302 Machine Learning

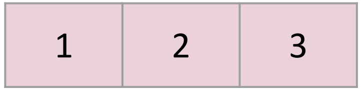
2025 Semester 1

WEEK 2

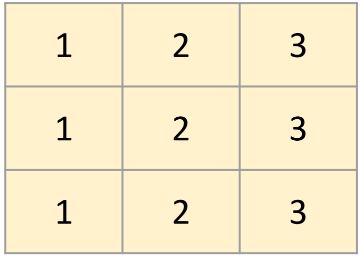
Python NumPy

Numpy (Numerical Python)

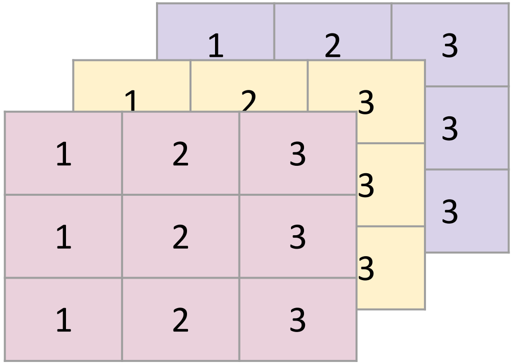
1D, 2D, and 3D NumPy Array Shape



array = (3,)



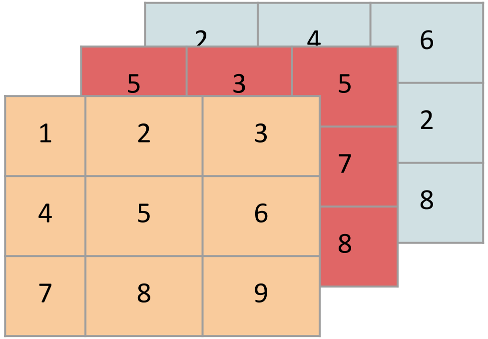
array = (3, 3)



array = (3, 3, 3)

```
import numpy as np
a = np.array([10, 20, 30, 40, 50])

print(a.dtype)          # int64
print(a.shape)          # (5,)
print(a.ndim)           # 1
print(type(a))          # <class 'numpy.ndarray'>
```

```
import numpy as np
b = np.array([6, 7.5, 8, 0, 1, 2])

print(b.dtype)          # float64
print(b.shape)          # (6,)
print(b.ndim)           # 1
print(type(b))          # <class 'numpy.ndarray'>
```



```
import numpy as np
c = np.array([
        [1, 2, 3], [4, 5, 6], [7, 8, 9],
        [5, 3, 5], [7, 1, 7], [8, 2, 8],
        [2, 4, 6], [8, 1, 2], [4, 6, 8]])
```

0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.

```
import numpy as np
e = np.zeros((3, 6))
```

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

```
import numpy as np
i = np.arange(20).reshape(4,5)
```

1.	1.	0.
0.	1.	0.
0.	0.	1.

```
import numpy as np
f = np.eye(3)
```

1.	1.	1.
1.	1.	1.

```
import numpy as np
g = np.ones((2, 3))
```

0	4	4
2	6	9
1	4	5

```
import numpy as np
h = np.random.randint(0, 10, (3,3))
```

0	2	4	6	8	10

```
import numpy as np
d = np.arange(0, 12, 2)
```

Element-Wise Arithmetic Operations

```
i = np.arange(6).reshape(2,3)
print(i * i)
```

0	1	2
3	4	5

X

0	1	2
3	4	5

=

0	1	4
9	16	25

Indexing and Slicing

```
c = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

1	2	3
4	5	6
7	8	9

print(c[0:2, :])

1	2	3
4	5	6

print(c[2, 1:])

8	9

print(c[:2, 1,2])

2	3
5	6

WEEK 3

Python Pandas

Python Panda Data Structures

Series
1D Labeled homogeneous array with immutable size

Name
Luffy
Nami
Usopp
Chopper

\+

Age
17
18
18
15

\+

Gender
M
F
M
M

\+

Rating
5
4
3
4

Data Frames
2D Labeled heterogeneous array with mutable size

Name	Age	Gender	Rating
Luffy	17	M	5
Nami	18	F	4
Usopp	18	M	3
Chopper	15	M	4

import pandas as pd

pandas.DataFrame(data, index, columns, dtype, copy)

data = {'Age': [17, 18, 18, 15], 'Gender': [M, F, M, M], 'Rating': [5, 4, 3, 4]}

df = pd.DataFrame(data, index=['Luffy', 'Nami', 'Usopp', 'Chopper'], index = ['Name', 'Age','Gender', 'Rating'])

Name	Age	Gender	Rating
Luffy	17	M	5
Nami	18	F	4
Usopp	18	M	3
Chopper	15	M	4

Using .loc[] and .iloc[] to Fetch / Slicing

.loc[] - Locates by name (Label-based indexing)
.iloc[] - Locates by numerical index

```
df.iloc(['Nami', 'Gender'])        # F
df.iloc(['Usopp'])                 # 'Usopp', 18, 'M', 3
df.iloc([:, 'Gender'])             # 'M', 'F', 'M', 'M'
df.iloc(['Chopper', 'Rating'])     # 4

df.loc([0, 0])                     # 'Luffy'
df.loc([0])                        # 'Luffy', 17, 'M', 5
df.loc([:, 1])                     # 17, 18, 18, 15
df.loc([0:1])                      # ['Luffy', 17, 'M', 5], ['Nami', 18, 'F', 4]
```

Reading Data from CSVs

```
df = pd.read_csv('dataset.csv')
df = pd.read_csv('dataset.csv', index_col='title')
```

Handling Duplicates

```
temp_df = onePiece_df.append(onePiece_df)
temp_df  = temp_df .drop_duplicates(inplace=True)
```

Isolate Single Rows

```
temp_df[temp_df.index == 1]
temp_df[temp_df['Gender'] == 'M']
```

Data Cleaning - Handling Missing Values

1. Removing rows with MV's
2. Replace MV's with a mean, median, mode

isnull() - Check for MV's
dropna() - Drop MV's
fillna(), replace() - Fill MV's

```
df.isnull().any()      # Returns a boolean per column for MV's
df.isnull.sum()        # Returns a sum of MV's
```

```
df.fillna(df['column_name'].mean(), inplace=True)
df['column_name'] = df['column_name'].replace('old_value', 'new_value')
```

Name	Age	Gender	Rating
Luffy	17	M	5
Nami	18	F	4
Usopp	18	M	NaN
Chopper	15	M	4
Sanji	NaN	M	4
Zoro	16	M	3
Robin	19	NaN	5

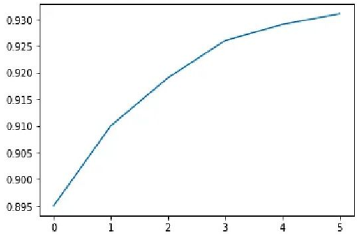
WEEK 4

Python - Data Visualisation

Data Visualisations

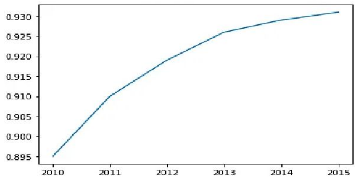
Line Graph (No X-Axis Information)

```
apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
plt.plot(apples)
```



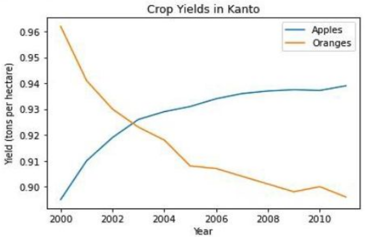
Line Graph (With X-Axis Information)

```
years = [2010, 2011, 2012, 2013, 2014, 2015]
plt.plot(years, apples)
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
```



Multiple Line Graph (With X-Axis Information)

```
oranges = [0.962, 0.941, 0.930, 0.923, 0.918, 0.908]
plt.plot(years, apples)
plt.plot(years, oranges)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title('Crop Yields in Kanto')
plt.legend(['Apples', 'Oranges'])
```



Imports

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Specify Graph Size

```
plt.figure(figsize=(12, 6))
```

Specify Markers

```
plt.plot(years, apples, marker='o')
plt.plot(years, oranges, marker='x')
```

Specify Grid

```
sns.set_style("whitegrid")          # Applies to entire file
sns.set_style("darkgrid")
```

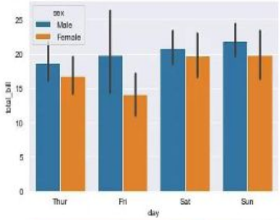
Load Seaborn Default Datasets

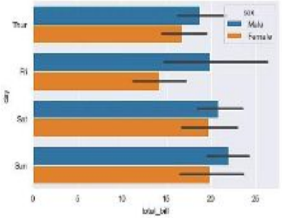
```
tips_df = sns.load_dataset("tips")
```

Different Types of Charts

Bar Graphs

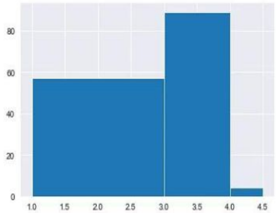
```
years = [2010, 2011, 2012, 2013, 2014, 2015]
apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
oranges = [0.962, 0.941, 0.930, 0.923, 0.918, 0.908]

# Seaborn uses barplot() while matplotlib uses bar()
plt.bar(years, oranges, bottom=apples)                                  # Stacked Bar Graph
sns.barplot(x='day', y='total_bill', data=tips_df)                      # Seaborn requires parameter specification
sns.barplot(x='day', y='total_bill', data=tips_df, palette='viridis')   # Customise color
sns.barplot(x='day', y='total_bill', data=tips_df, hue=sex)             # Color by a group

sns.barplot(x='total_bill', y='day', hue=sex)                           # Switching x and y creates a Horizontal Bar Plot
```





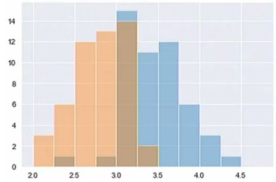
Histograms

