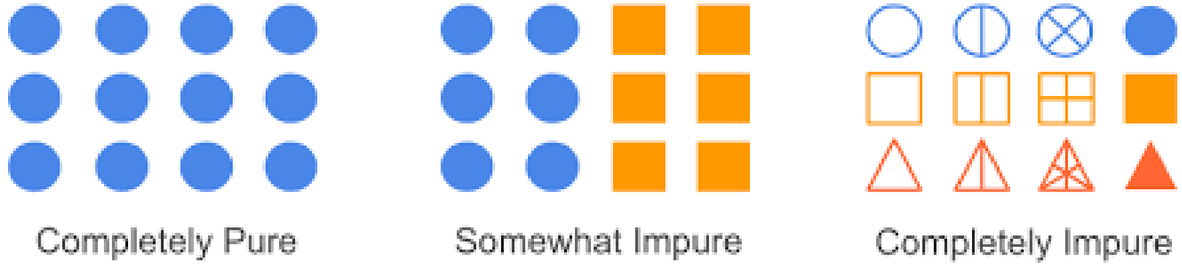
- Minimises Gini impurity (lower Gini → fewer mixed labels within a node)
- Maximises Information Gain (Reduction in entropy after the split)
- In both cases, we aim to reduce uncertainty about the class labels after each split as this suggests that the model has separated the classes well

2. Branch recursively using other features
3. The majority class (most frequent label at that node) is assigned as the predicted class

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

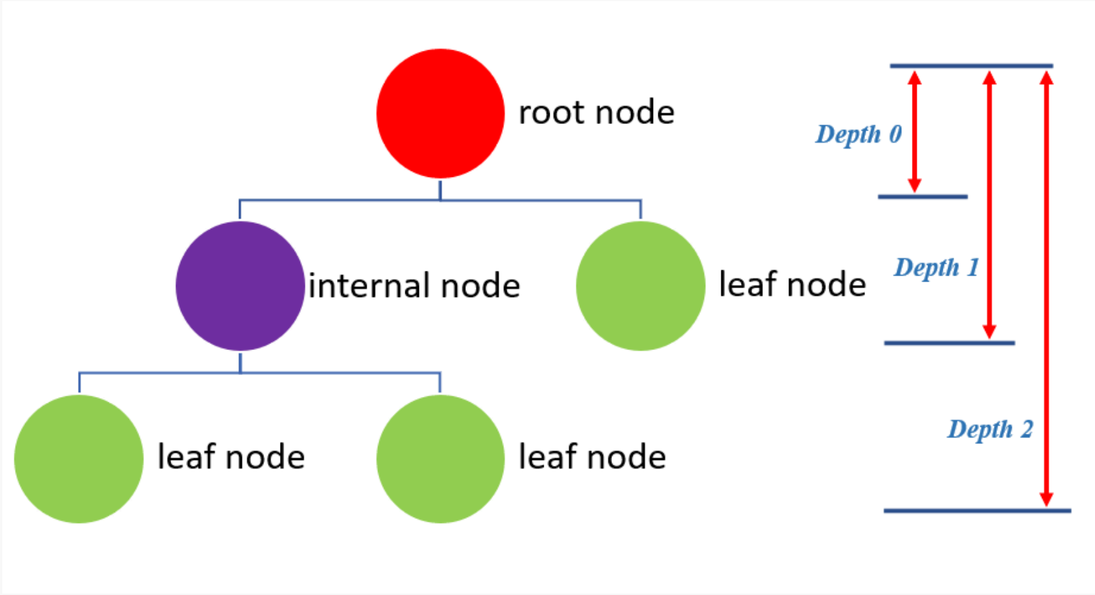
where $p(c_i)$ is the probability/percentage of class c_i in a node.



What is Decision Tree?

Limitation: Overfitting

- max_depth → limits how many levels of splits a decision tree can make
  - Purpose: control model complexity to prevent the tree from growing “too deep”



What is Decision Tree?

Limitation: Overfitting

Depth	Model Behavior	Example
Too shallow	**Underfitting:** misses important patterns	Only captures very broad trends
Too deep	**Overfitting:** memorises training data	Perfect on training, poor on test
Just right	**Good generalization**	Captures real structure, ignores noise

- As depth increases:
  - Training accuracy → always increases (model fits data better)
  - Test accuracy → increases up to a point, then drops (overfits)
- Thus, it is important to set the right max_depth to prevent overfitting!

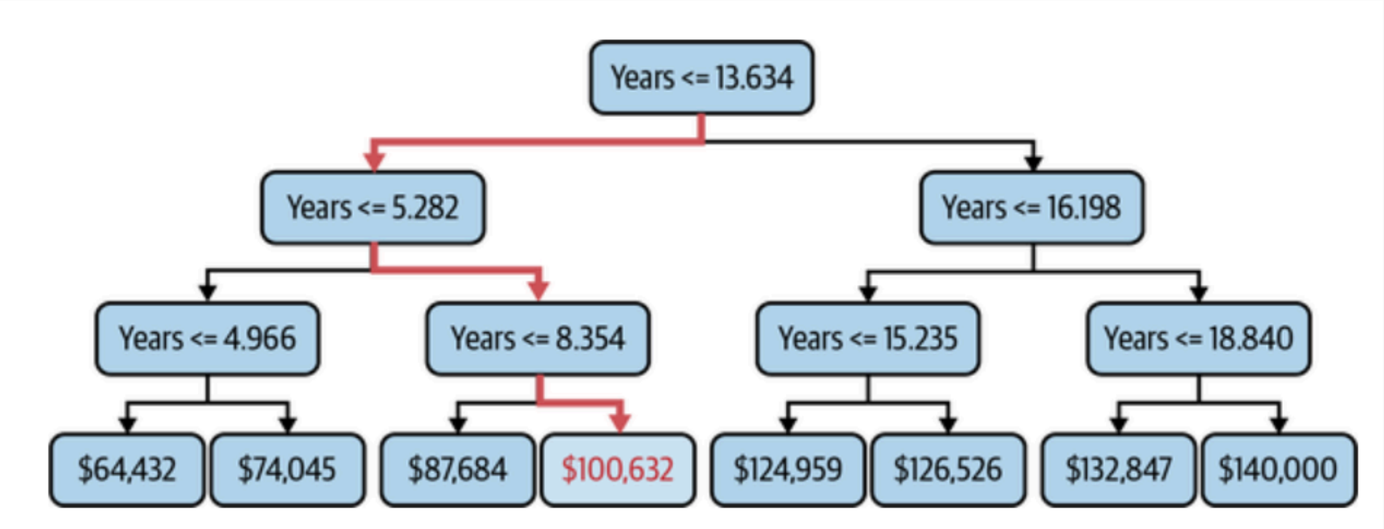
What is Decision Tree?

Decision Tree for Regression

1. Split data on the feature/threshold that minimises MSE → pick the split that makes each group's predicted values as close as possible to their real values

$$\text{MSE} = \frac{1}{n}\sum(Y_i - \hat{Y}_i)^2$$

2. Branch recursively until stopping criteria are met (depth, min samples, etc.)
3. Predict value at each leaf as the average of target values in that leaf



What is Decision Tree?

Code Implemetation

BEFORE WE START

As we observe our dataset, we notice the problem of class imbalance.

```
In [4]: # class imbalance
        print("Class distribution in y_train:")
        print(y_train.value_counts())

Class distribution in y_train:
RainTomorrow
No     90791
Yes    25577
Name: count, dtype: int64
```

Class imbalance means one class (e.g. "No Rain") is much more frequent than the other ("Rain").

- If not addressed, models may simply predict the majority class, ignoring the minority.
- Handling imbalance (e.g. with class weights or scale_pos_weight) helps the model pay more attention to the minority class, improving its ability to detect rare events like rain.

Thus, we tackle it using class_weight = 'balanced' for DT & RF, and scaling for XGBoost later on.

What is Decision Tree?

Code Implemetation

1. Train the decision tree using an 80/20 split

In [5]:

```
# Create Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42,
                                  max_depth=5,
                                  class_weight = 'balanced')

# Train the model
dt_model.fit(X_train, y_train)

# Predict on test set
y_pred = dt_model.predict(X_test)

# Compute precision (with "Yes" as positive class)
precision = precision_score(y_test, y_pred, pos_label="Yes")
print(f"Precision: {precision:.2f}")
```

```
Precision: 0.47
```

What is Decision Tree?

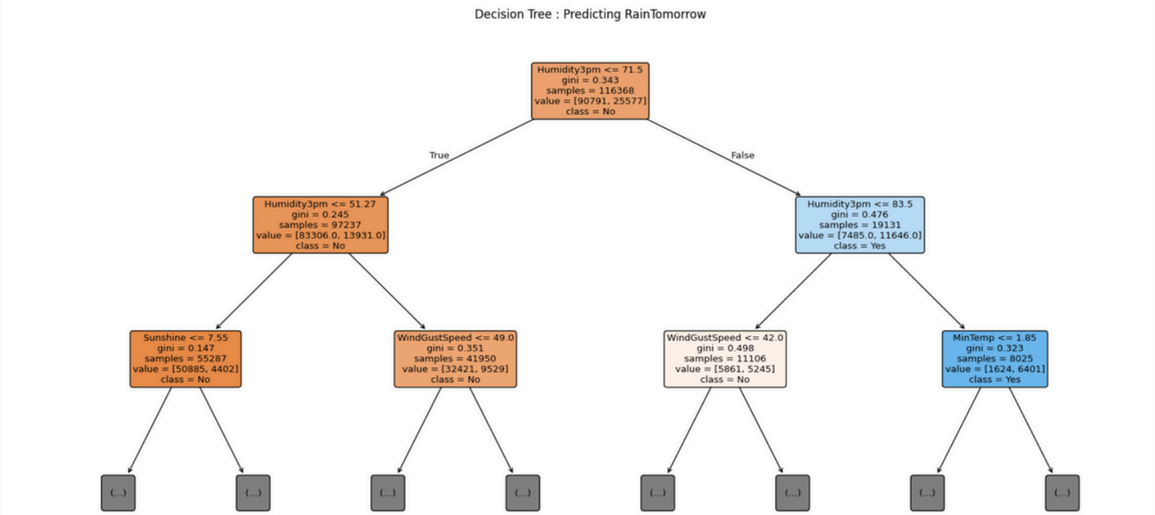
Code implementation

2. Visualise the Decision Tree

Visualise the Tree!

[19]:

```
plt.figure(figsize=(20,10))

#Here, we set max depth = 2 to get a better understanding of the split, but feel free to remove to see the entir
plot_tree(dt_model,
          filled=True,
          feature_names=X.columns,
          class_names=["No", "Yes"],
          max_depth=2,
          rounded=True)
plt.title("Decision Tree : Predicting RainTomorrow")
plt.show()
```



What is Decision Tree?

Code implementation

3. Use the model to predict a given set of data points

Now that we understand how Decision Trees work, let's try to predict a few points to see how it performs

]:

```
#Here, I use the same test_days as with Daeren's model
test_days = pd.DataFrame({
    'MinTemp': [15, 8, 22, 12, 18],
    'MaxTemp': [25, 18, 30, 20, 28],
    'Rainfall': [0.0, 5.0, 0.0, 10.0, 2.5],
    'Sunshine': [8.0, 3.5, 10.0, 2.0, 6.5],
    'WindGustSpeed': [25, 40, 15, 30, 20],
    'Humidity3pm': [45, 80, 35, 90, 60],
    'Pressure3pm': [1020, 1005, 1015, 1008, 1018]
})

pred_class = dt_model.predict(test_days)
pred_prob = dt_model.predict_proba(test_days)

for i, (cls, prob) in enumerate(zip(pred_class, pred_prob)):
    print(f"Day {i+1}: Predicted RainTomorrow = {cls}, Probability of rain = {prob[1]:.2f}")
```

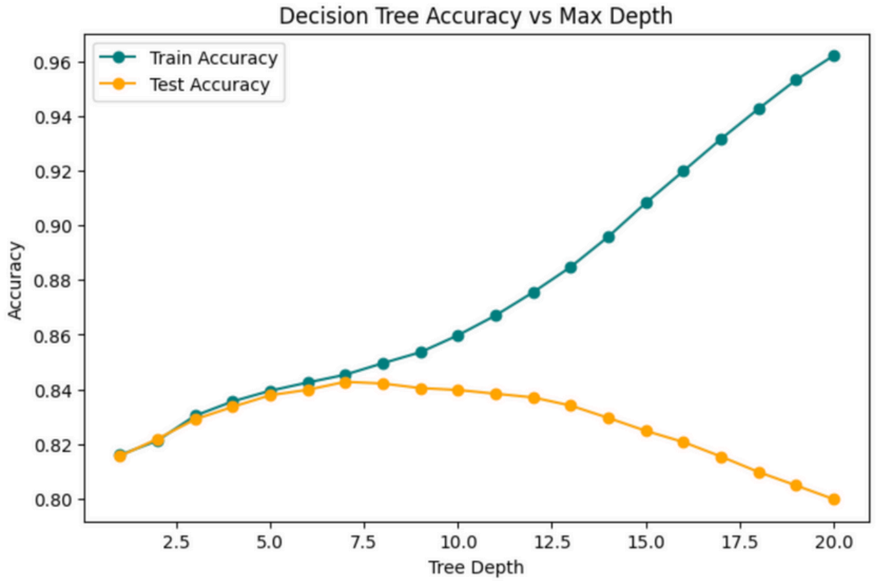
```
Day 1: Predicted RainTomorrow = No, Probability of rain = 0.04
Day 2: Predicted RainTomorrow = Yes, Probability of rain = 0.63
Day 3: Predicted RainTomorrow = No, Probability of rain = 0.04
Day 4: Predicted RainTomorrow = Yes, Probability of rain = 0.92
Day 5: Predicted RainTomorrow = No, Probability of rain = 0.13
```

What is Decision Tree?

Code Implementation

4. Lastly, let's observe how accuracy is related to the decision tree via the parameter : max_depth

[26]:
```
# Plot
plt.figure(figsize=(8,5))
plt.plot(depth_values, train_acc, marker='o', label='Train Accuracy', color='teal')
plt.plot(depth_values, test_acc, marker='o', label='Test Accuracy', color='orange')
plt.title("Decision Tree Accuracy vs Max Depth")
plt.xlabel("Tree Depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



As explained on the slides, we see how a decision tree might achieve high accuracy on training data (because it keeps splitting until nearly pure leaves), but this can lead to overfitting, and the test data performs horribly as the max depth continues to increase beyond a certain point.

Random Forest

What is Random Forest?

Currently: Decision Tree

MOTIVE

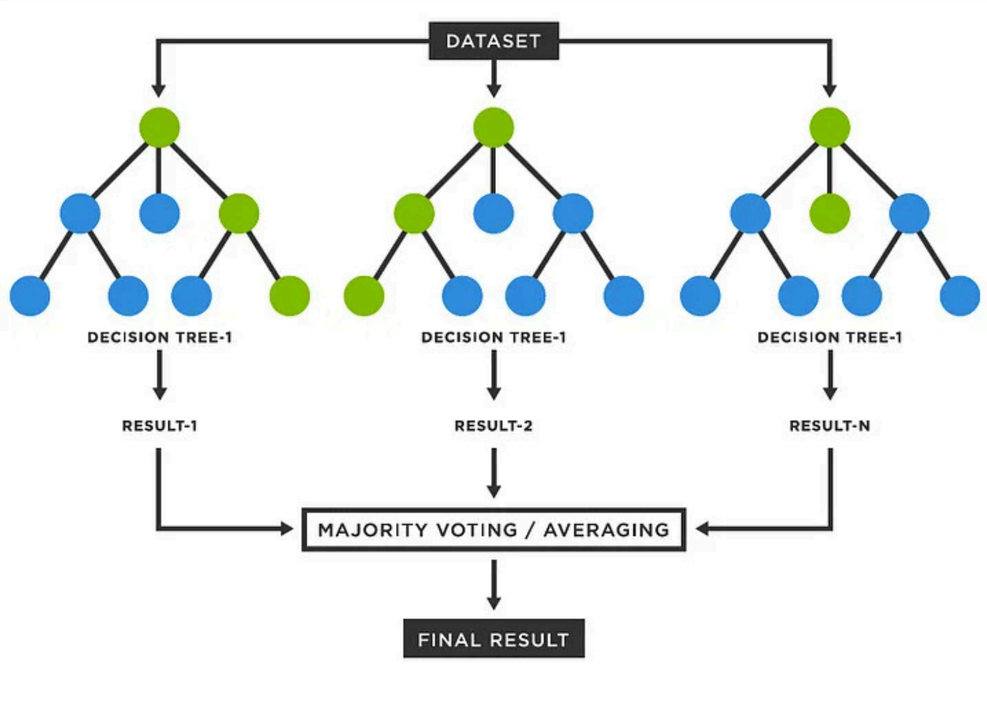
- In reality, while a single Decision Tree is easy to interpret, it is a weak learner:
  - Unstable, where small changes in data leads to big changes in the tree's predictions (due to high variance)
  - Prone to overfitting
- So: how do we make trees more stable and generalisable?
  - Combine many weak learners (Decision Trees) into a strong model!

What is Random Forest?

Now: Bagging

[CURRENTLY] Random Forest & Bagging

- Build many trees in parallel, each on a random bootstrap sample
- Combine predictions (majority vote or average)
- When averaged together, the random errors cancel out → helps to reduce variance
  -



What is Random Forest?

Boosting Vs Bagging

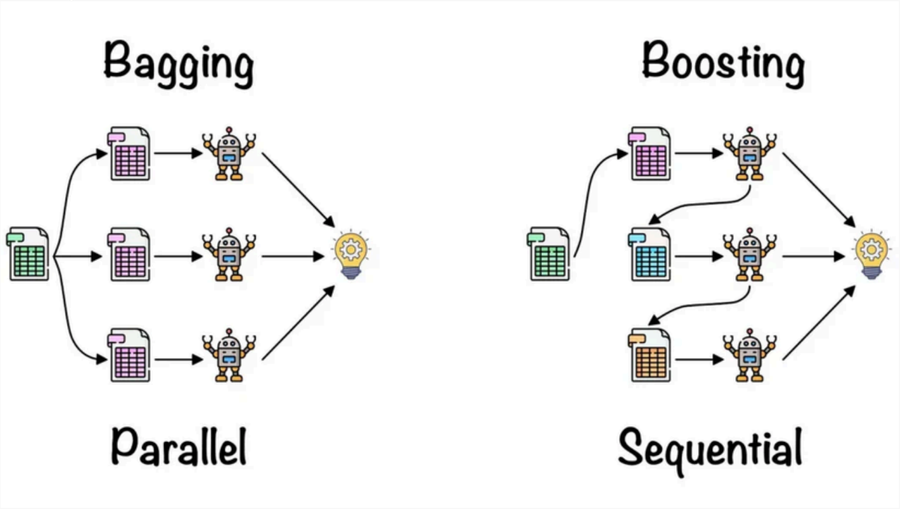
To improve upon the weak learner, we can look at:

Bagging (Bootstrap Aggregating):

- Build many trees in parallel, each on a random bootstrap sample
- Combine predictions (majority vote or average)
- Goal: reduce variance (stabilise high-variance models like decision trees)

Boosting:

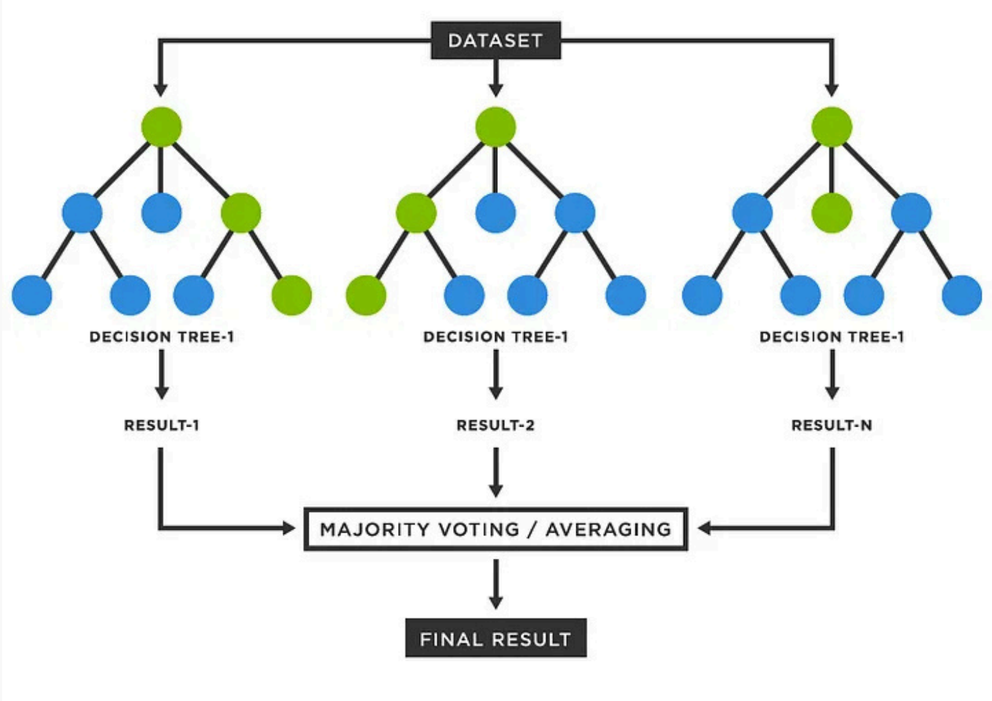
- Build trees sequentially, each one learning from the errors of the previous.
- Combine predictions additively (weighted sum).
- Goal: reduce bias and errors (make weak learners strong).



What is Random Forest?

An ensemble of decision trees, where each tree is trained on a random subset of the data (bagging) and at each split, only a random subset of features is considered.

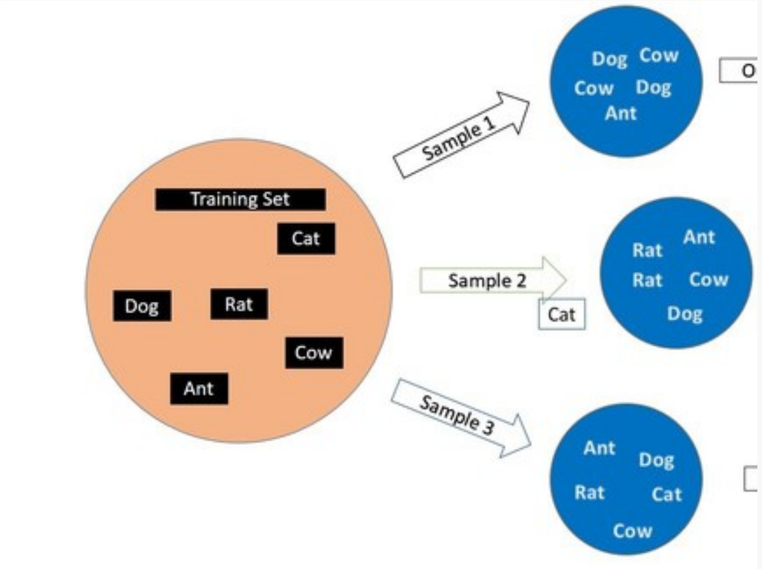
Predictions are aggregated by majority voting (classification) or averaging (regression)



What is Random Forest?

How it Works

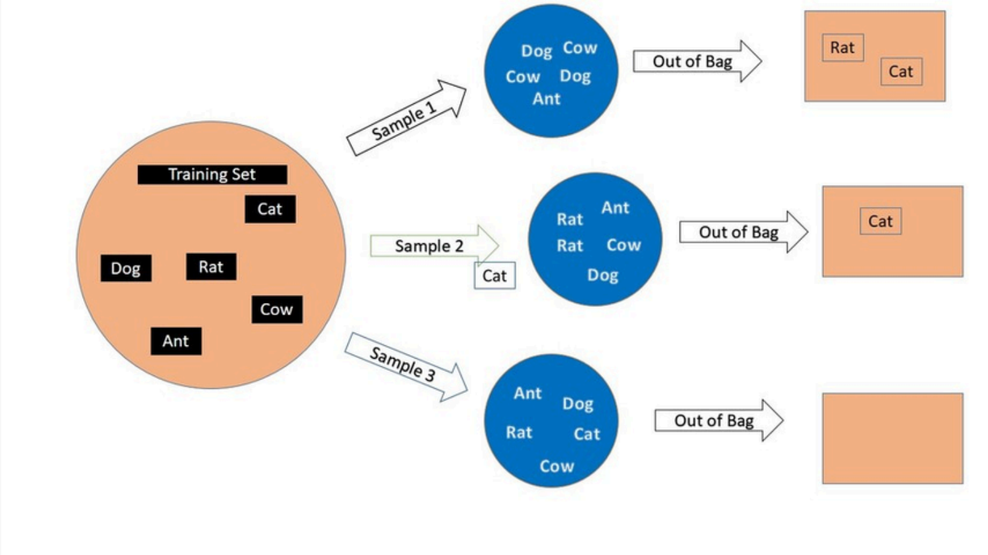
1. Bootstrap sampling: Randomly sample the training data with replacement to build multiple datasets



2. Random feature selection: At each tree split, only a random subset of features is considered
3. Train trees: Grow many decision trees (classification or regression)
4. Aggregate predictions:
    - Classification → majority vote
    - Regression → average of predictions

What is Random Forest?

Bootstrap & Out-of-Bag (OOB) Score

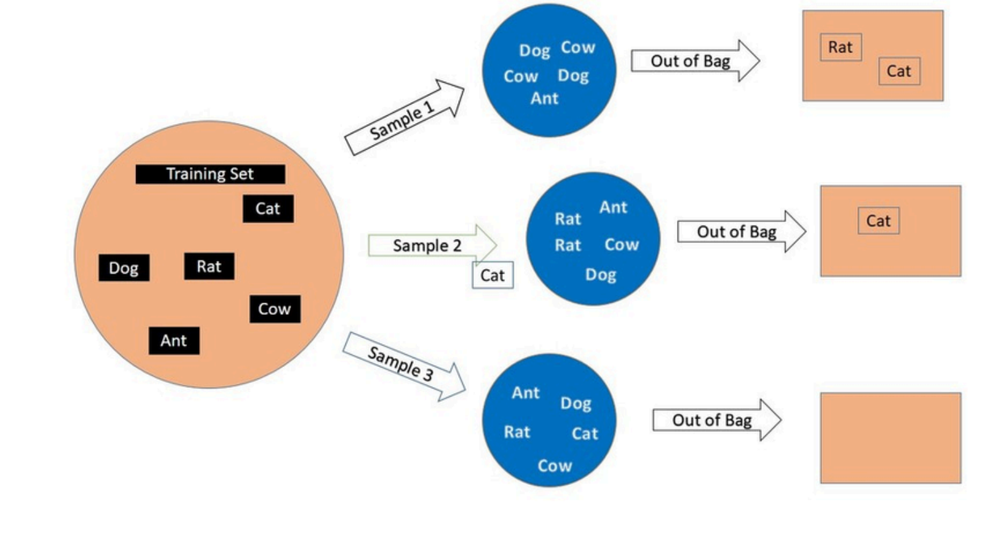


In this given example, observe that we repeated some animals while making the sample, and some animals did not even occur once in the sample.

While making the samples, data points were chosen randomly and with replacement, and the data points which fail to be a part of that particular sample are known as OUT-OF-BAG points.

What is Random Forest?

Bootstrap & Out-of-Bag (OOB) Score



- For the first tree (DT_1):
  - Training sample includes Dog, Cow, Ant → Rat and Cat are OOB for this tree.
  - DT_1 predicts Rat and Cat → their predictions are compared to true labels.
- Across many trees:
  - Each sample (e.g. “Rat”) is OOB for several trees.
  - Take the majority/average of those OOB predictions.
  - Compare to the actual label → correct or incorrect.

What is Random Forest?

Bootstrap & Out-of-Bag (OOB) Score

- The OOB Score is the accuracy of the Random Forest measured on data that each tree never saw during training.
  - If your Random Forest's OOB Score = 0.84, that means it correctly predicted 84% of the OOB samples, which is a good indicator that your forest will likely perform similarly on truly unseen test data.

$$OOBError = 1 - OOBScore$$

What is Random Forest?

Code Implemetation

[38]:
```
from sklearn.ensemble import RandomForestClassifier

# Use the same X_train, X_test, y_train, y_test as before for baseline rf
rf = RandomForestClassifier(
    n_estimators=200, #number of trees in the forest (more trees = better averaging, but slower
    max_depth = 5, #depth of each tree
    random_state=42, #seed for reproducibility
    class_weight='balanced'  # # automatically balances weights for imbalanced classes, which helps if "Yes" is
)
rf.fit(X_train, y_train)
```

By default, RandomForestClassifier in scikit-learn uses Gini impurity to decide how to split nodes.

- To use information gain (Entropy) instead, you can specify it as criterion='entropy'.

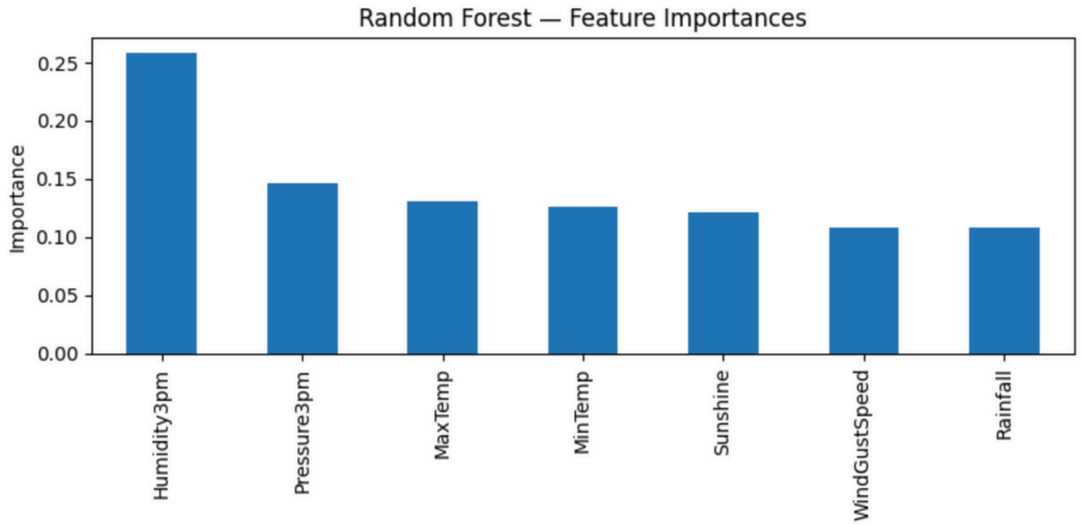
Both measure how well a split separates the classes, but entropy tends to be slightly slower and sometimes leads to more balanced trees, while Gini is faster and often yields similar accuracy.

What is Random Forest?

Code Implemetation

[38]:
```
from sklearn.ensemble import RandomForestClassifier