```
flowers_df = sns.load_dataset('iris')
plt.title("Distribution of Sepal Width")
plt.hist(flowers_df.sepal_width, bins=np.arange(2, 5, 0.25))    # Specify bins using arange(start, stop, step)
plt.hist(flowers_df.sepal_width, bins=[1, 3, 4, 4.5])           # Specify bins of unequal size

flowers_df.species.unique()                                     # Find unique items in a specific column
```



Multiple Histograms

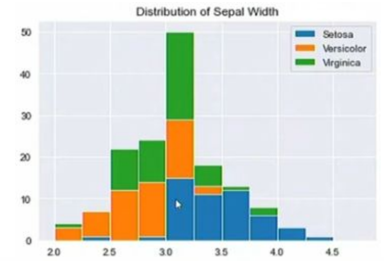
```
setosa_df = flowers_df[flowers_df.species == 'setosa']
versicolor_df = flowers_df[flowers_df.species == 'versicolor']
virginica_df = flowers_df[flowers_df.species == 'virginica']

# Adjust alpha to customise transparency
plt.hist(setosa_df.sepal_width, alpha = 0.4, bins = np.arange(2, 5, 0.25))
plt.hist(versicolor_df .sepal_width, alpha = 0.4, bins = np.arange(2, 5, 0.25))
```



Different Types of Charts (Cont.)

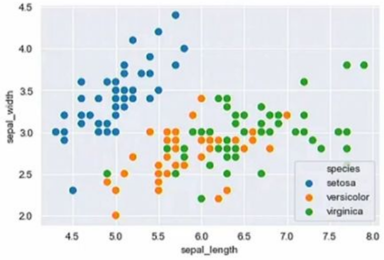
Stacked Histograms

```
plt.hist([setosa_df.sepal_width, versicolor_df .sepal_width, virginica_df .sepal_width],
            bins = np.arange(2, 5, 0.25),
            stacks = True)
plt.legend(['Setosa', 'Versicolor', 'Virginica'])
```



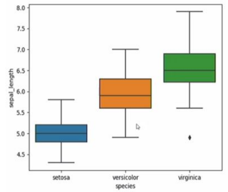
Scatter Plots [Plotting more than two variables]

```
# s adjusts the dot size
sns.scatterplot(x=flowers_df.sepal_length, y = flowers_df.sepal_width, hue = flowers_df.species, s = 70)
```



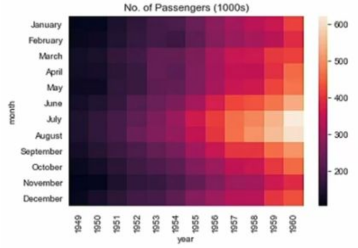
Box Plots

```
sns.boxplot(x = df['Species'], y = df['Sepal_length'], palette = "Blues")
plt.show()
```



Heatmaps

```
flights_df = sns.load_dataset('flights').pivot('month', 'year', 'passengers')
plt.title('No. of Passengers (1000s)')

# fmt = 'd' specifies = integers
# annot = True specifies numbers show
# cmap = 'Blues' specifies color
sns.heatmap(flights_df, fmt = 'd', annot = True, cmap = 'Blues')
```



WEEK 5

Python: EDA & Data Preprocessing

Key steps for Exploratory Data Analysis (EDA)

1. Understand the Problem and the Data

2. Importing Libraries

3. Loading / Reading Dataset

4. Data Inspection

5. Data Cleaning

    5.1. Checking for Missing Values

    5.2. Checking for Duplicates

6. Analysing the Data

    6.1. Univariate Analysis

    6.2. Bivariate Analysis

    6.3. Multivariate Analysis

1. **Understand the Problem and the Data**
    - What is the business goal or research question?
    - What are the variables in the data and what do they represent?
    - What types of data (numerical, categorical, text, etc
    - Are there any known data quality issues or limitations
    - Are there any domain-specific concerns or restrictions?

2. **Importing Libraries**
    - Pandas
    - NumPy
    - Matplotlib.pyplot
    - Seaborn

3. **Loading / Reading Dataset**
    - pd.read_csv()

4. **Data Inspection**
    - Get an overview of the data using df.head(), tail(), info()
    - Check data types with df.dtypes()

5. **Data Cleaning**

    5.1
    - Identify and handling missing values using df.isnull().sum()

    5.2
    - Find and address duplicates with df.duplicated().sum()

6. Analysing the Data

6.1
- Analyse single variables at a time
- Use descriptive statistics with df.describe() for numerical data
- Create histograms, box-plots, and density plots for visualise distributions

6.2
- Explore relationships between two variables
- Create scatter plots, pair plots to identify trends and potential correlations

6.3
- Interactions between three or more variables in a dataset are simultaneously analysed and interpreted in multivariate analysis
- Use various plots like heatmaps, line charts to explore

EDA → Data Pre-processing → Feature Engineering

Exploratory Data Analysis (EDA): Insurance.csv

1. **Understand the Problem and the Data**

The dataset Insurance.csv is used to predict customer chargers for an insurance company based on given variables so the company can decide how much they charge people correctly.

- Age
- Sex
- BMI
- Smoker
- Children (Number of children covered by health insurance / Number of dependents
- Region (The beneficiary's residential area in the US, northeast, southeast, southwest and northwest)
- Charges (Individual medical costs billed by health insurance)

2. **Importing Libraries**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sns.set_style('whitegrid')
import warnings
warnings.filterwarnings('ignore')

from google.colab import drive
drive.mount('/content/drive')
```

3. **Loading / Reading Dataset**

```
df = pd.read_csv('/content/drive/MyDrive/insurance.csv')
```

Exploratory Data Analysis (EDA): Insurance.csv (cont.)

4. Data Inspection

```
df.head(10)
df.tail(10)
df.sample(5)                   # Randomly select 5 rows
df.info()
df.dtypes()
df.describe()                  # Summary statistics for numerical data only
df.describe(include = 'o')     # 'o' = object data type; Descriptive statistics for categorical variables such as count, unique, top, frequency
list(df.sex.unique())
```

5. Data Cleaning

```
df.isnull().sum()
duplicate_rows = df[df.duplicated()]
df.drop_duplicates(keep='first', inplace = True)
```

6. Analysing the Data

6.1 Univariate Analysis

Distplot: Charges

```
plt.figure(figsize=(10,6))
sns.distplot(df.charges, color='b')
plt.title('Charges Distribution', size = 18)
plt.xlabel('Charges', size = 14)
plt.xlabel('Density', size = 14)
plt.show()
```

Distplot: Age

```
plt.figure(figsize=(10,6))
sns.distplot(df.age)
plt.title('Age Distribution', size = 18)
plt.xlabel('Age', size = 14)
plt.xlabel('Count', size = 14)
plt.show()
```

Hist: BMI

```
plt.figure(figsize=(10,6))
plt.hist(df.bmi, colo = 'g')
plt.title('BMI Distribution', size = 18)
plt.show()
```

Boxplot: Charges

```
plt.figure(figsize=(10,6))
sns.boxplot(df.charges)
plt.title('Distribution Charges', size = 18)
plt.show()
```

Exploratory Data Analysis (EDA): Insurance.csv (cont.)

6. Analysing the Data

6.1 Univariate Analysis

Countplot: Gender

```
plt.figure(figsize=(10,6))
sns.countplot(x = 'sex', data = df)
plt.title('Total num of M and F', size = 18)
plt.xlabel('Sex', size = 14)
plt.show()
```

Countplot: Smoker

```
plt.figure(figsize=(10,6))
sns.countplot(x = 'smoker', data = df)
plt.title('Smoker Distribution', size = 18)
plt.xlabel('Smoker', size = 14)
plt.xlabel('Count', size = 14)
plt.show()
```

Countplot: Region

```
plt.figure(figsize=(10,6))
sns.countplot(x = 'region', data = df, palette = 'Blues')
plt.title('Region Distribution', size = 18)
plt.xlabel('Region', size = 14)
plt.xlabel('Count', size = 14)
plt.show()
```

6.2 Bivariate Analysis

Scatter Plot: Age vs Gender

```
plt.figure(figsize=(10,6))
sns.countplot(x = 'age', y = 'Charges', data = df)
plt.title('Age vs Charges', size = 18)
plt.xlabel('Age', size = 14)
plt.xlabel('Charges', size = 14)
plt.show()
```

Pairs Plot: All Numeric Variables

```
sns.pairplot(df, diag_kind = 'kde')
plt.show()
```