# Use the same X_train, X_test, y_train, y_test as before for baseline rf
rf = RandomForestClassifier(
    n_estimators=200, #number of trees in the forest (more trees = better averaging, but slower
    max_depth = 5, #depth of each tree
    random_state=42, #seed for reproducibility
    class_weight='balanced'  # # automatically balances weights for imbalanced classes, which helps if "Yes" is
)
rf.fit(X_train, y_train)
```

[33]:
```
fi = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)

plt.figure(figsize=(8,4))
fi.plot(kind='bar')
plt.title("Random Forest – Feature Importances")
plt.ylabel("Importance")
plt.tight_layout()
plt.show()
```



Here, we visualise what the Random Forest has "learned".
Note that instead of one big tree diagram, Random Forest summarises which features mattered most across all trees. High bars (e.g. Humidity3pm) are the features the model frequently split on to decide rain vs no rain.

What is Random Forest?

Code Implementation

Mirroring our approach with how we adjust depth to refine accuracy, let's adjust depth to observe how precision & recall work!

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{ActuallyPostive}$$

- Recall := Out of all actual positives, how many did we correctly identify?
  - Think of it as “How many of the real positives did Random Forest catch?”
  - High recall → fewer missed cases

What is Random Forest?

Code Implementation

Mirroring our approach with how we adjust depth to refine accuracy,
let's adjust depth to observe how precision & recall work!

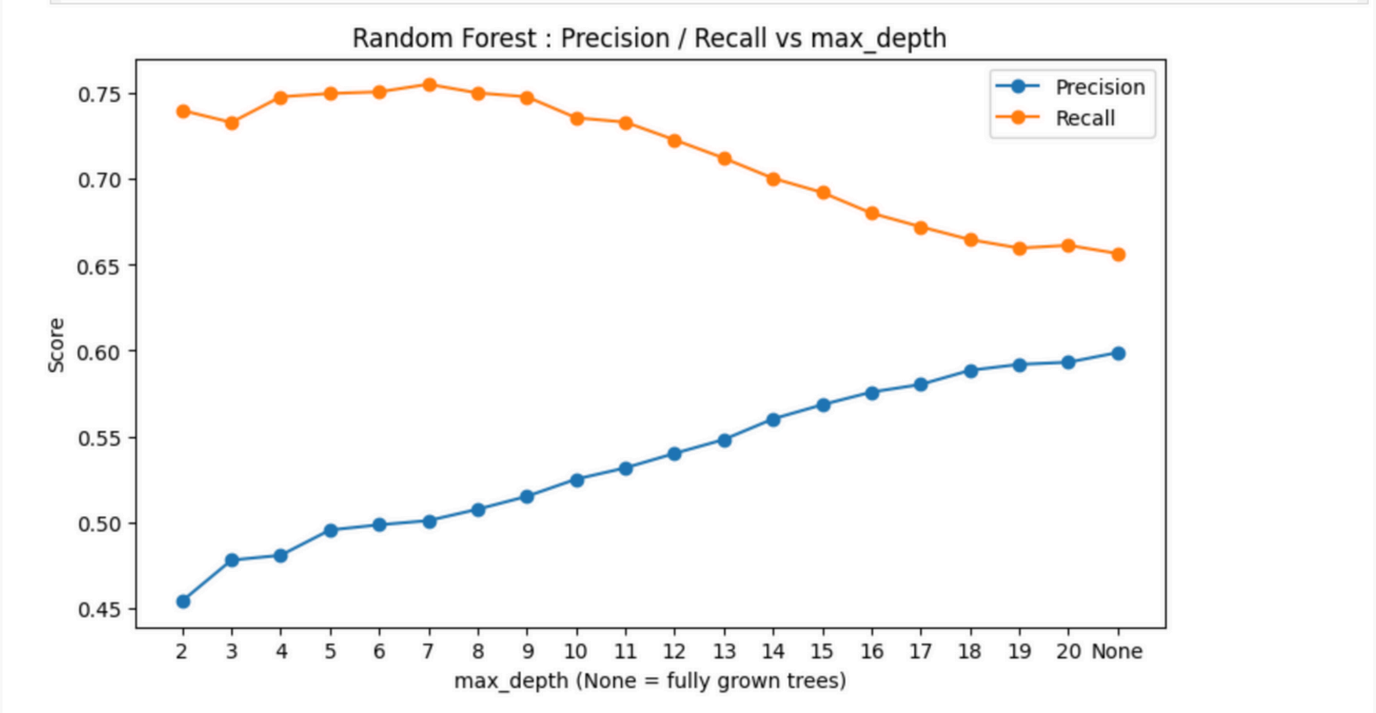
$$Precision = \frac{TP}{TP + FP} = \frac{TP}{PredictedPostive}$$

- Precision := Out of all the predicted positives in the data, how many did the forest successfully detect?
  - Think of it as “When Random Forest raises a flag, how often is it right?”
  - High precision → fewer false alarms.

What is Random Forest?

Code Implemetation

Now observe how tuning max_depth affects different error metrics



Thus, as tree depth increases, Random Forest experiences higher precision but lower recall, implying that it becomes better at being right when it predicts rain, but misses more actual rainy days. The goal is to find a depth that balances both, capturing most rainy days without overfitting.

Boosting

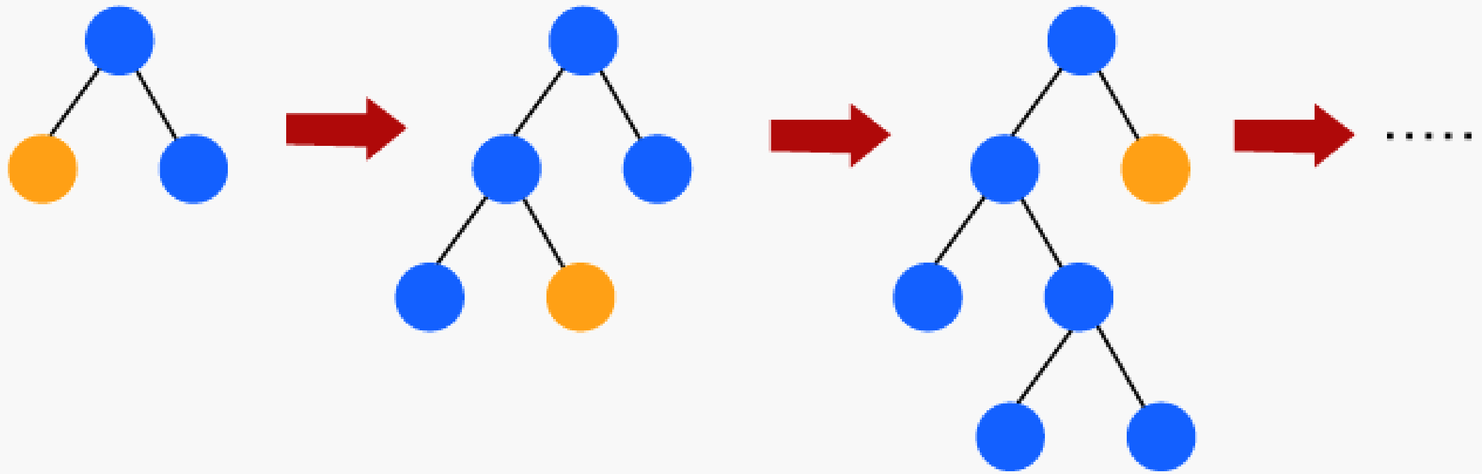
What is Boosting?

An ensemble method where trees are built sequentially, and each new tree is trained to correct the errors of the previous model

---

Example

Netflix Recommendation System → first builds a simple model to predict how much you'll like a movie. Then, it builds new trees that focus on the errors, movies it predicted wrong. Each new tree corrects the previous mistakes, and together they form a stronger, more accurate model for more personalized recommendations!



What is Gradient Boosting?

How it Works

Gradient Boosting is an algorithm that gradually improves its performance!

1. Start with an initial model to predict the target variable y
   - Regression → mean of target values
   - Classification → class prior (log-odds)

2. Compute residuals (errors, the difference between each observed value & the initial prediction):
   - For this specific example, the initial prediction is set as the model predicts all purchases to be $156

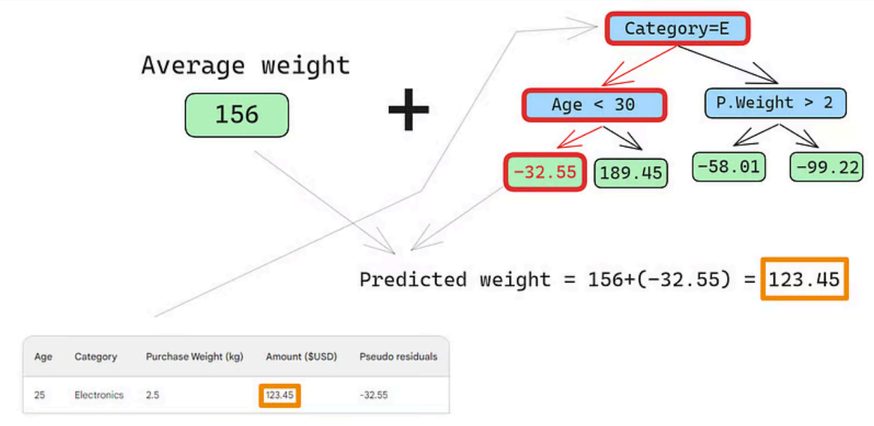
$$Error = (y_i - \hat{y}_i)$$

Age	Category	Purchase Weight (kg)	Amount ($USD)	Pseudo residuals
25	Electronics	2.5	123.45	-32.55
34	Clothing	1.3	56.78	-99.22
42	Electronics	5.0	345.67	189.45
19	Homeware	3.2	98.01	-58.01

What is Gradient Boosting?

How it Works

3. Build a decision tree (weak learner) that predicts the residuals using the three features we have. After the tree is fit to the data, we make a prediction for each row in the data, and we will now have new predictions if we calculate the original data points into the modified tree!



Age	Category	Purchase Weight (kg)	Amount ($USD)	Pseudo residuals	New Predictions
25	Electronics	2.5	123.45	-32.55	152.745
34	Clothing	1.3	56.78	-99.22	146.078
42	Electronics	5.0	345.67	189.45	174.945
19	Homeware	3.2	98.01	-58.01	150.199

What is Gradient Boosting?

How it Works

The new predictions are formed upon a learning rate.

- Learning rate is a value between 0 and 1 that scales the prediction of each weak learner to generalise the model to prevent overfitting.

Age	Category	Purchase Weight (kg)	Amount ($USD)	Pseudo residuals	New Predictions
25	Electronics	2.5	123.45	-32.55	152.745
34	Clothing	1.3	56.78	-99.22	146.078
42	Electronics	5.0	345.67	189.45	174.945
19	Homeware	3.2	98.01	-58.01	150.199

```
eta = 0.1

Predicted weight = 156+0.1*(-32.55) = 152.75
```

When we add an arbitrary learning rate of 0.1 into the mix, our prediction becomes 152.75, and the general formula is as such:

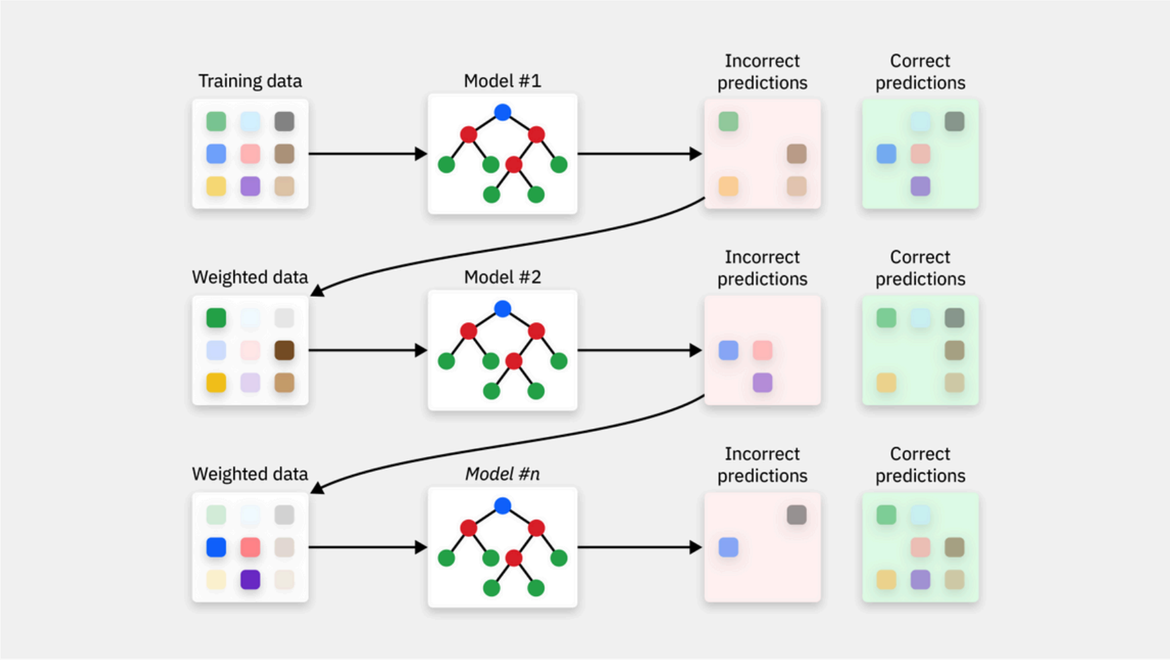
$$Ynew = Yold + (learningrate \times residual)$$

What is Gradient Boosting?

How it Works

4. Next, we find the new pseudo-residuals by subtracting new predictions from the purchase amount. Our pseudo-residuals are now smaller, which means our loss is going down.

5. Lastly, iterate on step 3 by building more weak learners → keep adding the residuals of each tree to the initial prediction to generate the next!



What is Gradient Boosting?

XGBoost

- XGBoost
  - Uses the same idea : building trees sequentially to correct previous errors, but it’s faster, more efficient, and more accurate thanks to several computational enhancements.
  - We’ll be using this today!

What is XGBoost?

Code Implemetation

[53]:

```
from xgboost import XGBClassifier

# Encode labels: 'No' → 0, 'Yes' → 1
y_train_enc = (y_train == "Yes").astype(int)
y_test_enc  = (y_test == "Yes").astype(int)
```

Before we start, we need to encode the labels for classification! Unlike previously with decision tree & RF classifier which can handle string labels like "Yes" and "No", XGBoost expects the target y to be numeric (0/1) for classification tasks.

What is XGBoost?

Code Implemetation

```python
# baseline XGBoost model
xgb = XGBClassifier(
    n_estimators=200,       # number of boosting rounds (trees)
    learning_rate=0.1,      # step size shrinkage
    max_depth=5,            # depth of individual trees (controls complexity)
    subsample=0.8,          # use 80% of samples per tree (reduces overfitting)
    colsample_bytree=0.8,   # use 80% of features per tree
    random_state=42,
    eval_metric='logloss', # standard classification loss
    n_jobs=-1
)

# Train on the training set
xgb.fit(X_train, y_train_enc)

# Predict on the test set
y_pred_enc = xgb.predict(X_test)
y_prob = xgb.predict_proba(X_test)[:, 1]

# Evaluate
print(f"Accuracy : {accuracy_score(y_test_enc, y_pred):.3f}")
print(f"Precision: {precision_score(y_test_enc, y_pred, pos_label=1):.3f}")
print(f"Recall   : {recall_score(y_test_enc, y_pred, pos_label=1):.3f}")
print(f"F1-score : {f1_score(y_test_enc, y_pred, pos_label=1):.3f}")
```