6.3 Multivariate Analysis

Heatmap: All numeric variables

```
plt.figure(figsize=(10,6))
numeric_df = df.select_dtypes(include=np.number)
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

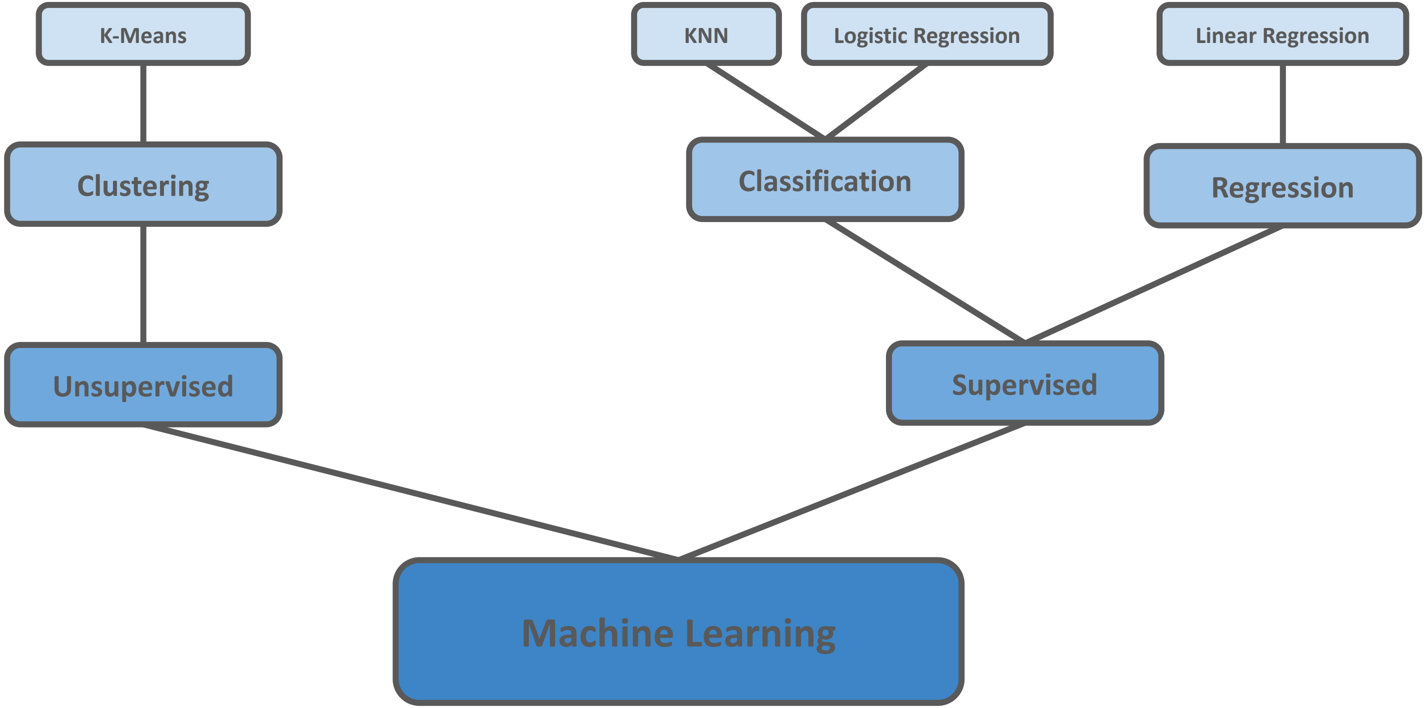
WEEK 6

Introduction to Machine Learning and Regression

Machine Learning Tasks

- Self-driving cars and robots use object detection and performance feedback to improve their interaction with the world → **learning to interact in an environment**
- Retail companies and streaming services use data to recommend new content we might like based upon our demographic and historical preferences → **classifying data into groups and categories**
- Banks look for trends in transaction data to detect outliers that may be fraudulent → **finding trends in data**
- Email inboxes use text to classify whether an email is spam or not, and adjust their classification rules based upon how we flag emails → **making decisions and predictions**
- Travel apps use live and historic data to predict traffic, travel times, and journey routes → **making decisions and predictions**

Types of Machine Learning Algorithms



Supervised

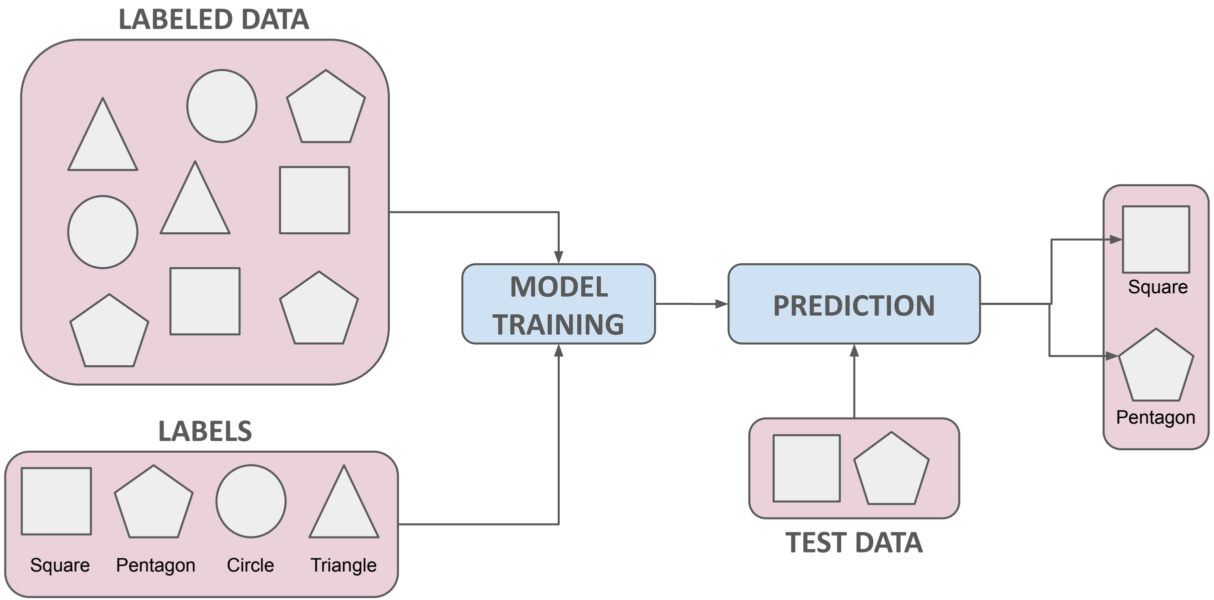
Needs external supervision to learn

Models are trained using the labeled dataset

Takes a known set of input data and known responses to the data and trains a model to generate predictions for the response to new data

Once training is done, the model is tested by providing a sample test data to check whether it predicts the correct output

Use supervised learning if you have known data for the output you are trying to predict



Regression [Linear Regression]
If theres a relationship between input and output variables. Used to make predictions of continuous variables, like weather forecasting or market trends

Classification [KNN, Logistic Regression]
When the output variable is binary and is a categorical variable, which means there are two classes such as Yes-No, Male-Female, True-False, 0-1, etc

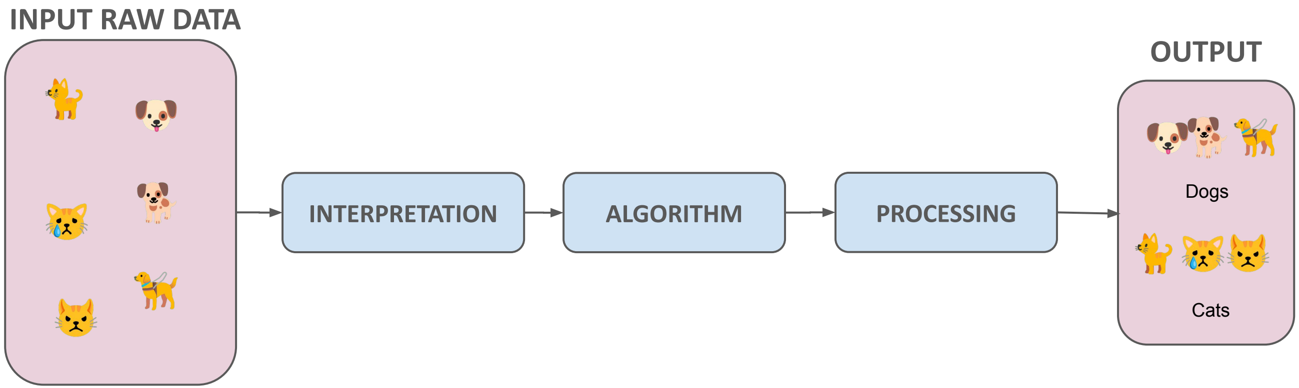
Unsupervised

The machine does not need any external supervision to learn from the data

Can be trained using the unlabelled dataset that is not classified, nor categorised

It is used to draw inferences from data sets consisting of input data without labeled responses

No predefined output and tries to find useful insights from a large amount of data



Clustering [K-Means]

Applications include gene sequence analysis, market research and anomaly detection, for example, customer segmentation

Machine Learning Workflow [STEPS]

1. Extract features
2. Split dataset
3. Train dataset
4. Train model
5. Evaluate (Using test dataset)

Choosing the right Algorithm

There is no best method, it is partly trial and error, and partly depends on the size and type of data, the insights you want, and how insights will be used

Supervised	- You need to make a prediction such as the future value of a continuous variable (such as temperature, stock) or a classification (such as identifying car makers from webcam video footage)
Unsupervised	- You need to explore your data and want to train a model to find a good internal representation (such as splitting data up into clusters)

Regression

Multiple Linear Regression: One dependent variable and multiple independent variables
Simple Linear Regression: One independent and one dependent variable

- Predicting what the price of a product will be in the future, whether prices go up or down
- Estimating the number of houses a builder will sell in the coming months and at what price
- Predicting the number of runs a baseball player will score in upcoming games based on previous performance
- Understanding how temperature affects ice cream sales
- Predicting the price of a house given house features
- Predicting the impact of college scores on University admission

EXAMPLE

Create a regression model for a dataset that will predict exam scores from hours spent revising using score.csv

```
import pandas as pd
study_scores = pd.read_csv('score.csv')
sns.relplot(x='Scores', y = 'Hours', data=study_scores)         # After seeing that a linear regression is appropriate (straight-line), fit the model

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

X = study_scores .iloc[:,:-1].values
y = study_scores .iloc[:,1].values

from sklearn.model_selection import train_test_split          # Split dataset in ration of 70:30, where 70% is the training set and 30% is the testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state = 1)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)                             # Mean Squared Error (MSE)
np.sqrt(mean_squared_error(y_test, y_pred))                    # Root Mean Squared Error (RMSE)
```

MSE & RMSE

MSE:

- Measures average squared error per prediction.
- Higher MSE means worse performance (larger errors).
- Because it squares the errors, larger errors are penaliSed more than smaller ones.

RMSE:

- RMSE is in the same units as the target variable, which makes it more interpretable than MSE.
- Gives a sense of the typical size of the prediction error.
- Like MSE, larger errors have a higher impact due to squaring.

For a RMSE of 7.5: On average, if your score is between 0-100, the model predicts values below or above 7.5 marks, which is pretty bad

WEEK 7

Regression, and Model Performance

Logistic Regression

Logistic Regression

- Determining whether an employee would get a promotion or not based on their performance
- Banks predicting whether a customer would default on loans or not
- Predicting weather conditions of a certain place (sunny, windy, rainy)
- Identifying buyers if they are likely to purchase a certain product
- Predicting whether they will gain or lose money in the next quarter, year or month based on their current performance
- To classify objects based on their features and attributes

EXAMPLE

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=1, max_iter=1000)          # max_iter is optional

X = df['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp', 'pedigree']
y = df.label

from sklearn.model_selection import train_test_split                # Split dataset in ration of 75:25, where 75% is the training set and 25% is the testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = 16)
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
```