```
Accuracy : 0.849
Precision: 0.722
Recall   : 0.494
F1-score : 0.587
```

F1-score (harmonic mean of precision & recall) = 0.587

- **Precision = 0.72 → When the model predicts “rain,” it’s right about 72% of the time.**
- **Recall = 0.49 → It successfully catches about half of the actual rainy days.**

Together, the F1-score (0.587) shows a moderate balance; while precision is decently high, it is weighed down by the lower recall

What is XGBoost?

Metric	Random Forest	XGBoost
Accuracy	0.776	0.849
Precision	0.489	0.722
Recall	0.749	0.494

So which is better?

Random Forest:

- **High recall but low precision → more lenient (good for catching positives but many false alarms).**

XGBoost:

- **High precision but lower recall → more conservative (misses some true positives).**

Thus, it depends on what matters more for your use case! Recall our goal is to predict whether it will rain the next day

What is XGBoost?

Metric	Random Forest	XGBoost
Accuracy	0.776	0.849
Precision	0.489	0.722
Recall	0.749	0.494

Analysing the use case

- If doing general weather prediction (like in a forecasting dashboard), we likely prefer XGBoost because users care more about accuracy and fewer false positives (“if it says it’ll rain, I can trust it”) than about catching every possible drizzle
- BUT if your model feeds into like a warning system or risk alert (e.g. disaster management), Random Forest might be better as missing rain could have high cost
- Ultimately, we observe how there is a need to balance different error metrics due to trade-offs!

Unsupervised Learning

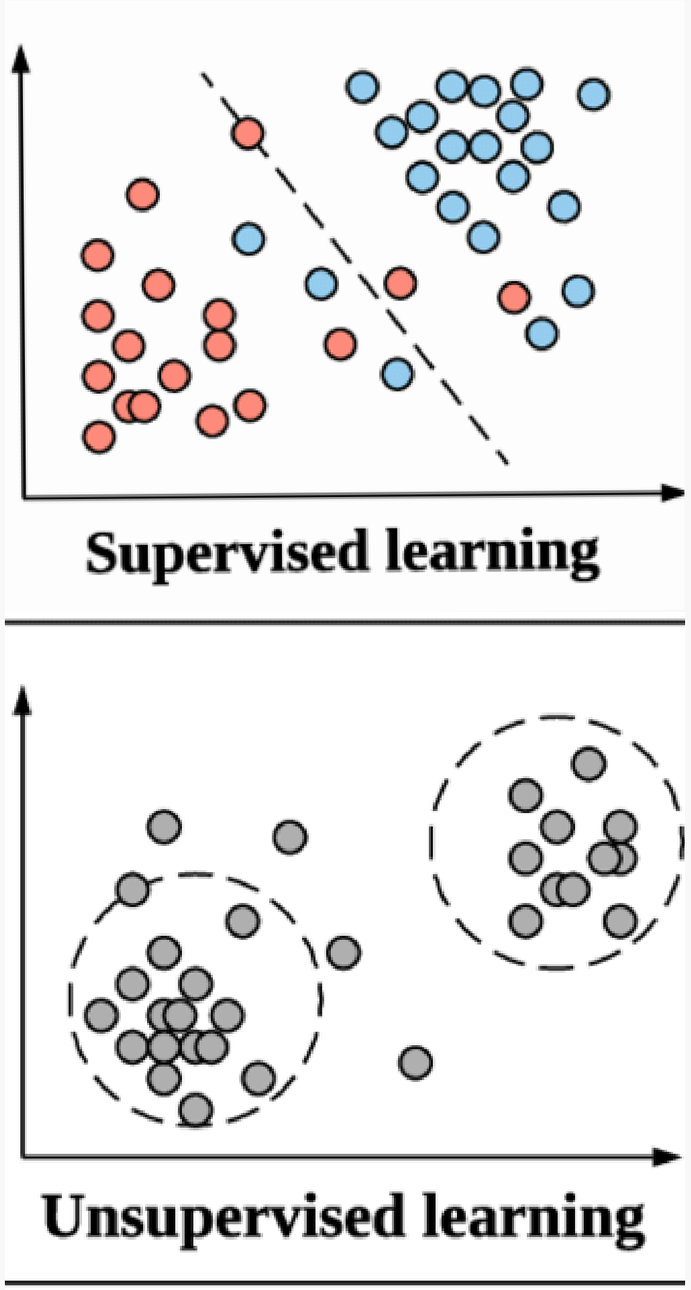
What is Unsupervised Learning

Unsupervised learning uses no labeled data (Y) . The goals of unsupervised learning are to:

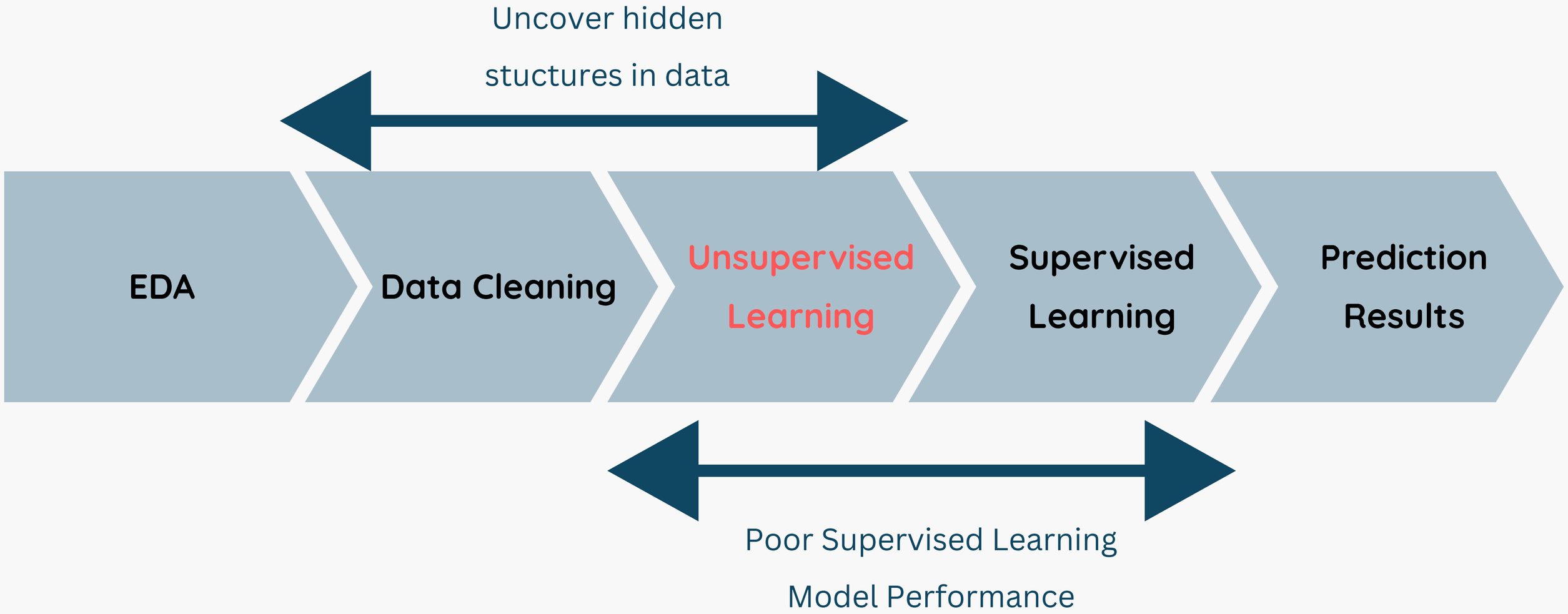
- Find similar patterns that can be grouped into specific classes or events.
- Categorise similar input (X) data together.

Examples:

- Genomics - Gene Expression Clustering
- Computer vision - Image compression, Feature learning



Where Unsupervised Learning Fits In



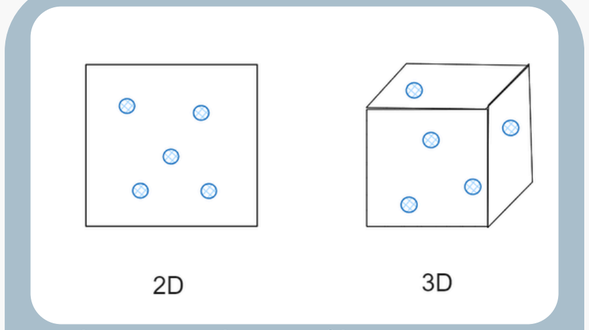
Types of Unsupervised Learning

Clustering

- Grouping similar input data together

Models:

- K-Means
- Hierarchical
- DBSCAN

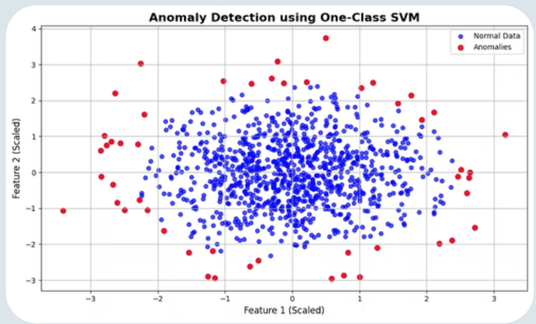


Dimensionality Reduction

- Finding key features from the data

Models:

- **PCA**
- t-SNE
- **UMAP**

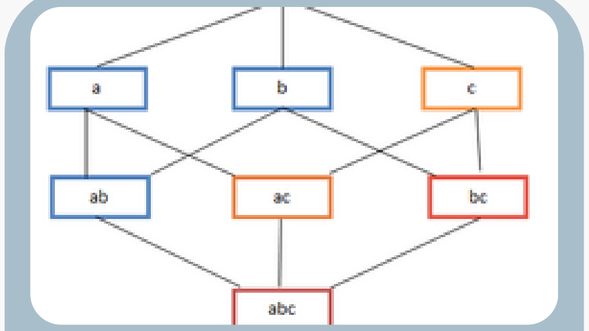


Anomaly Detection

- Finding data that deviate significantly from the majority of the data

Models:

- Local Outlier Factor
- Unsupervised KNN
- K-Means
- Isolation Forest



Association Rule Mining

- Discover inherent patterns and relations within dataset without prior knowledge of target outcomes

Models:

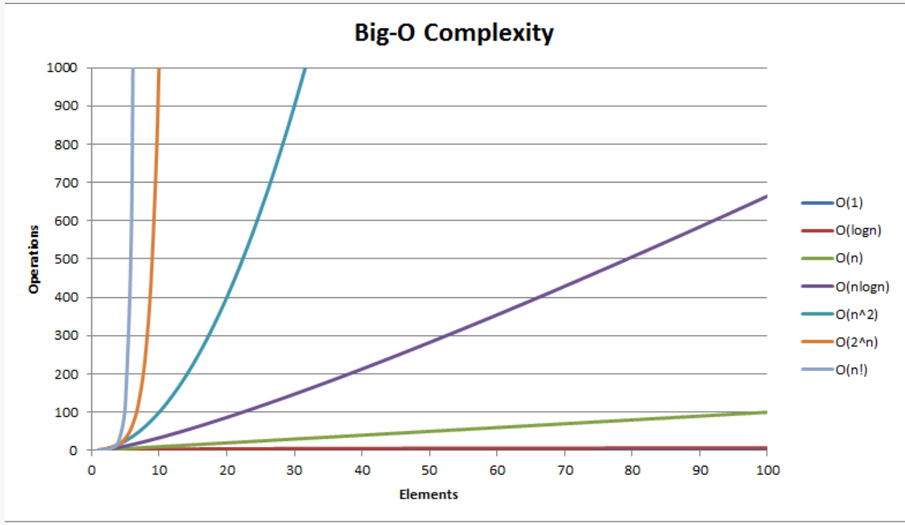
- Apriori
- FP-Growth

Why Focus on Dimensionality Reduction?

Reduces Computational Requirements

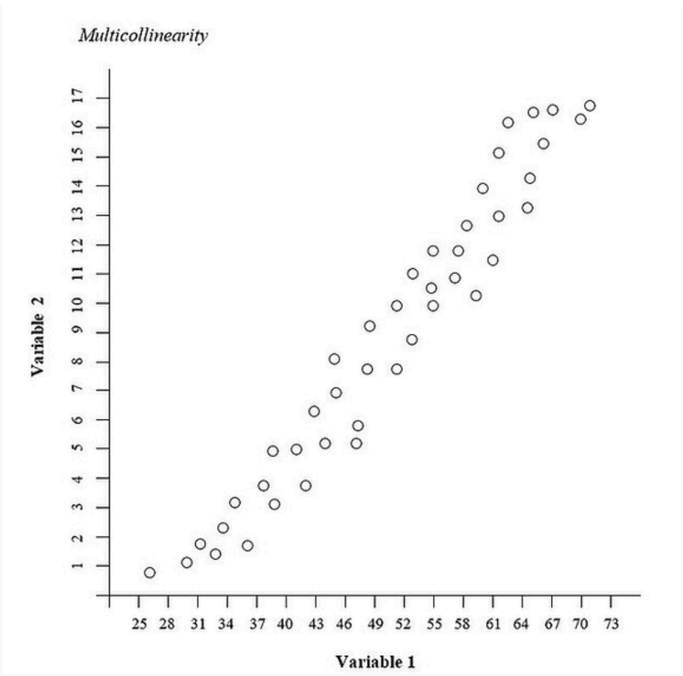
The “Curse of Dimensionality”: Data with more dimensions results in more expensive computation and more errors

- More dimensions → more computation → longer run time
- Concentration of Distance phenomenon → Euclidean Distance loses meaning



Capturing As Much Variance in As Little Dimensions As Possible

- Problem of multicollinearity → 2 or more variables are highly correlated with each other
- Model cannot determine which X variable is responsible for causing the variation in the target variable Y.
- Makes it difficult to determine the independent effect of each variable on the outcome.



PCA

Principal Component Analysis

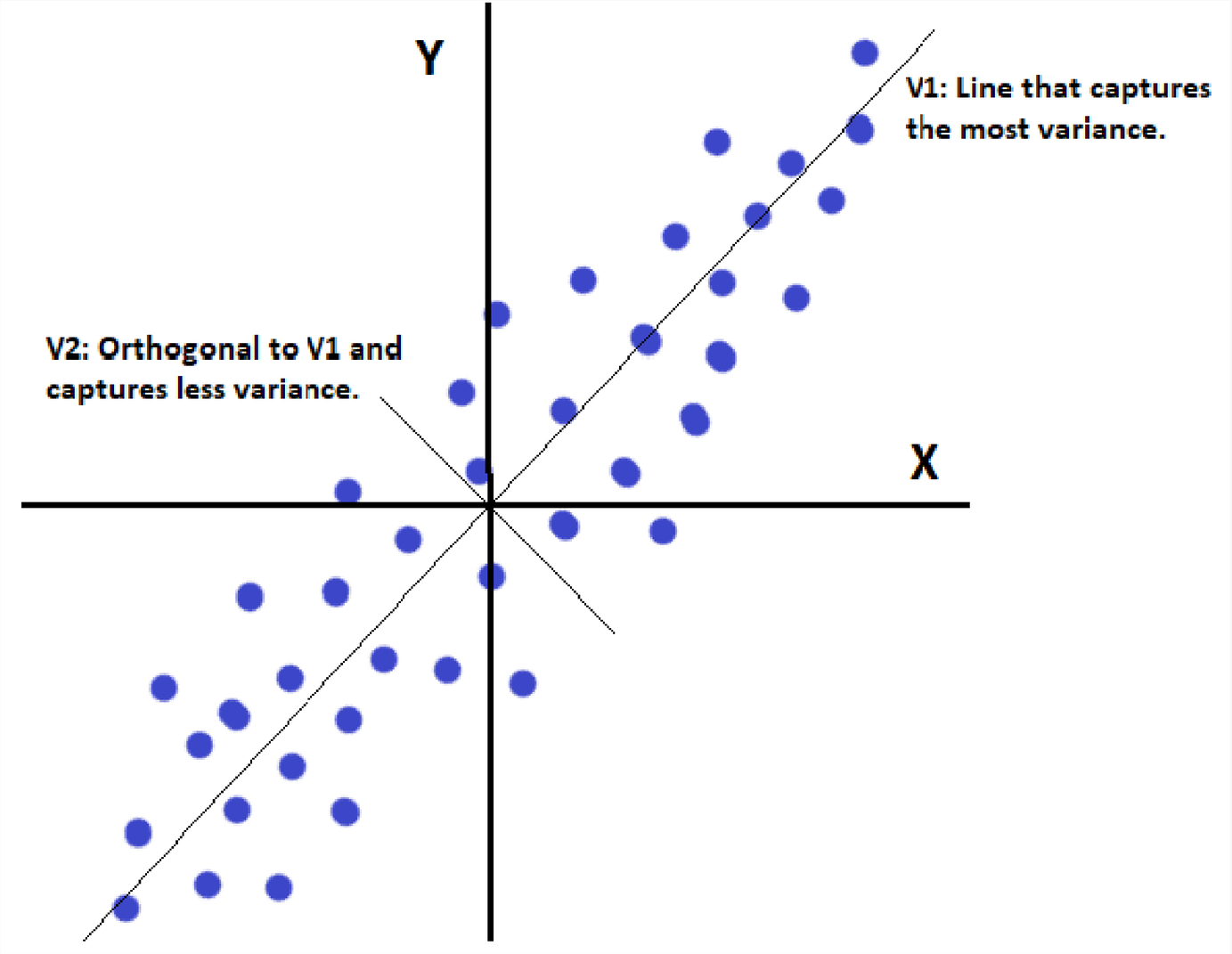
Aim: To summarise a large set of data into uncorrelated variables

- We want to create new uncorrelated variables corresponding to the directions along which the data vary the most
- These new variables are linear combinations of the original features.
- These new variables are called principal components
- Principal components are ordered according to how much variation in the data it can explain

An Intuitive Visualisation

Describing Data

- To find each data point I need 2 coordinates
  - x-coordinate
  - y-coordinate
- I need two variables to describe this data



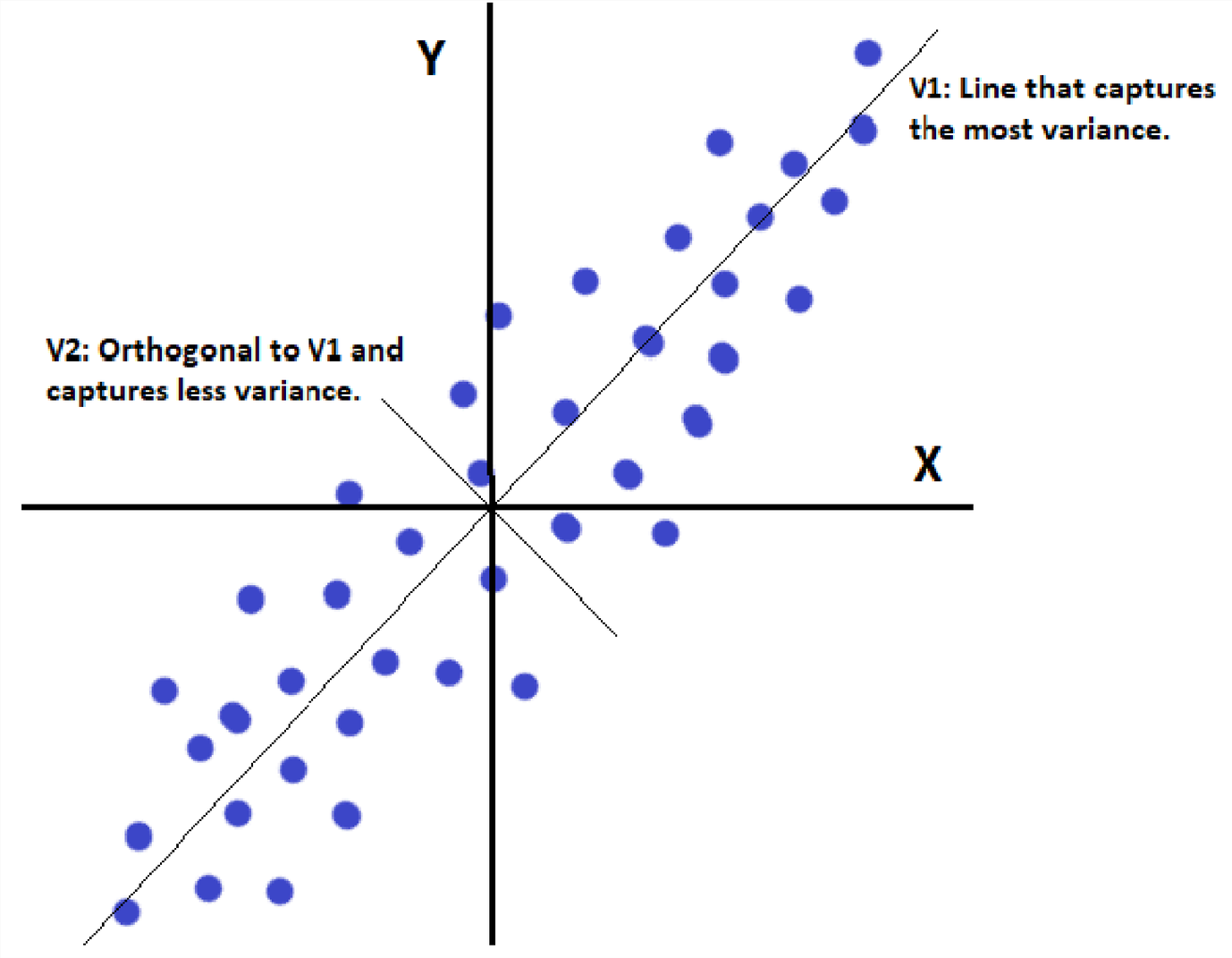
An Intuitive Visualisation

What If I only want to use 1 variable?

- I can use vectors to describe this data
- x-axis is represented by the vector $\begin{pmatrix}1\\0\end{pmatrix}$
- The line y = x is represented by the vector $\begin{pmatrix}1\\1\end{pmatrix}$
- Treat $\begin{pmatrix}1\\1\end{pmatrix}$ as my new axis $\begin{pmatrix}1\\1\end{pmatrix}$

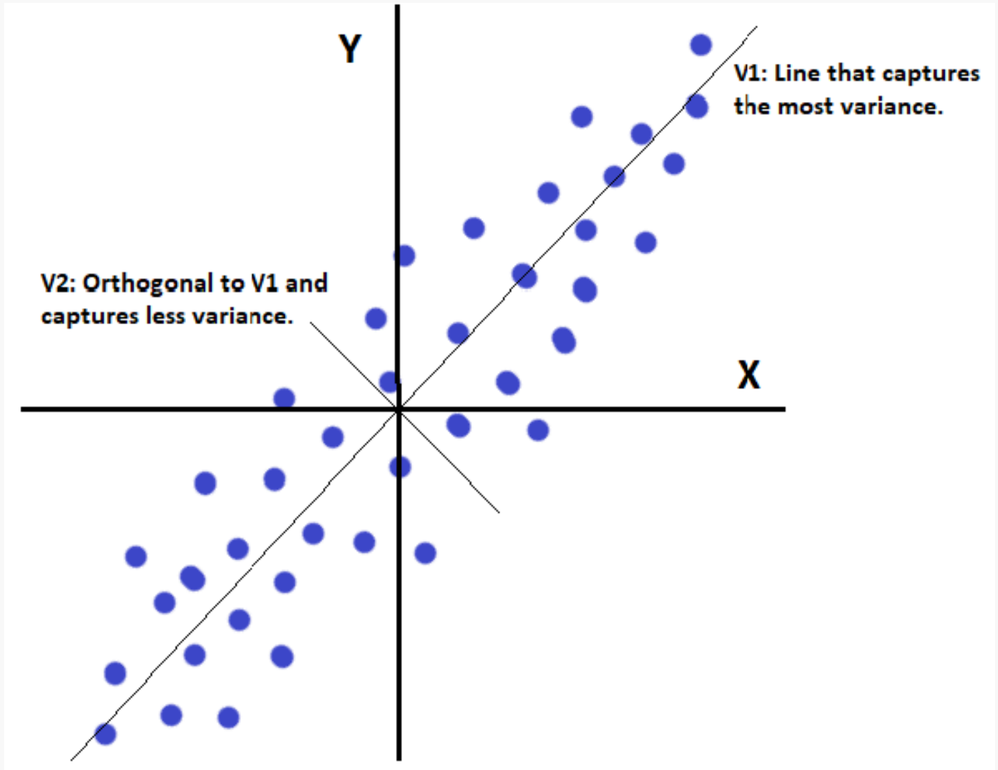
How does this help?

- Now I only need 1 number to get as close to the data point as possible.
- Congratulations you have done dimensionality reduction



Note: Don't worry about understanding changing axes for now.
More to come in **MA2001**

An Intuitive Visualisation

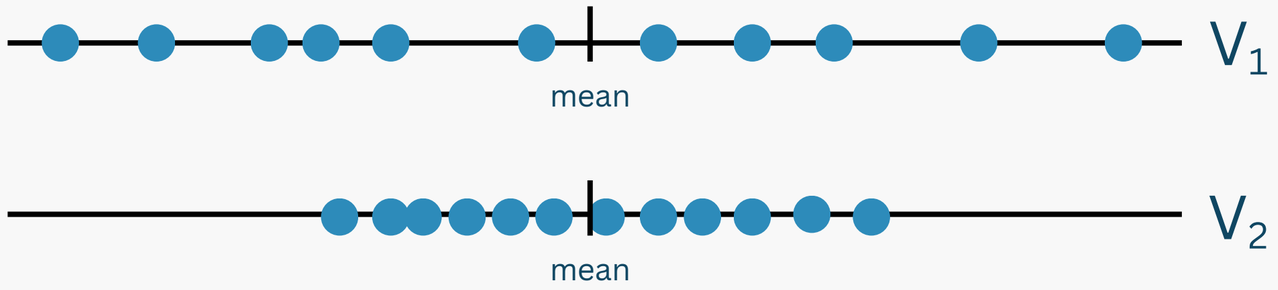


How do I know what line to choose?

- Choose the direction along which data varies the most

Variance

- Variance is the average squared distance from the mean
- Intuitively, it is the how widely the data is spread out from its mean along a direction



Under the Hood

1. *Calculate Covariance Matrix of the Dataset*

a. Calculate the mean for each variable and subtracting the mean from each datapoint. This step centres our data.

b. Calculate the Covariance Matrix C

$$C = \frac{1}{n-1} X^T X$$

2. Find the Eigenvectors of the Covariance Matrix

a. This is a common task in Linear Algebra

b. You will learn techniques in MA2001, DSA2102, etc.

3. Sort the Eigenvectors in descending order based on their Eigenvalues

a. These Eigenvectors are the principal components

4. Create a matrix to project the data onto the principal components

X =

Data	X_1	X_2	X_3
1	...	...	...
2	...	...	...
3	...	...	...

v is an Eigenvector if: $Cv = \lambda v$

These are vectors whose directions do not change even though the matrix is applied to them

λ Is known as an Eigenvalue

More to come in DSA2102!

Under the Hood

What happens when we run PCA on the Rain Dataset?

$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots\ldots + a_{1,22}x_{22} = z_1$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \ldots\ldots + a_{2,22}x_{22} = z_2$$
$$a_{3,1}x_1 + a_{3,2}x_2 + \ldots\ldots + \underbrace{a_{3,22}}x_{22} = z_3$$
$$\vdots$$
$$a_{22,1}x_1 + a_{22,2}x_2 + \ldots\ldots + a_{22,22}x_{22} \underbrace{= z_{22}}$$

The value $a_{i,j}$ is known a loading

Principal Components

1. Each principal component is a linear combination of the variables.
2. The dataset has 22 features → maximum 22 Principal Components

Principal Component Analysis

Code Implemetation

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df) # standardise the dataset
df_scaled = pd.DataFrame(df_scaled, columns=df.columns, index=df.index)
df_scaled.head()
```

Standardising is important in PCA. If not standardised, features with larger scales tend to dominate the principal components regardless of their importance

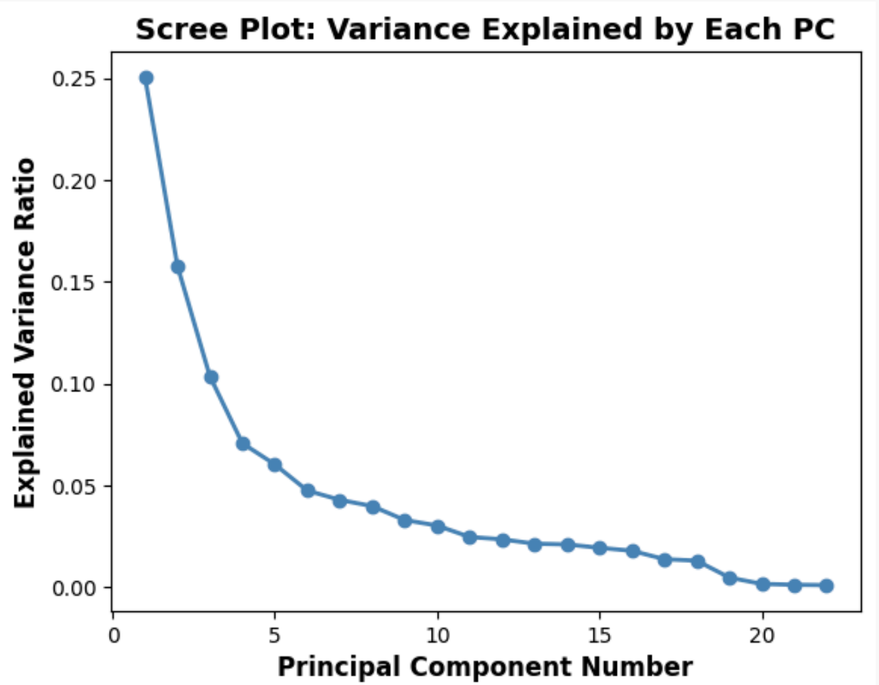
```
from sklearn.decomposition import PCA # PCA library in python
import matplotlib.pyplot as plt

pca = PCA(n_components = 22, random_state=1000) # there are 22 X-columns in my cleaned dataset. Max PCs = 22
pca.fit(X_small) # this function does the PCA decomposition for you
```

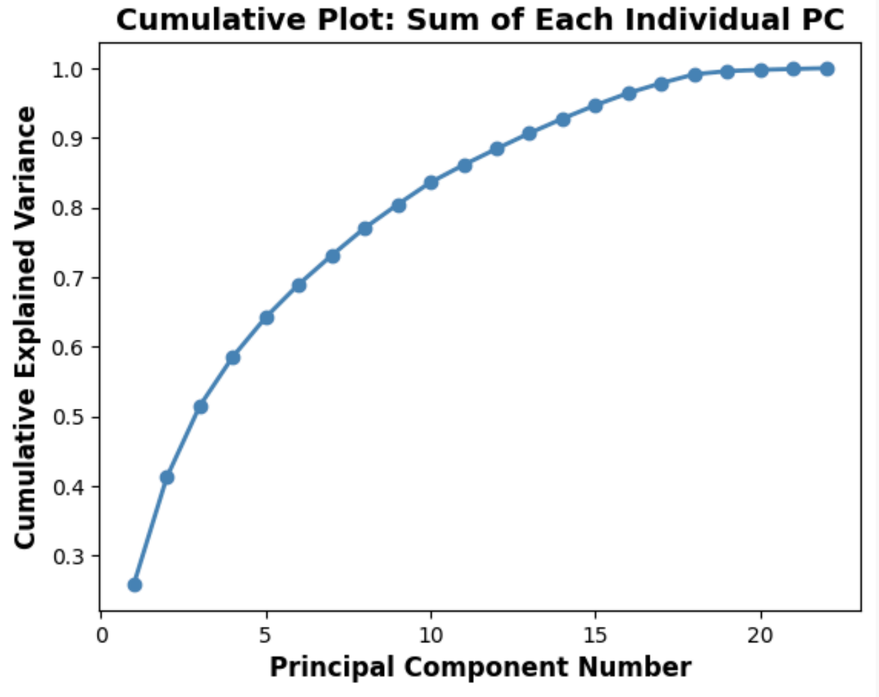
This function does all the computation for you!

Scree Plot

What is a Scree Plot?

- A scree plot shows the percentage of variance that is explained by each Principal Component
- Principal Components are arranged in descending order.
- The sum of the percentage of variance explained by each PC should total to 1
- Double check with the cumulative plot to ensure that the percentage of variance explained sums to 1

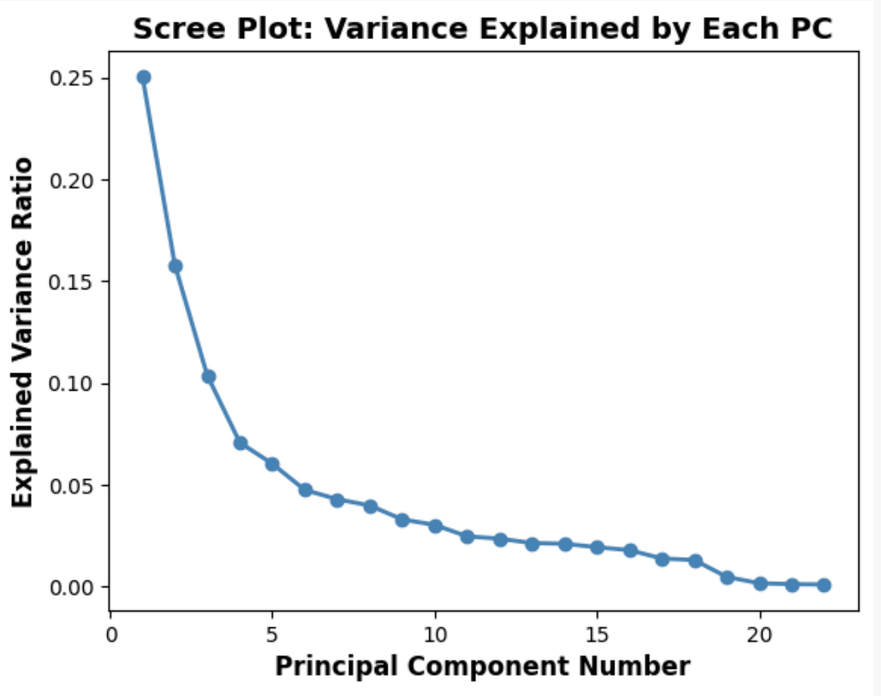




Scree Plot

What to do next?