Logistic Regression Performance Metrics [Confusion Matrix, Accuracy, Precision, Recall, F1-Score]

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
```

		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Logistic Regression (Cont.)

Logistic Regression Performance Metrics [Confusion Matrix, Accuracy, Precision, Recall, F1-Score]

```
from sklearn.metrics import accuracy_score        # TP + TN / (TP + TN + FP + FN)
accuracy = accuracy_score(y_test, y_pred)         # An accuracy score of 0.73, means 27% of the time, the model is NOT correctly predicting values

from sklearn.metrics import precision_score       # TP / (TP + FP)
precision = precision_score(y_test, y_pred)

from sklearn.metrics import recall_score          # TP / (TP + FN)
recall = recall_score(y_test, y_pred)

from sklearn.metrics import f1_score              # Inverse of the mean; mean = (x1 + x2 + x3 + … + xn) / n
f1score= f1_score(y_test, y_pred)                 # 2 * Precision * Recall / (Precision + Recall)
```

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

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

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

Use BOTH precision and recall: When there is an imbalance in the observations between the two classes

- EG: There are more of one class (1) and only a few of the other class (0) in the dataset

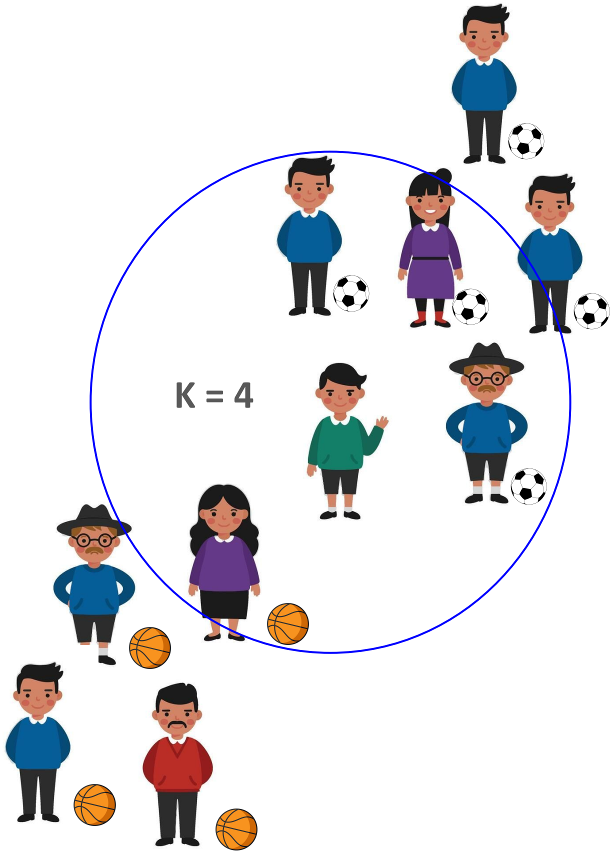
USE PRECISION: False positives are costly (spam detection, fraud alerts) OR if you have a balanced dataset
USE RECALL: False negatives are costly (disease diagnosis, safety alerts)
USE F1 SCORE: Both false positives and false negatives should be small

WEEK 8

Classification

KNN (K-Nearest Neighbour)

Predicting the green guys favourite sport based on his nearest neighbours



Euclidean Distance = sqrt(x1 - y1)^2 + (x2 - y2)^2

Predicting Luffy's default status (Y or N)

For K = 5

Name	Age	Loan	Default	Euclidean Distance	Minimum ED	
Usopp	**15**	**5000**	**N**	**40002**	**3**	= sqrt(17 - 15)^2 + (5200 - 5000)^2
Sanji	**16**	**2000**	**N**	**10240001**	**5**	= sqrt(17 - 16)^2 + (5200 - 2000)^2
Robin	18	1500	N	13689999		= sqrt(17 - 18)^2 + (5200 - 1500)^2
Gaban	**15**	**6400**	**N**	**1440002**	**4**	= sqrt(17 - 15)^2 + (5200 - 6400)^2
Vivi	18	9300	N	16809999		= sqrt(17 - 18)^2 + (5200 - 9300)^2
Zoro	**16**	**5200**	**Y**	**1**	**1**	= sqrt(17 - 16)^2 + (5200 - 5200)^2
Nami	**17**	**5400**	**Y**	**40000**	**2**	= sqrt(17 - 17)^2 + (5200 - 5400)^2
Buggy	15	1500	Y	13690002		= sqrt(17 - 15)^2 + (5200 - 1500)^2
Lucci	17	8200	Y	11560000		= sqrt(17 - 17)^2 + (5200 - 1800)^2
Luffy	17	5200	?			

The default status for Luffy is N because for K=5, there are 3 points = N and only 2 points = Y.

In Python:

1. The k-nearest neighbour import
2. Create feature and target variables
3. Split data
4. Generate KNN model using neighbour value
5. Train or fit data into the model
6. Predict the future

KNN is a lazy learner, it parses through ALL data points for each classification, and therefore only works on smaller datasets.

KNN also fails when variables have different scales. Feature scaling (standardisation and normalisation) is required before applying KNN.

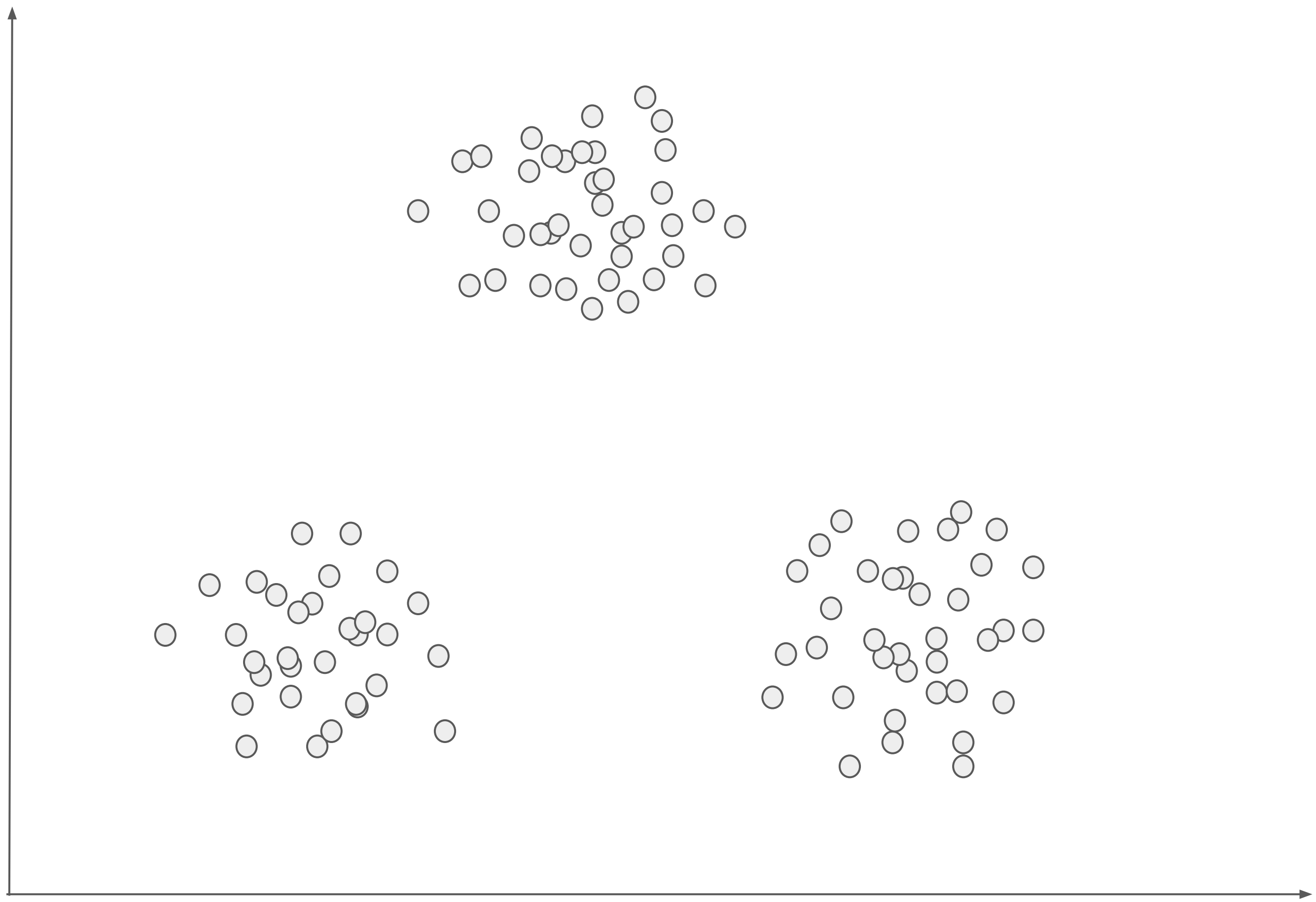
KNN - Classification: Predicting a category or class

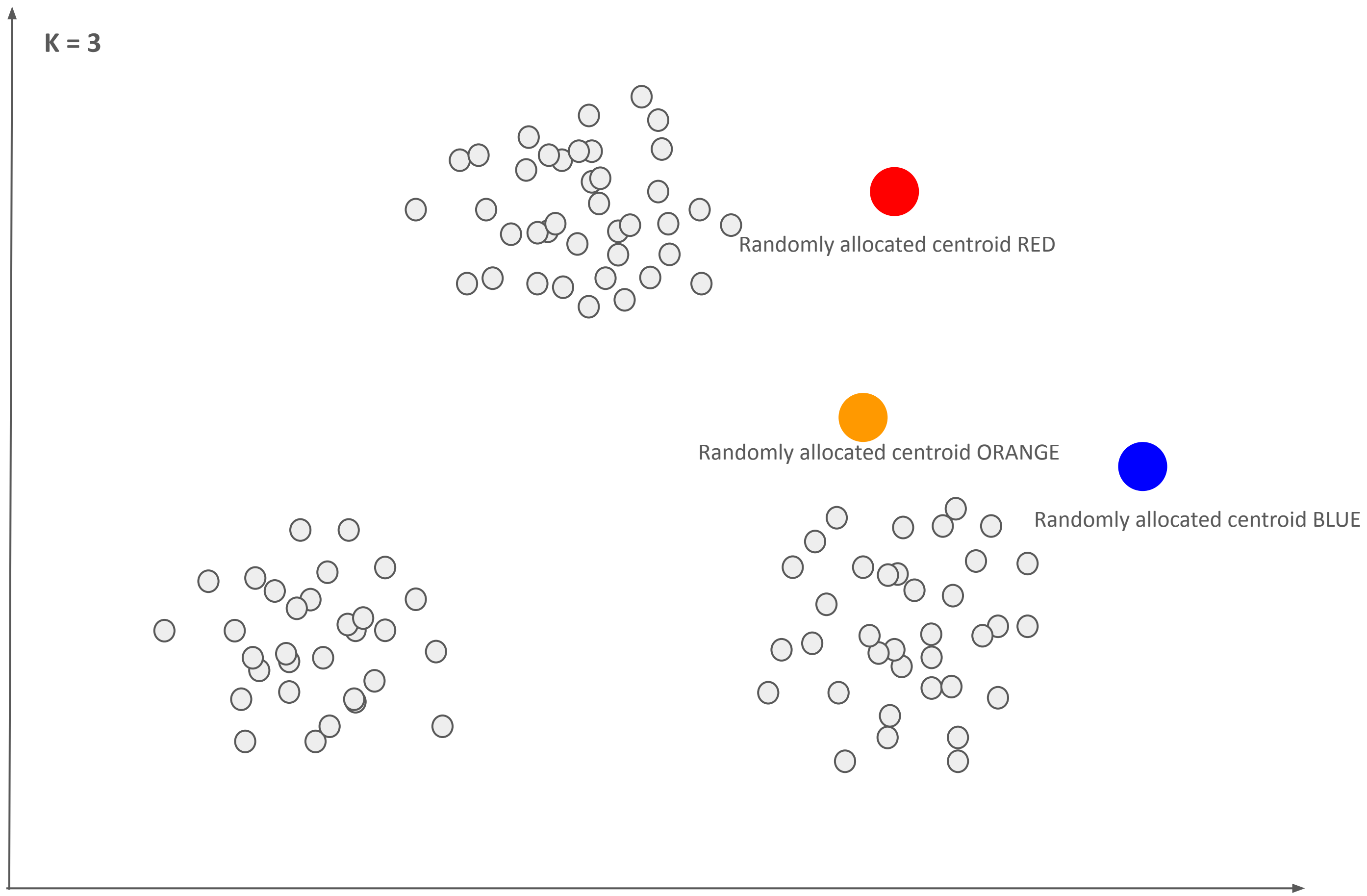
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
df = pd.read_csv('iris.csv')
x = df.iloc[:,:4]
y = df.iloc[:, 4].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 42)
knn = KNeighborsClassifier(n_neighbirs = 7)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)

from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
```