- There is no point in using all 22 Principal Components → There is no dimensionality reduction!
- Our goal is to find the fewest principal components that retain as much of the original data's variability (information) as possible.
- We use the elbow point as a guide, in the case of the rain dataset: the elbow point is 4



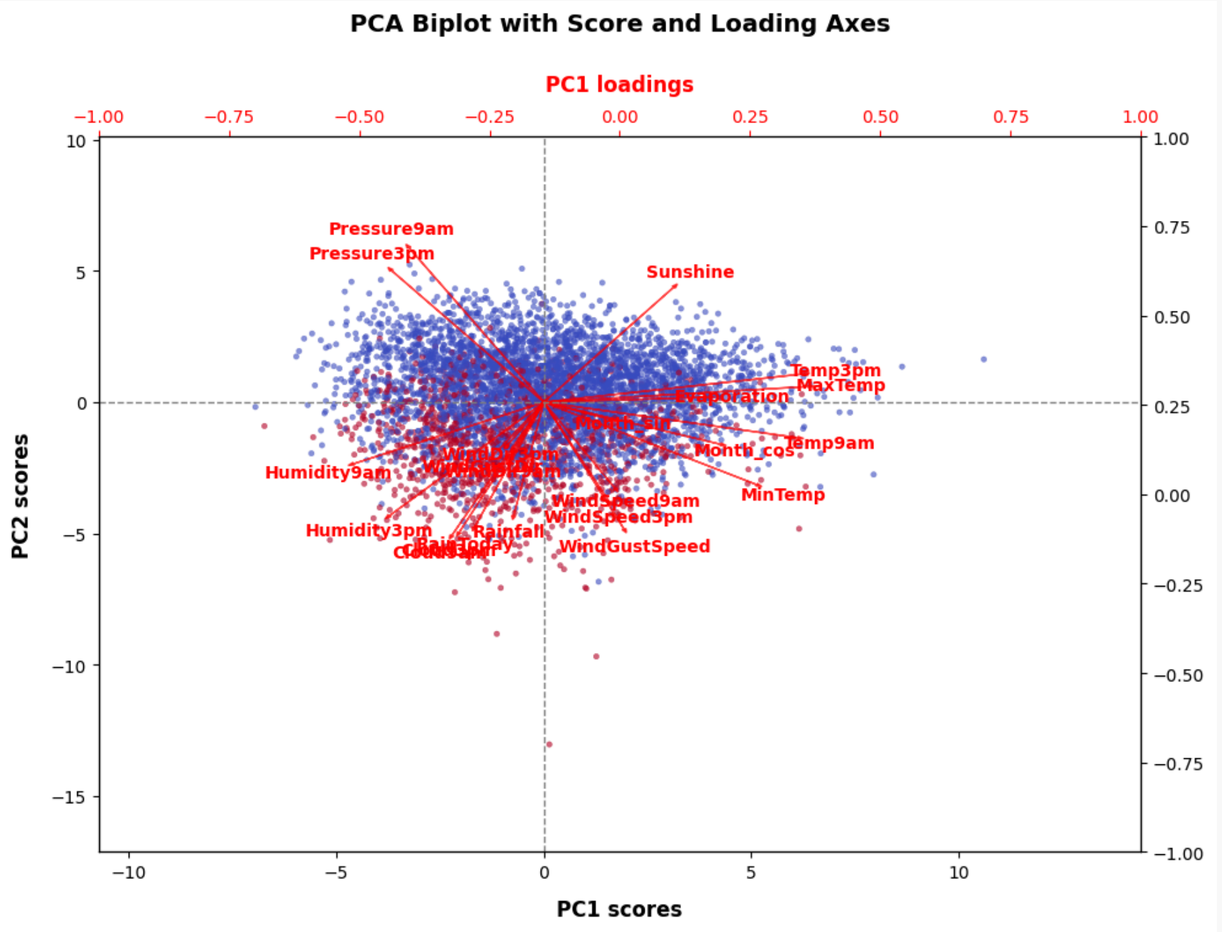
Biplot

What is a Biplot?

- Plots all your data points along the first two principal components
- Shows you the loadings of each feature with respect to the principal components

Interpretation

- Principal component scores and loading weights
- Clustering of our y-data
- Correlation of original features.
- Heuristic: Angle between features



0: Perfect Correlation | 90: Uncorrelated | 180: Negative Correlation

Biplot

Put on your detective thinking caps

PC1 - Temperature cycle Component

- Features with High |PC1| loadings:

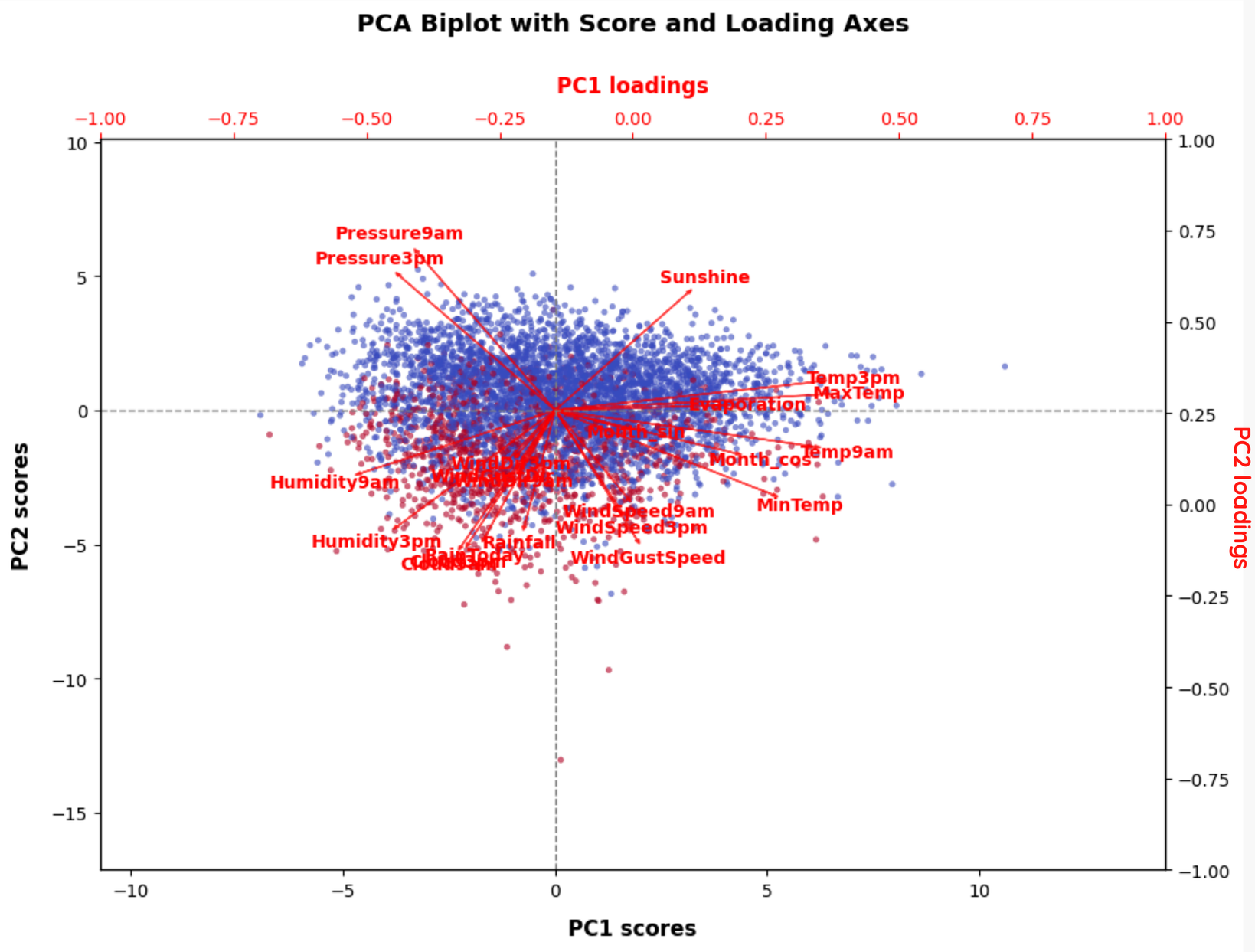
  MaxTemp, Humidity

PC2 - Atmospheric Variability

Component

- Features with High |PC2| loadings:

  Pressure, Windspeed



Limitations

PCA struggles with non-linear data

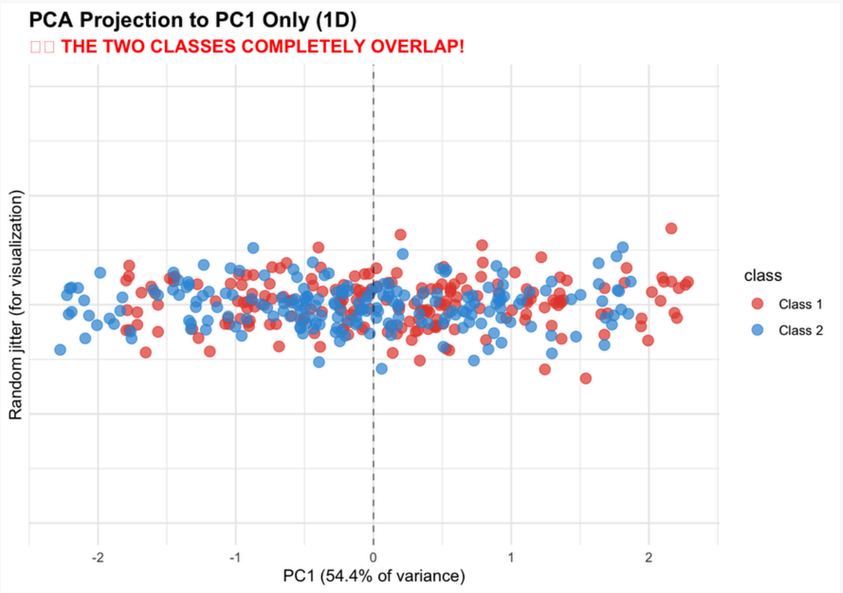
- Even while visually the two classes are completely distinct
- When we apply PCA to reduce the dimension, it causes data from the two classes to overlap

Non-linear

- Data that do not lie on a flat hyperplane, plane or line.
- Examples of linear:
  - $z = x + y$, (3D)
- Examples of non-linear:
  - $x^2 + y^2 + z^2 = 1 \rightarrow$ sphere

Solutions

- Try Kernel PCA
- t-SNE or UMAP



t-SNE

t-distributed Stochastic Neighbour Embedding

Aim: To project data from a high dimension space into a lower dimension space while preserving similarity relationships.

t-distribution	used in calculation of similarity between points in the low-dimension space
stochastic	using probabilities to represent similarity (instead of saying these points are 5.2 units apart, we interpret the two points as 85% chance of being neighbours)
neighbour	focus on preserving local relationships instead of global distances
embedding	representing data from high dimensions to data in low dimensions

Key intuition: Points in a high dimension space should still be close to each other in low dimensions

Step 1: Measuring Similarity in High Dimensions

Process:

1. Calculate the Euclidean distance between any two points
2. Plot the distance on the Gaussian Distribution curve
3. Find the similarity probability

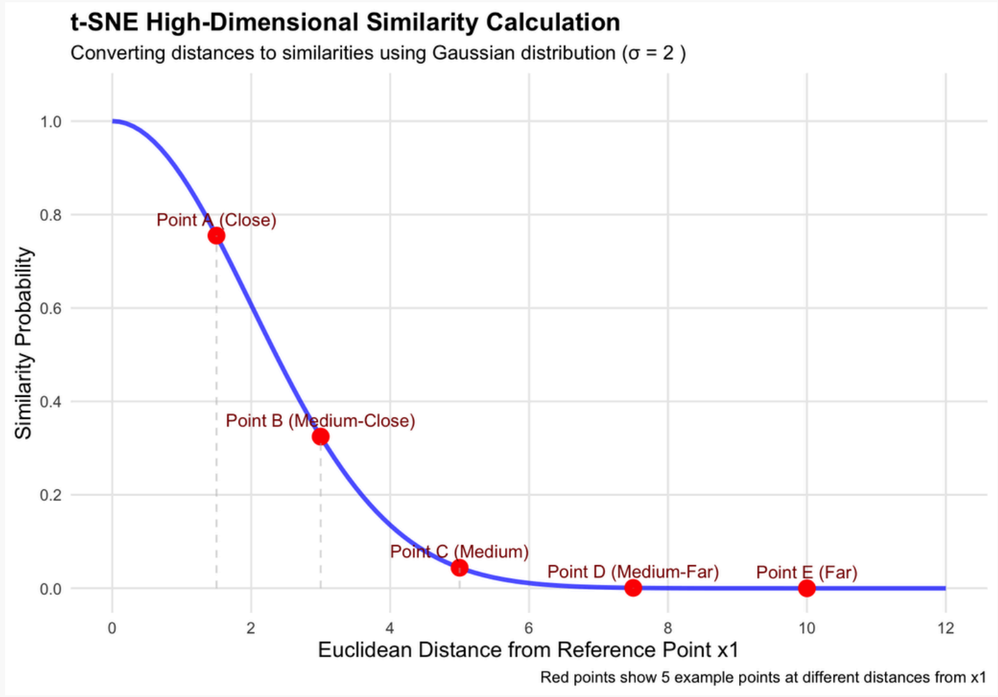
Euclidean distance → $d_{ij} = \left(\sum_{k=1}^{n} \left(x_{ik} - x_{jk}\right)^2\right)^{\frac{1}{2}}$

Unnormalised Gaussian Density Function → $p_{0,j} = e^{\left(-\frac{\|x_0 - x_j\|^2}{2\sigma_0^2}\right)}$

Rationale:

1. Using probabilities instead of the raw Euclidean distance because pairwise distances become similar in high dimension (concentration of distance phenomenon). t-SNE overcomes this by focussing on preserving local relationships.
2. Omit the normalisation constant in the density function because it will get cancelled out in the conditional probability step

Data	Distance from X_1	Similarity
1	1.5	0.755
2	3	0.325
3	5	0.0439
4	7.5	0.000883
5	10	0.00000373



Step 1: Measuring Similarity in High Dimensions

Hyperparameter: Perplexity

Caveats with Gaussian Distribution

- Variance affects the shape of the Gaussian Distribution → σ^2
- A Gaussian Distribution with a smaller variance assigns a higher probability to neighbours that are closer and a lower probability to neighbours that are further → lesser data points will be considered neighbours
- How do we choose a good σ ?