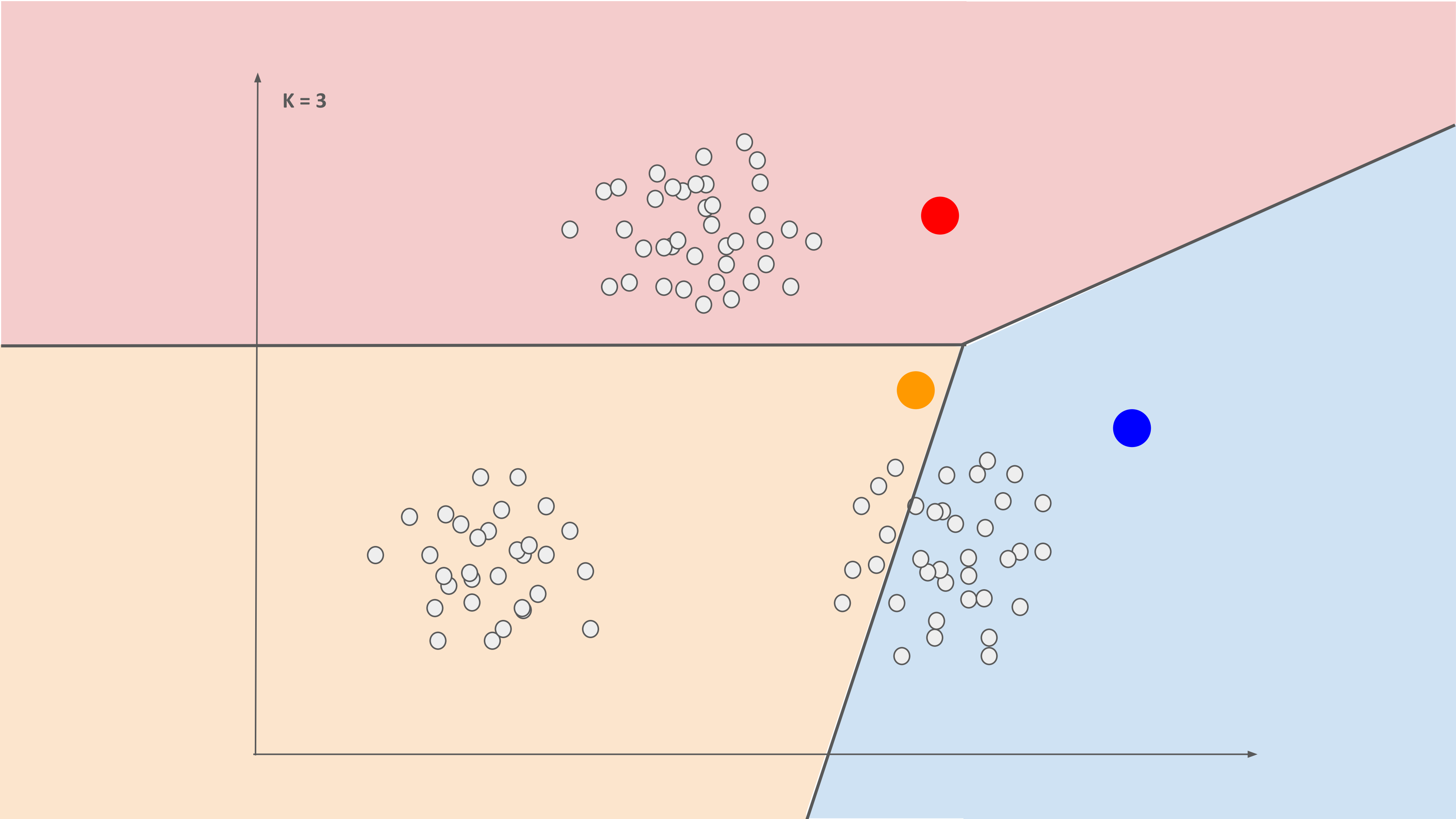
KNN - Regression: Predicting a continuous number

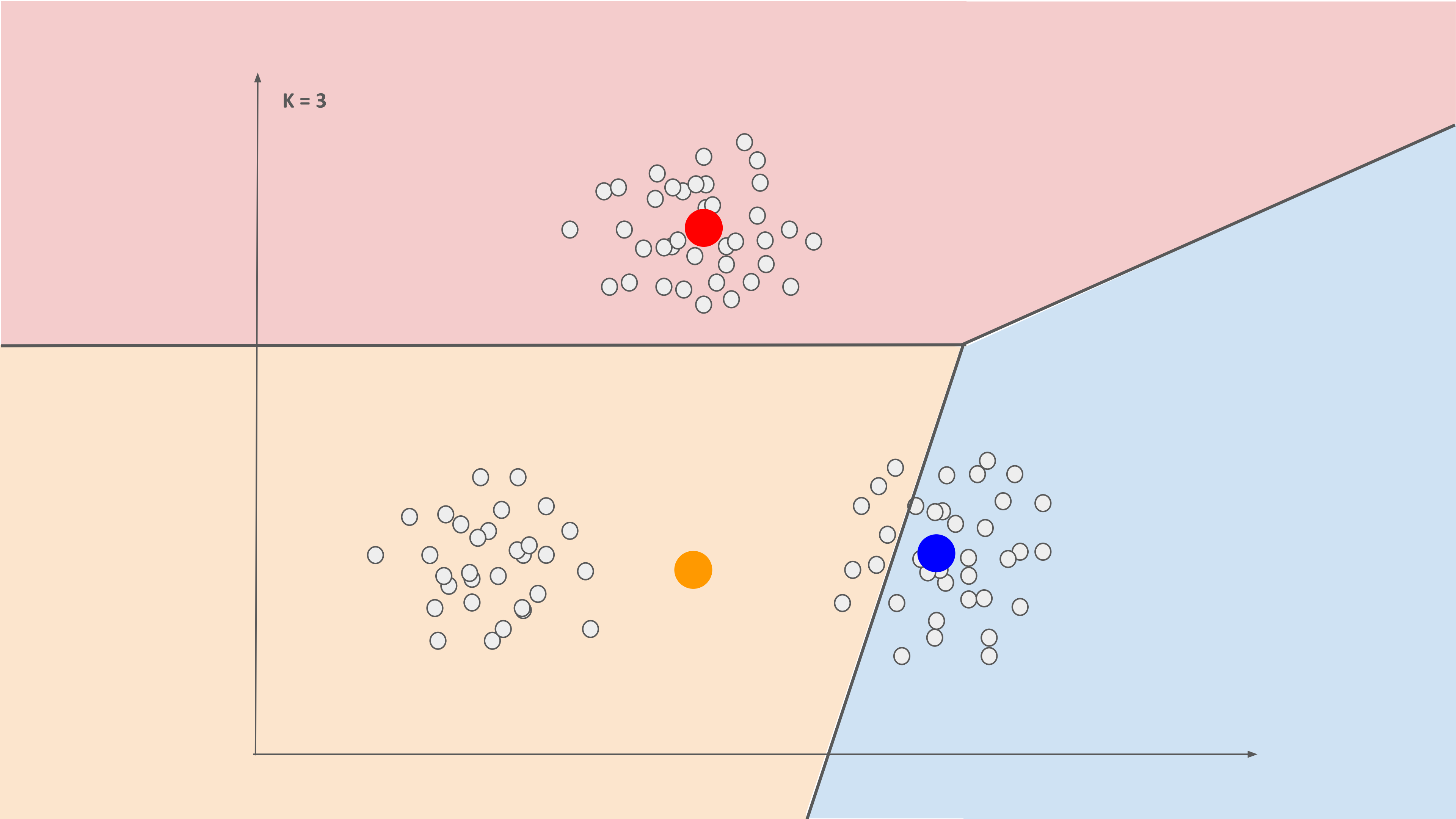
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 42)
model = KNeighborsRegressor(n_neighbirs = 9)
model .fit(X_train, y_train)
y_pred = model.predict(X_test)

import math
import sklearn.metrics as skl_metrics                # Outcome is a continuous, so NO confusion matrix
e1 = skl_metrics.mean_squared_error(y_test, y_pred)  # Mean Square Error
math.sqrt(e1)                                        # Root Mean Square Error
```

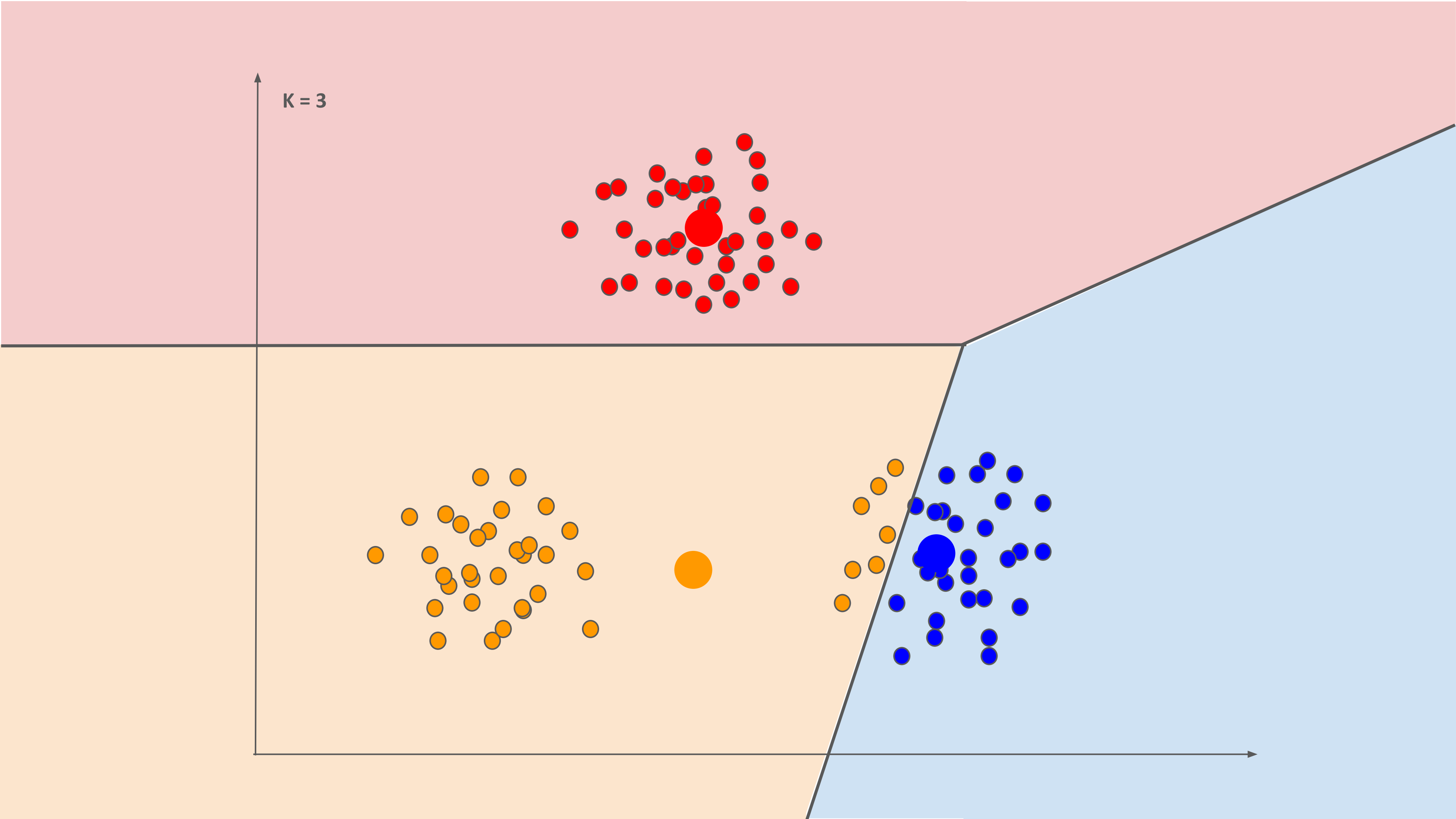
K-Means Simulation

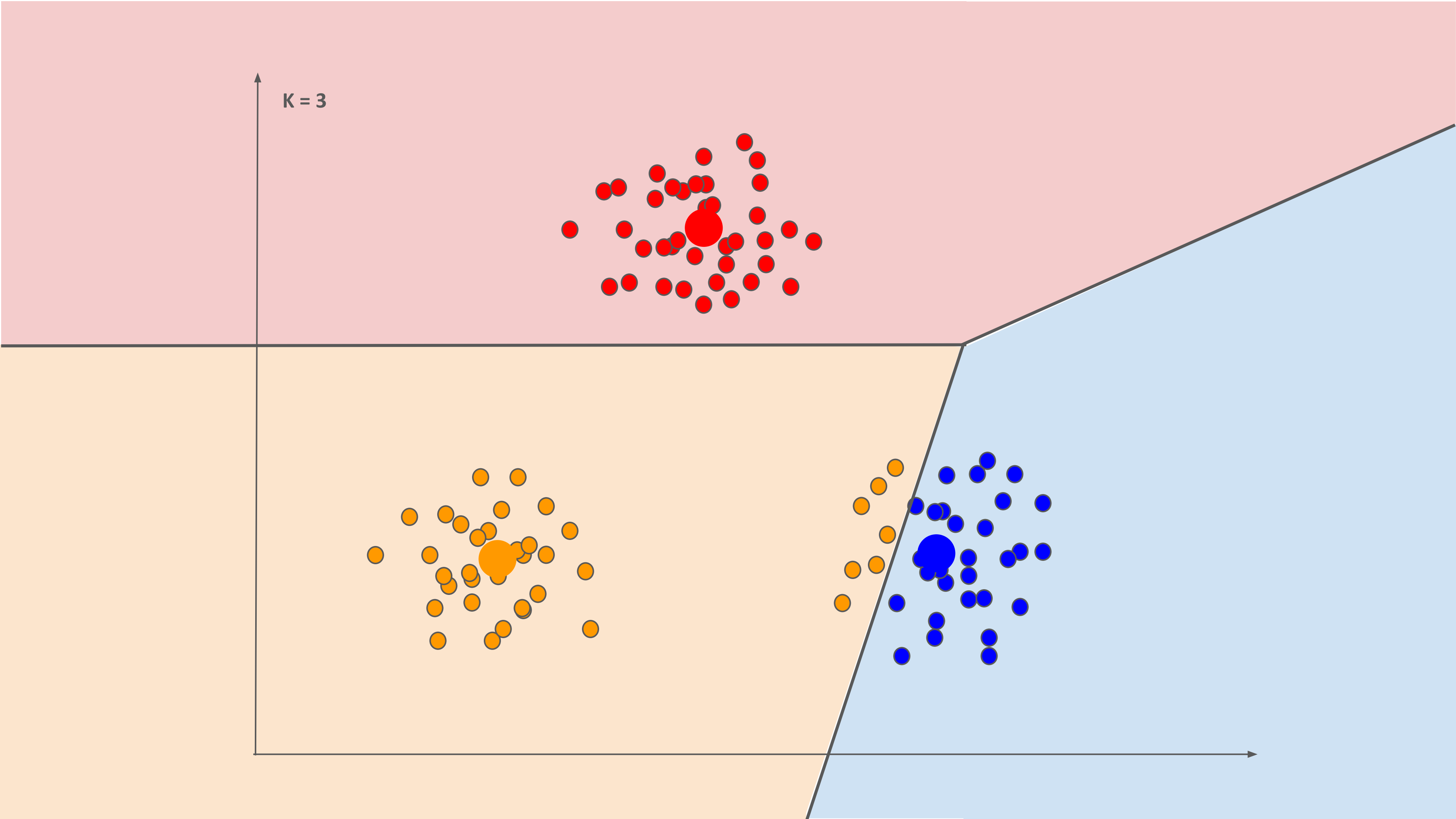


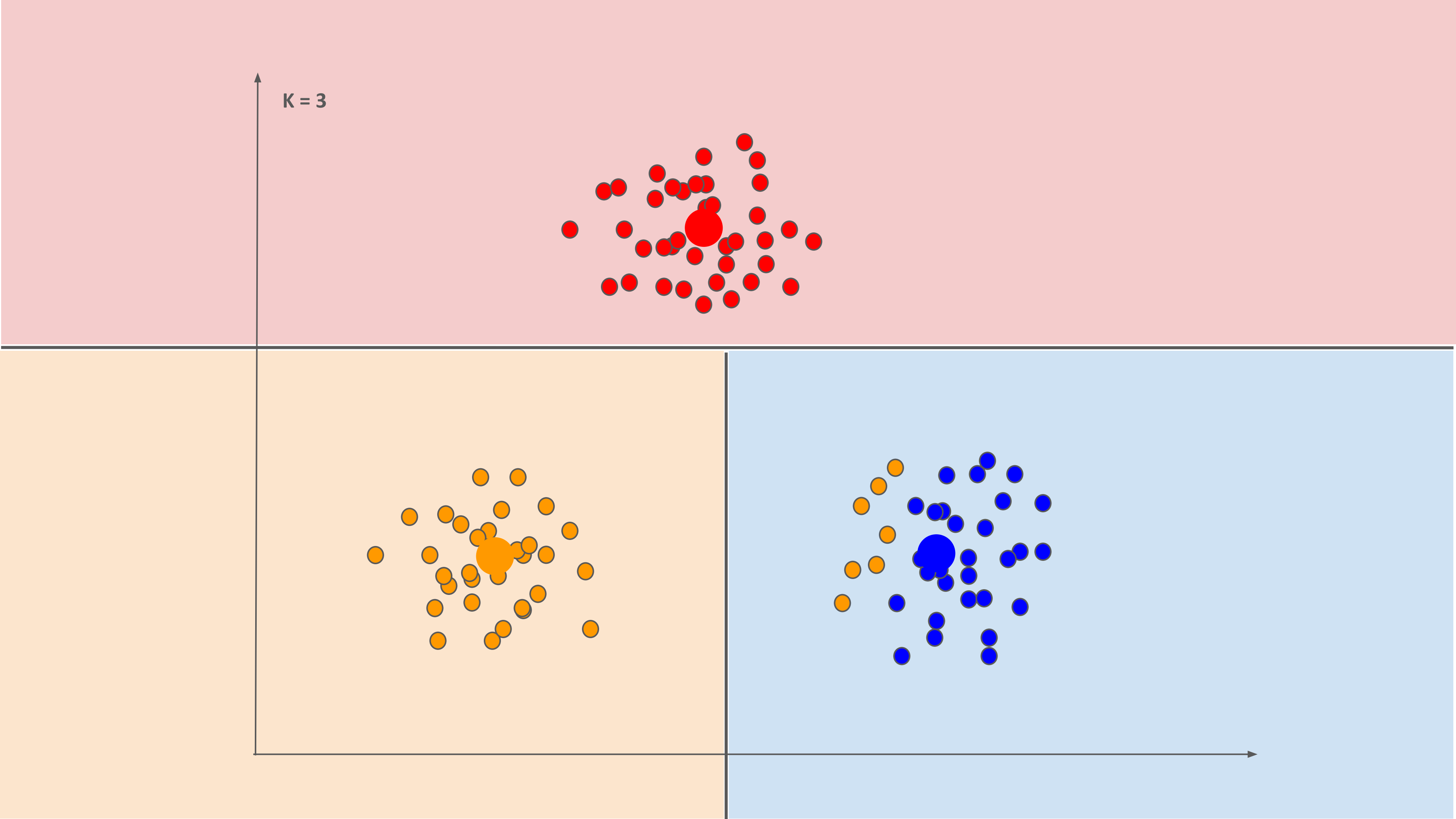

K = 3
Randomly allocated centroid RED
Randomly allocated centroid ORANGE
Randomly allocated centroid BLUE

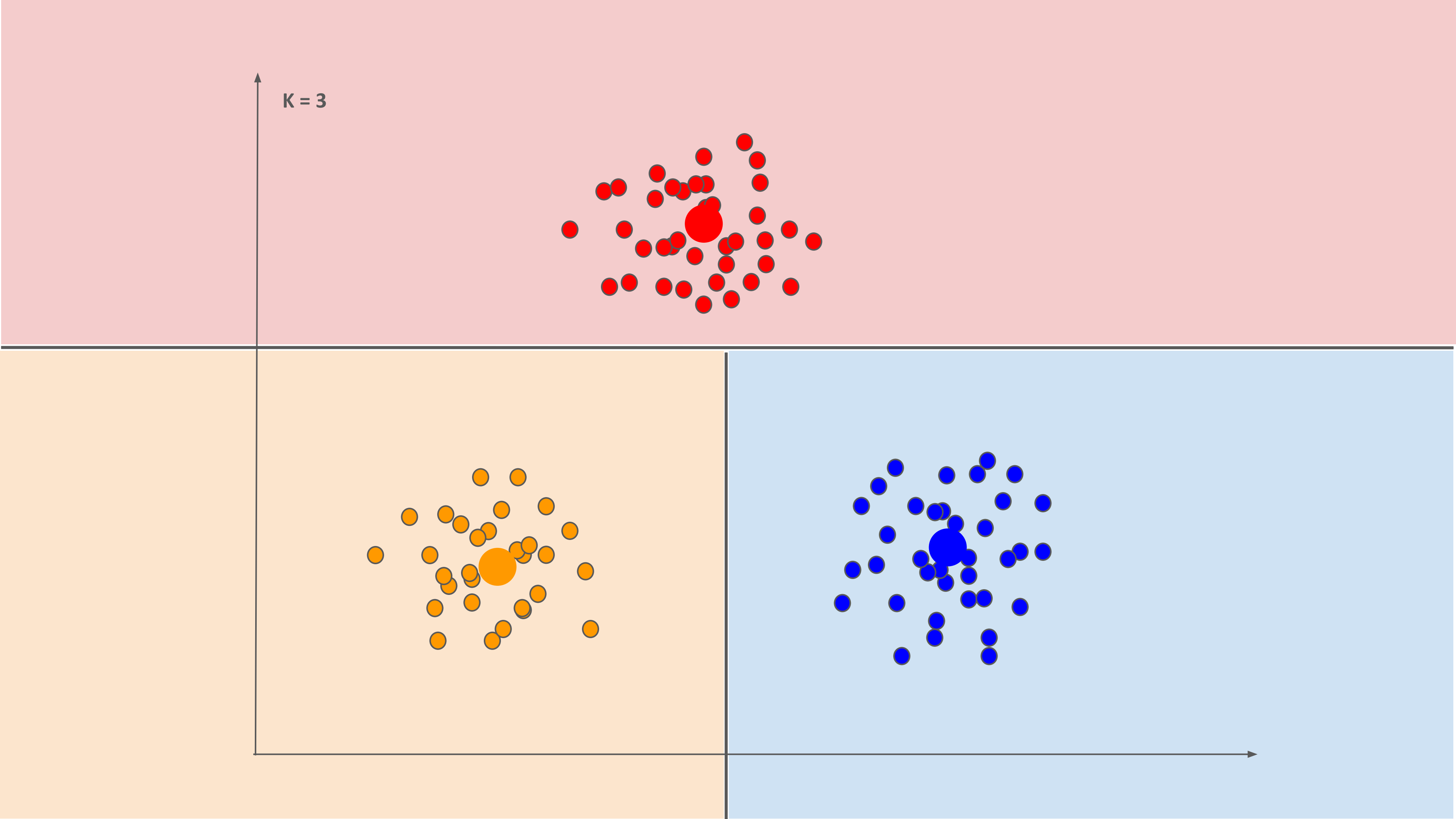

K = 3


K = 3

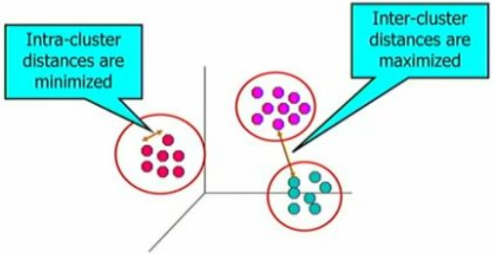

K = 3


K = 3


K = 3


K = 3

***UNSUPERVISED K-Means Clustering



K-Means is an UNSUPERVISED clustering algorithm designed to partition unlabeled data into a certain number (K) of distinct groups.

Each cluster is represented by its center (centroid) which corresponds to the arithmetic mean of data points assigned to the cluster. 'K' represents the number of clusters we want to classify our data points into. A centroid is a datapoint that represents the mean and might not necessarily be a member of the dataset. The algorithm works iteratively until each datapoint is closer to its own clusters' centroid than to other clusters' centroids, minimising intra-cluster distance at each step.