Finding σ

Gaussian density function → $p_{0,j} = e^{\left(-\frac{\|x_0 - x_j\|^2}{2\sigma_0^2}\right)}$

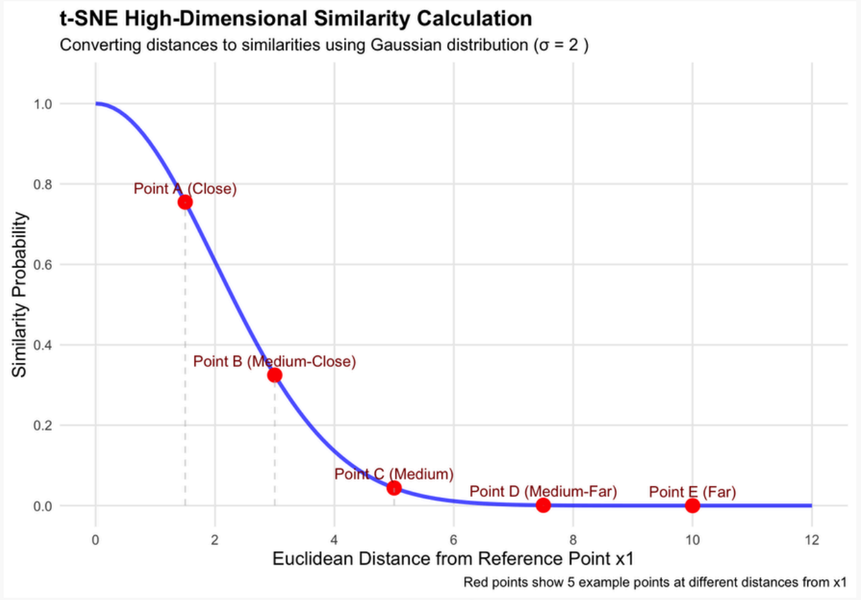
We now know the probabilistic distance of each data to X_0

We want all our probabilities to sum to 1, so we normalise each individual probability using conditional probability

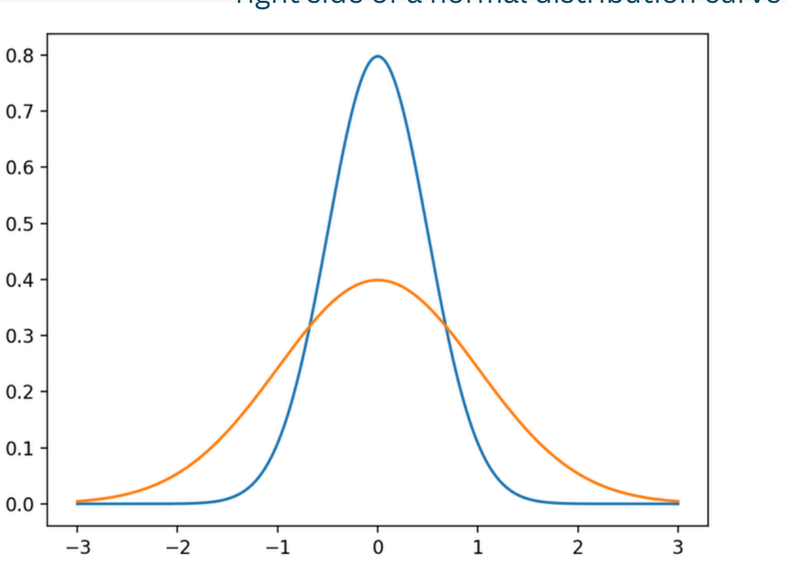
Unnormalised Gaussian Density Function

Conditional probability → $p_{i,j} = \frac{exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)} = P(A|B) = \frac{P(A \cap B)}{P(B)}$

Sum of all probabilistic distance from X_i



right side of a normal distribution curve



Step 1: Measuring Similarity in High Dimensions

Hyperparameter: Perplexity

Finding σ Continued

Entropy: measures the impurity or uncertainty present in the data

Analogy:
In a jar, I have 95 red marbles and 5 blue marbles. If I take a marble out by random, it is almost certain (P_R = 0.95) that the marble is red.
The entropy of this distribution is **LOW**

In a jar, I have 25 red marbles, 25 blue marbles, 25 yellow marbles and 25 green marbles. If I take a marble out by random, I am unsure if the marble will be red (P_R = 0.25).
The entropy of this distribution is **HIGH**

Perplexity

$$Perplexity = 2^{H(P_i)}$$

Perplexity gives you the number of effective neighbours for a data point

$Perp\,(P_0) = 2^{1.071} = 2.1\,(2d.p.)$ ← X_0 has effectively 2.1 neighbours

In practice, the algorithm iteratively chooses a few σ until the perplexity is approximately equal to the value chosen by the user

t-SNE worked example

$$H\,(P_i) = -\sum p_{j|i} log_2\,(p_{j|i})$$

Point A	0.8
Point B	0.1
Point C	0.05
Point D	0.03
Point E	0.02

$$H\,(P_0) = -\,(0.8\,(log_2\,(0.8)) + (0.1\,(log_2\,(0.1)) + \ldots. = 1.071$$

You are very certain that Point A is main neighbour of X_0

Entropy should be low

Point A	0.6
Point B	0.2
Point C	0.1
Point D	0.07
Point E	0.03

$$H\,(P_1) = -\,(0.6\,(log_2\,(0.6)) + (0.2\,(log_2\,(0.2)) + \ldots. = 1.659$$

Probabilities for this set is more spread out

Entropy should be higher

Step 2: Randomly Distribute Points

Intuitve Idea

1. In the previous step, we calculated the probabilistic distance between all points → We will store this distance in a matrix **P**
2. Now we want to create a 2 dimensional visualisation for us to interpret the data clusters

Process:

1. Raw features X_1, X_2, X_3 are not directly used in this step
2. Create a random (x,y) coordinate *[2-dimensional visualisation]* for each data entry and calculate the probabilistic distance between any 2 points
3. Instead of using a Gaussian Distribution, this time we will use a t-distribution
4. **Note!** There is no hyperparameter for this step and we are using joint distributions instead of conditional probabilities
5. We store the probabilistic distances in this low-dimension space in another matrix **Q**

t-distribution with 1 degree of freedom →
Equivalent to the standard Cauchy distribution

$$p_{0,j} = \frac{1}{\pi\left(1 + x^2\right)}$$

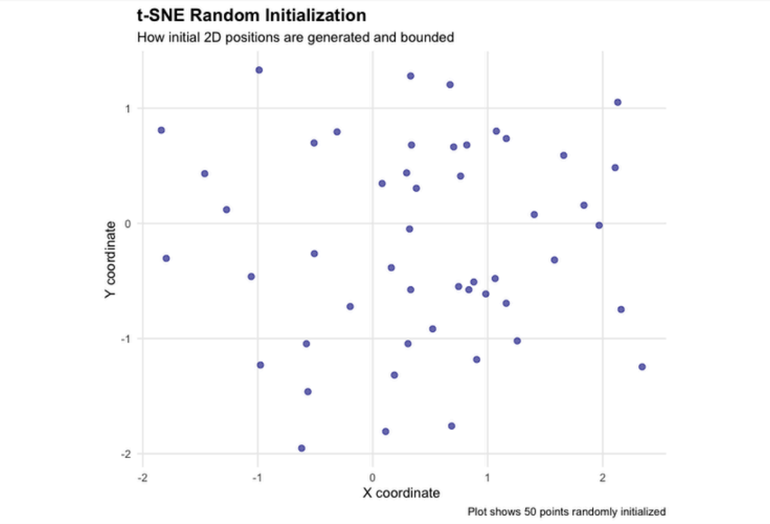
A joint probability is used instead because we want to calculate

- On a global scale
- Symmetric distance between i → j and j → i

$$Q_{i,j} = \frac{\left(\pi\left(1 + \|y_i - y_j\|^2\right)\right)^{-1}}{\sum_{k \neq l}\left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

Data	X1	X2	X3
1	5.0	7.24	8.2
2	7.3	5.62	0.4
3	2.4	4.99	6.3

Data	X-coord	Y-coord	Similarity
1	-1.796	-0.3036	0
2	1.162	0.737	0.0923
3	2.131	-1.0522	0.4905



Step 3: Preserving Local Relationship

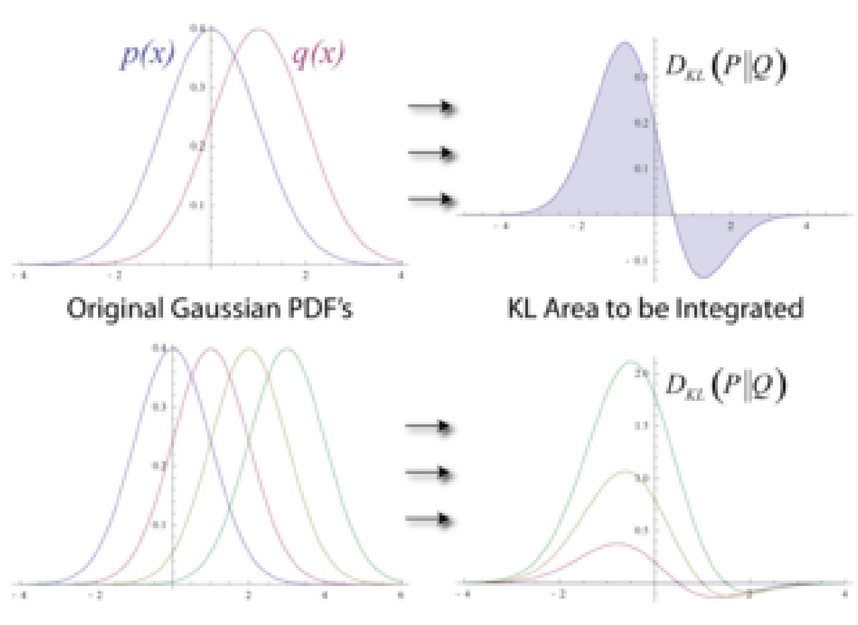
Aim: We now have 2 similarity matrix (high dimension [P] and low dimension [Q]), we want to make the low dimension matrix match the high dimension matrix by moving points around the 2D space.

Process:

$$D_{KL} = \sum_{i=1}^{n} \sum_{j \neq i} P(i,j) log\left(\frac{P(i,j)}{Q(i,j)}\right)$$

$$\frac{\delta D_{KL}(P||Q)}{\delta y_i} = 2\sum_j (p_{i,j} - q_{i,j} + p_{j,i} - q_{j,i})(y_i - y_j)$$

1. Kullback-Leiber (KL) Divergence ← Loss function
    - We want to measure how different Q is from P
    - Formally: measures the difference between two probability distributions
2. Gradient Descent
    - For each point in the 2-Dimension Space, which direction and how far should it be moved
    - Essentially the same formula as linear regression but with a different derivative
3. Update Positions
    - For each data point, gradient descent is applied to calculate which direction and how far the data point should be moved in the 2D space
    - Repeats until gradient of cost function stops decreasing
    - Take this value to be its minimum point



t-SNE

Code Implemetation

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df) # standardise the dataset
df_scaled = pd.DataFrame(df_scaled, columns=df.columns, index=df.index)
df_scaled.head()
```

Standardising is still important in t-SNE as we are using Euclidean distance in the algorithm.
If not standardised, features with larger scales will dominate the distance calculation.

```
from sklearn.manifold import TSNE #tSNE library in python

tsne = TSNE(n_components=2, random_state=2102, verbose=1, perplexity = 40)
# the default perplexity for tSNE is 40. This means 40 effective neighbours

df_tsne = tsne.fit_transform(X_small) # this function does the tSNE decomposition for you
```

This function does all the computation for you!
Hyperparameters you can control:

- Perplexity
- Learning rate (for Gradient Descent)

Limitations

Time Complexity

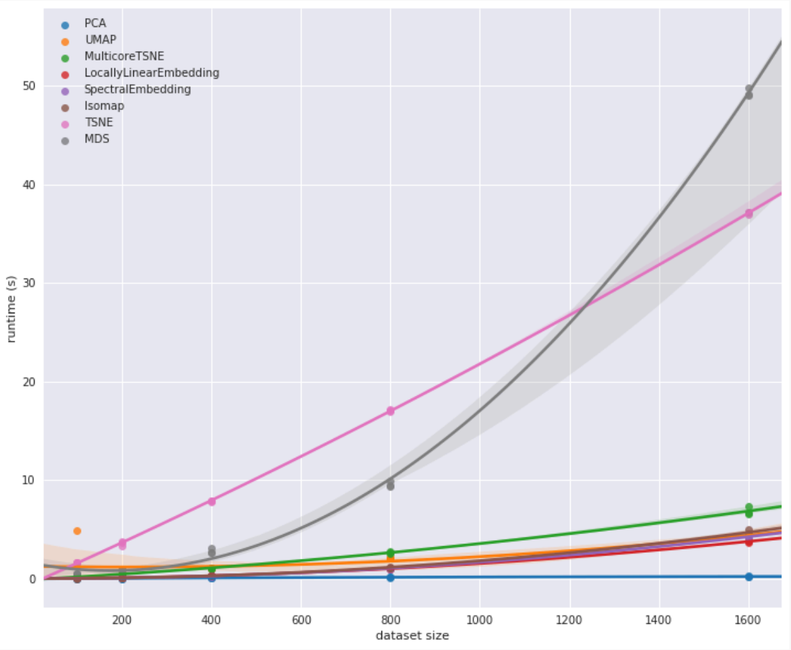
- The algorithm runs in $O(n^2)$
- PCA runs in O(n)
- It is often costly in terms of time to run t-SNE on large datasets.
- In Big O notation, n refers to the number of data points you have.
- This means that the time taken for the algorithm to run is proportional to the square of the number of data points you have in the dataset.

Sensitivity of Hyperparameters

- Often, changing the perplexity by a bit changes the visualisation generated by t-SNE completely.
- Trial and error needed to find the optimal hyperparameters. This can be time consuming especially because the algorithm itself is slow.

Lack of Interpretability

- t-SNE only helps in visualising the structure of the data in high dimension space
- You can combine it with other unsupervised techniques like K-Means and DBSCAN to identify groups in your data
- But it does not help to extract features that can be used in predictive tasks



UMAP

Uniform Manifold Approximation and Projection

Aim: To project data from a high dimension space into a lower dimension space while preserving similarity relationships.

Uniform	UMAP assumes that the data is uniformly distributed on a Riemannian manifold. This means that the data points lie on a smooth, curved surface that can be locally approximated by Euclidean space.
Manifold	A curved surface or shape in high-dimensional space where the data lives
Approximation	Computer approximations to efficiently estimate the manifold structure
Projection	Map the high-dimensional manifold structure down into 2D or 3D while preserving relationships

This model has a strong foundation in topology and category theory

- Riemannian manifold (topology)
- Fuzzy set (category theory)

We will not be going in depth into the theory in this workshop

Key Idea: Data points that are close together in a high dimension space should still be close together in low dimension

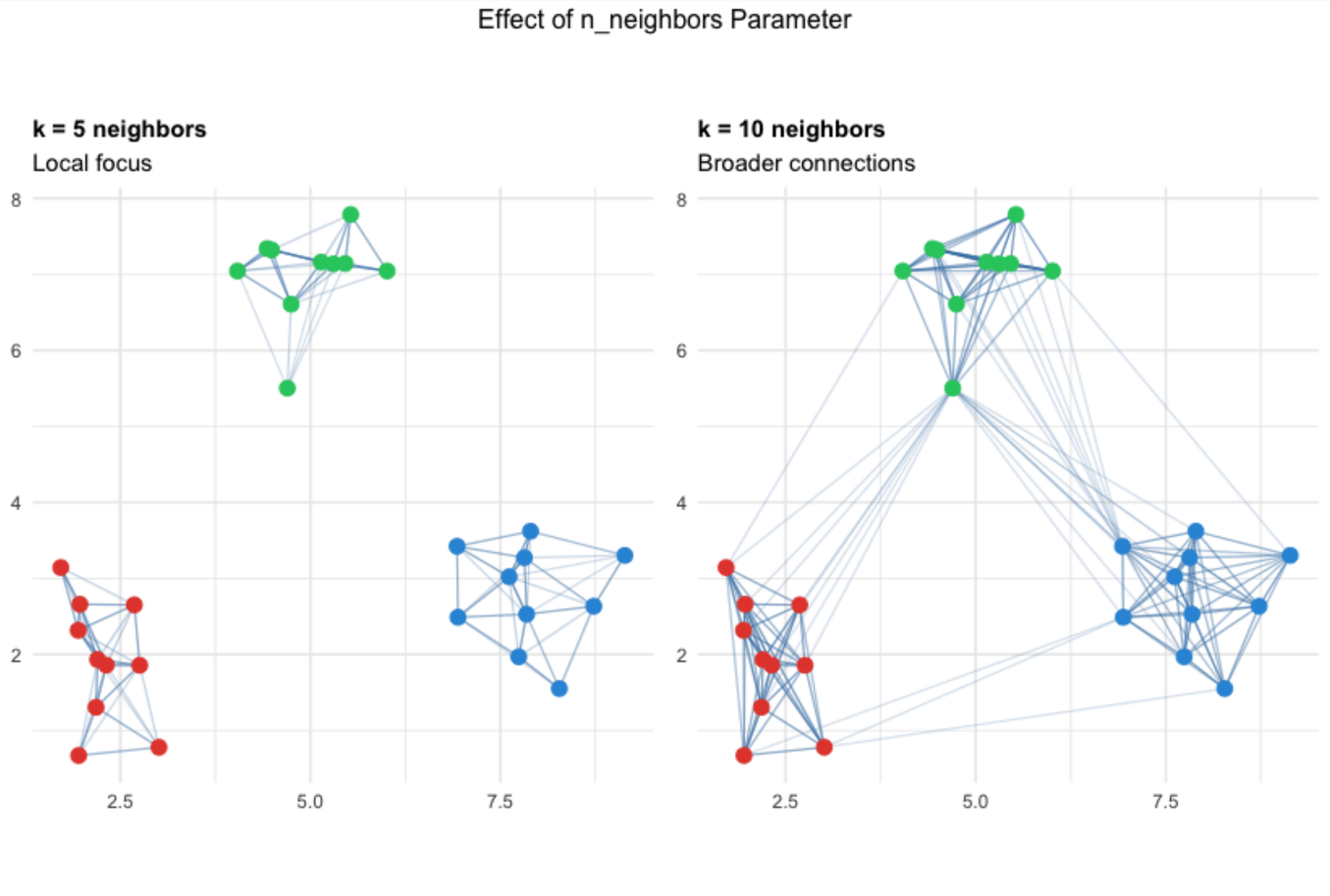
Step 1: Building a Graph in High Dimension

Process

1. Find k-Nearest Neighbours for each point
2. Create edges between neighbours
3. Closer neighbours have higher weighted edges

Hyperparameters

- n_neighbours



Step 1: Building a Graph in High Dimension

The Math Behind

Calculating Edge Weights

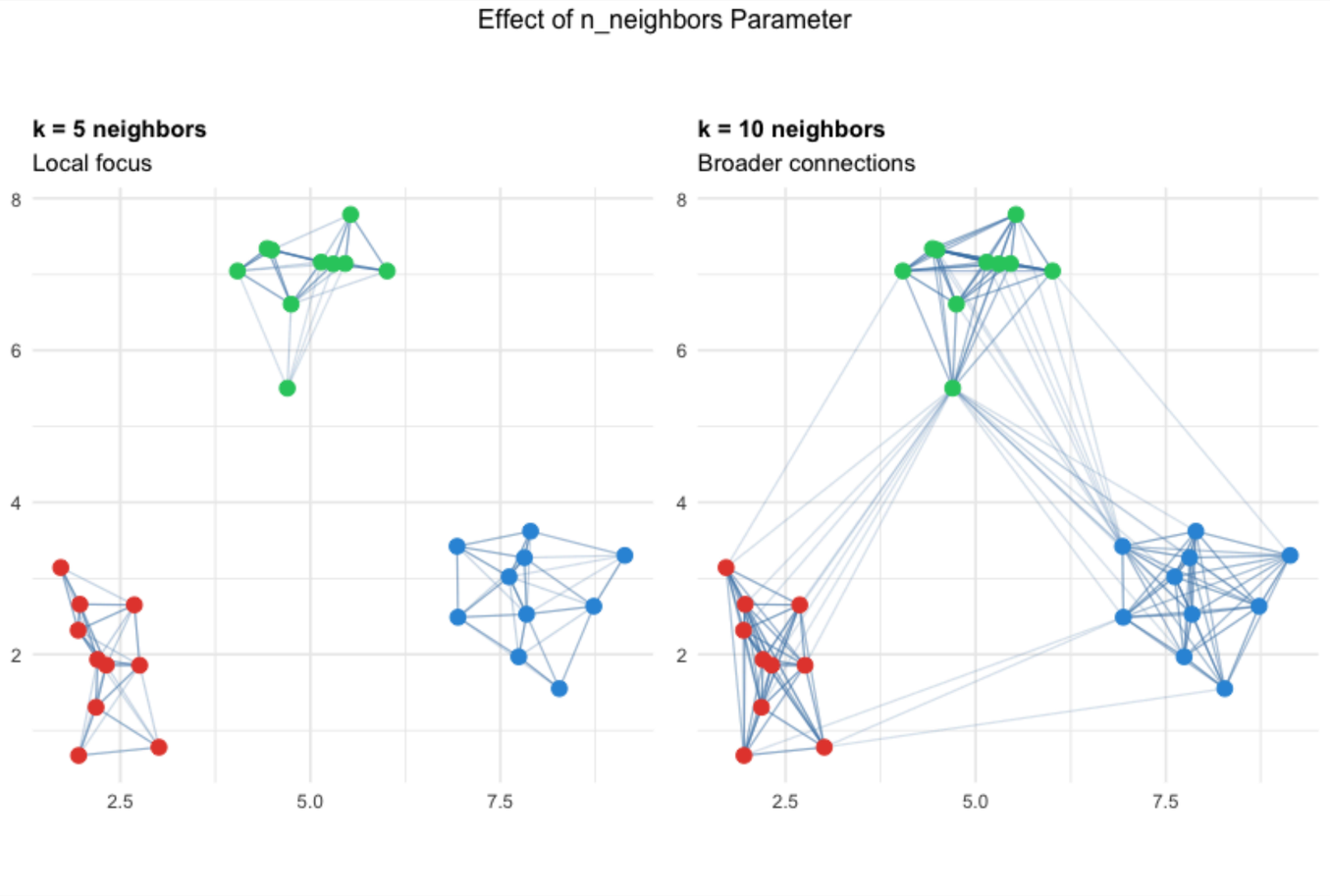
$$\mu_{i,j} = exp\left(-\frac{max\left(0, d\left(x_i, x_j\right) - \rho_i\right)}{\sigma_i}\right)$$

$d(x_i, x_j)$ ← Euclidean distance between two points

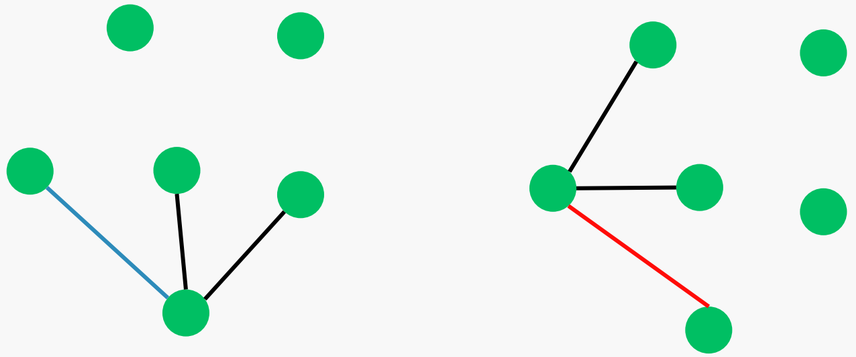
ρ_i ← Forces the nearest neighbour to get a value of 1

σ_i ← Scaling factor to make distances comparable across dense and sparse regions

This formula comes from fuzzy set theory
Don't worry about what it means, just know that it takes a value from 0 to 1 and the larger the number the stronger the edge between two neighbours



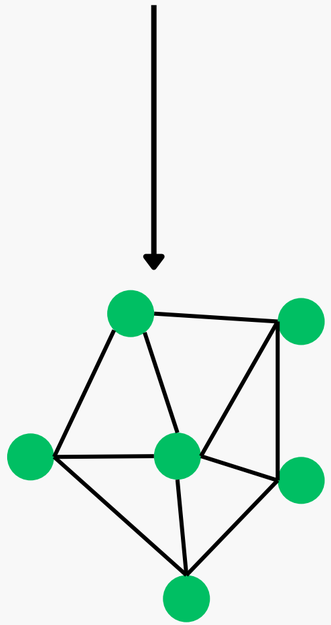
Step 2: Combining Different Graphs



Graph 1 Graph 2

From Previous Step

- We have calculated the weighted edge for each data point to its closest neighbour, forming a graph
- But how do we combine these individual graphs together to form a manifold?



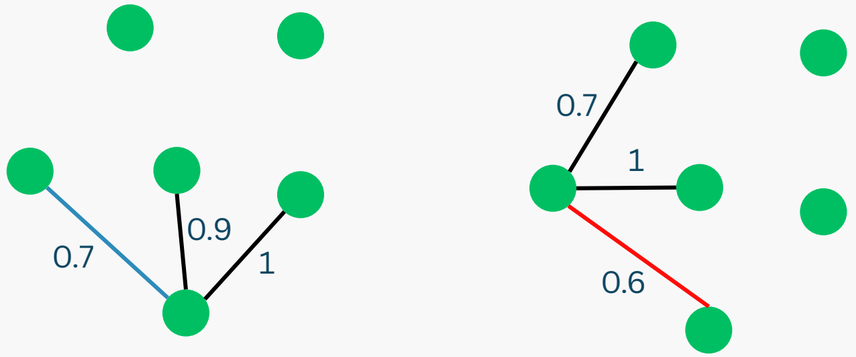
Manifold

Step 2: Combining Different Graphs

Process

1. Symmetrise the edges so that there is only 1 edge between any two points
2. Apply the formula on below:

$$\mu_{i,j}^{sym} = \mu_{i,j} + \mu_{ji} - (\mu_{ij} \times \mu_{ji})$$



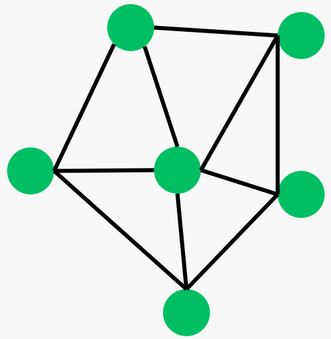
Symmetrise the blue and red edge

$\mu_{i,j} = 0.7 + 0.6 - (0.7 \times 0.6) = 0.88$

Repeat this step for every pair of points

Results

- Now we have a complete graph of the high dimension manifold
- We store the weighted edges between two points in a matrix **P**



Step 3: Creating a Graph in Low Dimension

Process

1. Each data point gets a random (x, y) coordinate
2. Repeat step 1 and step 2 to get a 2 dimension graph
3. Store the edge weights between points in a matrix **Q**

Note: The original features from your data set is not used in this stage

Goal (next step)

- Make this 2D graph similar to the high dimensional graph

Data	X1	**X2**	**X3**
1	5.0	7.24	8.2
2	7.3	5.62	0.4
3	2.4	4.99	6.3

Data	**X-coord**	**Y-coord**	**Edge Weight**
1	-1.796	-0.3036	0
2	1.162	0.737	0.0923
3	2.131	-1.0522	0.4905

Step 4: Preserving Relationship Between Points

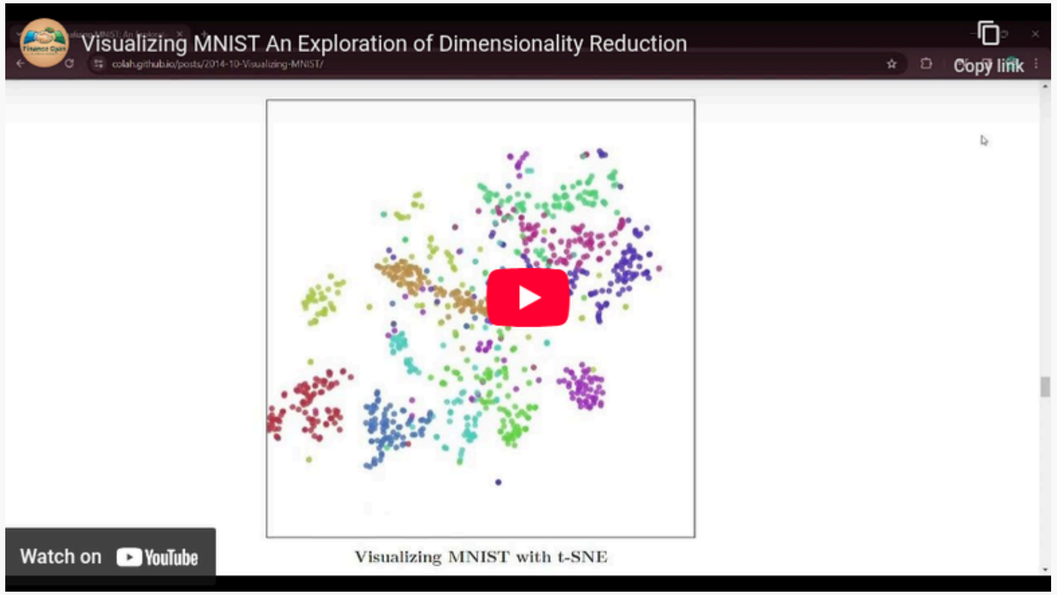
Where we are at now:

Matrix P: Which points are close together in the high dimension

Matrix Q: Which points are close together in the low dimension

What we aim to do:

- Make Matrix Q match Matrix P
- Between any 2 points, their edge weights in low dimension should be equivalent to their edge weights in high dimension
- The structure that you see in 2D is similar to the high dimension structure



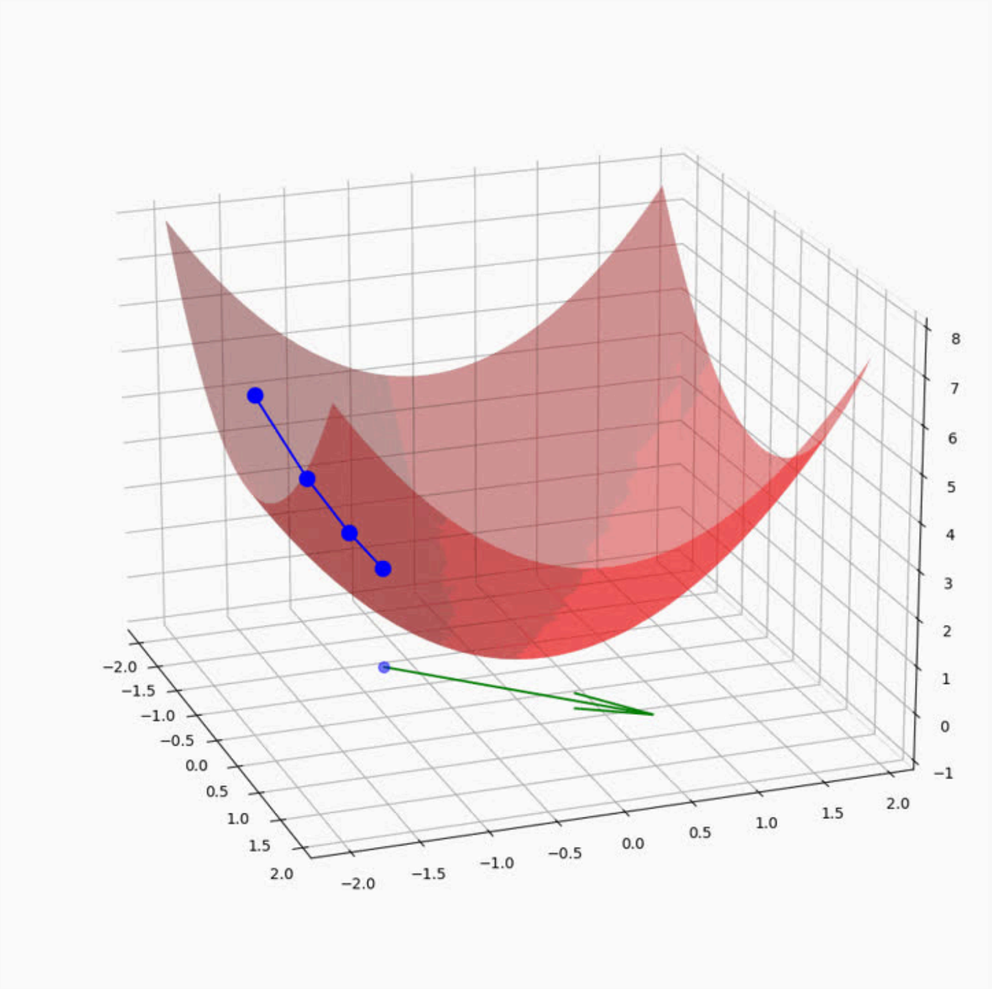
Note: While the video says t-SNE, the underlying process of minimising a loss function using gradient descent and it effect is the same

Visualising MNIST An Exploration of Dimensionality Reduction
https://www.youtube.com/watch?v=SKb6rcmJfpM
Channel: Finance Gyan in Stock Market

Step 4: Preserving Relationship Between Points

How we do it:

- We use Cross-Entropy as a loss function to measure the difference between each entry in P and Q
- Minimise the loss function using Gradient Descent
- Iterate until the loss function stops decreasing



UMAP

Code Implemetation

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df) # standardise the dataset
df_scaled = pd.DataFrame(df_scaled, columns=df.columns, index=df.index)
df_scaled.head()
```

Standardising is still recommended in UMAP as we are using Euclidean distance in the algorithm. If not standardised, features with larger scales will dominate the distance calculation.

```
from umap import UMAP # python library for UMAP

umap = UMAP(n_neighbors= 10, n_components= 2, random_state= 2102)
df_umap = umap.fit_transform(X_small)
```

This function does all the computation for you!

Hyperparameters you can control:

- n_neighbours

Challenge Time!

Challenge Instructions

Aim: Predict Weekly Sales Figures!

- Whoever creates predictions with the lowest RMSE will win a prize!
- Focus on Supervised Learning
  - Try out Feature Engineering
  - Change Model Hyperparameters
  - Try out different Models! Find out which is best
- Unsupervised Learning has been done for you… but read through it! It might present some clues for what Supervised Learning Models to focus on.

Time Limit: 20 minutes

Materials are also on

Github

https://github.com/NUS-SDS-Workshops/AY-25-26-Public

Our next workshop:

Week 11: Deep Learning!

Beginner friendly! If you did not request to receive email and want to for this workshop, tele pm @kaiironglee

Feedback Form

Fill in so we can imporve future workshops for you!

Thank you