```
from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters = 3)
kmeans_predict = kmeans_model.fit_predict(x)

iris['Cluster'] = kmeans_predict           # Merge the result of the clusters with our original dataset
centroids = km.cluster_centers_

x1 = x[x['Cluster']==0]
x2 = x[x['Cluster']==1]
x3 = x[x['Cluster']==2]
plt.scatter(x1['Age'],x1['Income($)'],color="blue",s=100)
plt.scatter(x2['Age'],x2['Income($)'],color="red",s=100)
plt.scatter(x3['Age'],x3['Income($)'],color="purple",s=100)
plt.scatter(centroids[:,0],centroids[:,1],color="orange",marker="*",s=150);

predicted_cluster=km.predict([[23,50000]])
```

WEEK 9

Clustering

K-Means Performance Metrics

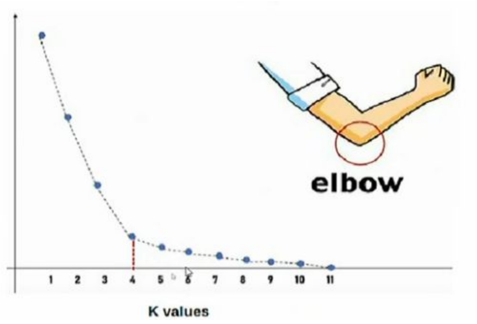
Elbow Method

To find the best value of K

WCSS: The total within-cluster sum of squares measures the compactness of the clustering, and we want it to be as small as possible. WCSS is the average distances to the centroid across all data points.

The elbow method runs k-means clustering on the dataset for a range of values k.

ELI5: Imagine you have a big box of legos, they are different sizes and different colors. You want to sort them, so you start with two boxes, this is better, but then you use three boxes, even better, you then feel overly optimistic and try 10 boxes, but this is just too much work. K-Means Elbow method finds a threshold value for K which provides the perfect WCSS for a particular K-Value



Silhouette Score / Analysis

A metric to evaluate the quality of clustering performed by K-Means. Measuring how well data points are grouped within their assigned clusters compared to data points in other clusters.

You use K-Means ML algorithm, and now you want to measure how good the clusters are.

a = How close it is to points in its own cluster (you want this to be small)
b = How close it is to points in the next nearest cluster (you want this to be large)

Silhouette Score = (b - a) / max(a, b)
The silhouette score ranges from -1 to 1

1: Ideally close data points within a cluster and far from other clusters (GOOD)
0: Data points on the border between clusters indicating some overlap (AVERAGE)
-1: Data points might be assigned to the wrong cluster (BAD)

Feature Engineering

Imputation | Outlier Handling | One-hot encoding | Log transformation | **Scaling**

The most common techniques of feature scaling are **Normalisation** and **Standardisation**

Normalisation: Values are bound between 0 and 1 or -1 and 1
Standardisation: Transforms values to have zero mean and a variance of 1

Absolute Maximum Scaling (Very sensitive to outliers)

1. Select the maximum absolute value out of all entries of a column
2. Divide each entry by this maximum value
3. Observe that each entry lies in the range of -1 to 1

```
max_vals = np.max(np.abs(df))
print((df - max_vals) / max_vals)
```

Min-Max Scaling

1. Find the minimum and the maximum value of the column
2. Subtract the minimum value from the entry and divide the result by the difference between the maximum and the minimum value

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_data, columns = df.columns)
```

Mean Normalisation

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
scaled_data = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_data, columns = df.columns)
```

Feature Engineering (Cont.)

Imputation | Outlier Handling | One-hot encoding | Log transformation | **Scaling**

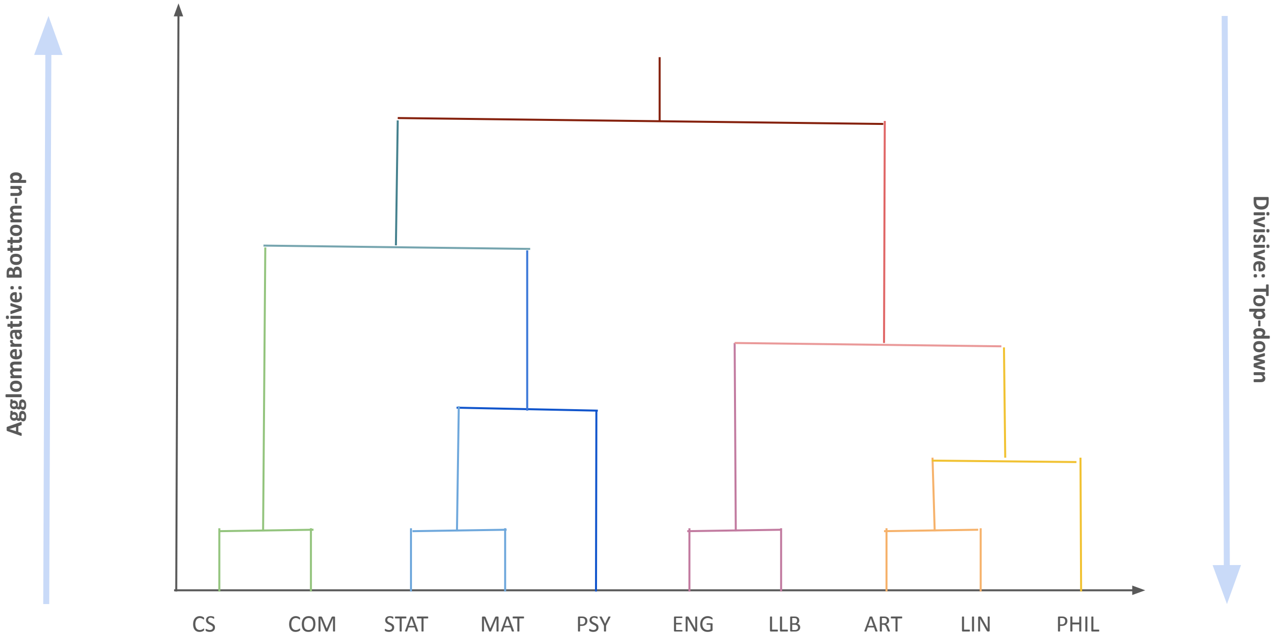
Standardisation

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_data, columns = df.columns)
```

Label Encoder

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
dataset['Gender'] = label_encoder.fit_transform(dataset['Gender'])
dataset['Gender'].unique()
```

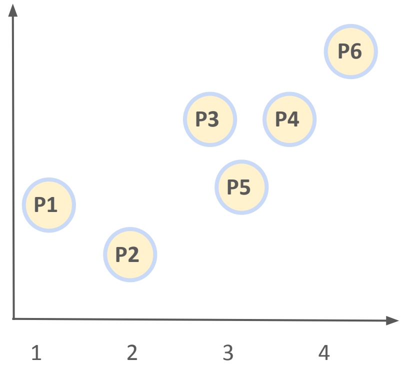
Hierarchical Clustering - Dendrogram

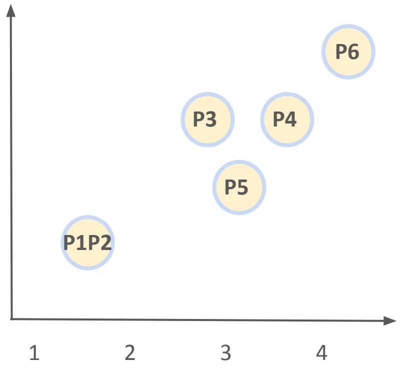


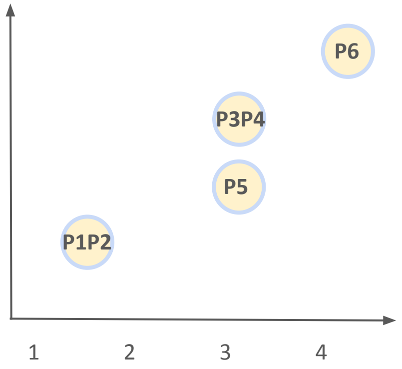
Finding K using Dendrogram

1. Scan the dendrogram to identify the longest vertical line that does NOT intersect with any horizontal lines (clusters)
2. Draw a horizontal line through it
3. Count the number of times the horizontal line intersects with the horizontal lines representing clusters in the dendrogram

1. Treat each data points as a separate cluster
2. Calculate the distance between each pair of clusters (Euclidean distance), resulting in an N x N distance matrix where the distance between a cluster and itself is zero







	P1	P2	P3	P4	P5	P6
P1	0					
P2	1	0				
P3	1.5	0.5	0			
P4	2.5	1.5	1	0		
P5	3	1	0.5	0.5	0	
P6	4	2	1.5	0.5	1	0

The shortest distance is P1 to P2, hence we should merge. We will then be left with 5 clusters. Again, recalculate the euclidean distance to get a 5x5 matrix

	P1P2	P3	P4	P5	P6
P1P2	0				
P3	1.5	0			
P4	2	1	0		
P5	1.5	0.1	0.5	0	
P6	2.5	1.5	0.5	1	0

The shortest distance is P3 and P4, hence we should merge. We will then be left with 4 clusters. Again, recalculate the euclidean distance to get a 4x4 matrix

	P1P2	P3P4	P5	P6
P1P2	0			
P3P4	1.5	0		
P5	1.5	0	0	
P6	2.5	1	1	0

The shortest distance is P3P4 and P5, hence we should merge. We will then be left with 4 clusters. Again, recalculate the euclidean distance to get a 4x4 matrix

WEEK 10

Feature Engineering

Dendrograms in Python

```
x = data[['Age', 'Income($)']]

from sklearn.preprocessing import StandardScaler()
sc = StandardScaler()
x = sc.fit_transform(x)
x = pd.DataFrame(x)
x.columns = ['Age', 'Income($)']                    # Standardise age and income to a computer-interpretable scale

from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering(n_clusters=3, linkage='single')
ypred = ac.fit_predict(x)

x['Cluster'] = ypred

import scipy.cluster.hierarchy as sch
dend = sch.dendrogram(sch.linkage(x, method='single'))

from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

linked = linkage(X_scaled, method='ward')
X['Cluster'] = fcluster(linked, t=3, criterion='maxclust')
```

Linear Regression

```
import pandas as pd
import matplotlib.pyplot as plt
stud_scores = pd.read_csv('score.csv')
stud_scores.head()
stud_scores.shape
import seaborn as sns
sns.relplot(x='Scores', y='Hours', data=stud_scores, height=3.8, aspect=1.8, kind='scatter')
sns.set_style('darkgrid')

X = stud_scores.iloc[:,:-1].values                    # feature matrix
y = stud_scores.iloc[:,1].values                      # response vector

# SPLITTING THE DATA
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

regressor.coef_
regressor.intercept_
y_pred = regressor.predict(X_test)
comparison_df = pd.DataFrame({"Actual":y_test,"Predicted":y_pred})

from sklearn.metrics import mean_squared_error
print("MSE",mean_squared_error(y_test,y_pred))

import numpy as np
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

Logistic Regression

```
sns.countplot(x=data['Pclass'],hue=data['Survived']);
sns.countplot(x=data['Sex'],hue=data['Survived']);
sns.countplot(x=data['Embarked'],hue=data['Survived']);

cols = ['PassengerId','Name','Ticket','Fare','Cabin']
data = data.drop(cols,axis=1)
data.isnull().sum()
mean_age = round(data['Age'].mean(),2)
data['Age'] = data['Age'].fillna(mean_age)
data.isnull().sum()
data = data.dropna()

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['Sex'] = encoder.fit_transform(data['Sex'])
data['Embarked'] = encoder.fit_transform(data['Embarked'])

y = data['Survived'].values
x = data.drop(['Survived'],axis=1).values

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,train_size=0.8,random_state=9014)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(xtrain,ytrain)
ypred = model.predict(xtest)

from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,confusion_matrix
confusion_matrix(ytest,ypred)
accu = accuracy_score(ytest,ypred)
f1_score(ytest,ypred)
precision_score(ytest,ypred)
recall_score(ytest,ypred)
```

KNN

KNN Classifier: When your target variable is categorical (e.g., 'spam' vs. 'not spam', or 'dog', 'cat', 'rabbit').

```
x=df.iloc[:,:4]
y = df.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=42)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)          # Predict on dataset which model has not seen before

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred)

from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
```

KNN Regressor: When your target variable is numerical (e.g., house price, temperature, age).

```
x = df.iloc[:, [1]].values
y = df.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=42)

from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors = 9)
model.fit(X_train, y_train)
y_pred=model.predict(X_test)       # Predict on dataset which model has not seen before

import math
import sklearn.metrics as skl_metrics
e1=skl_metrics.mean_squared_error(y_test, y_pred)
print("Mean Square Error=", e1)
error = math.sqrt(e1)
print("Root mean square error=", error)
```

K-Means Clustering