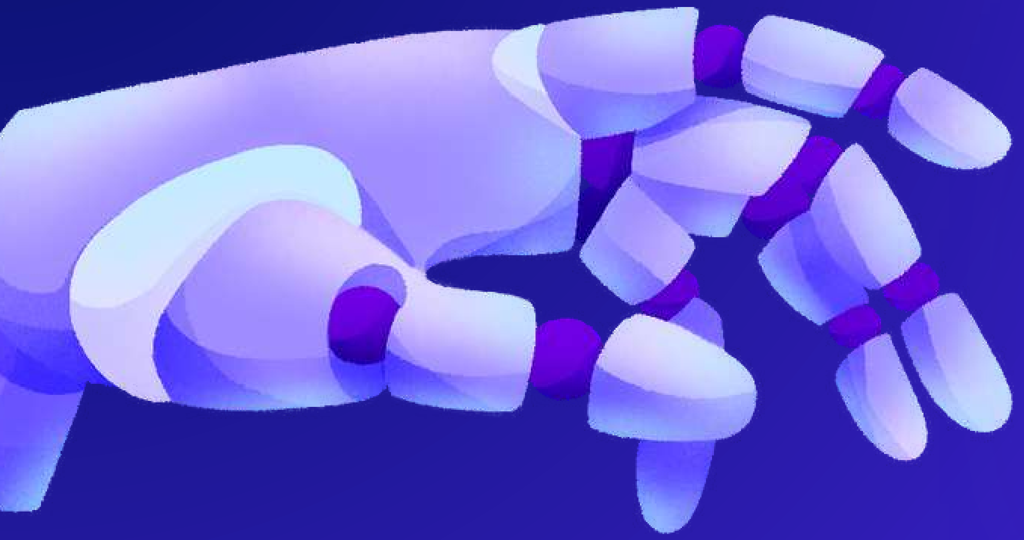
MACHINE LEARNING

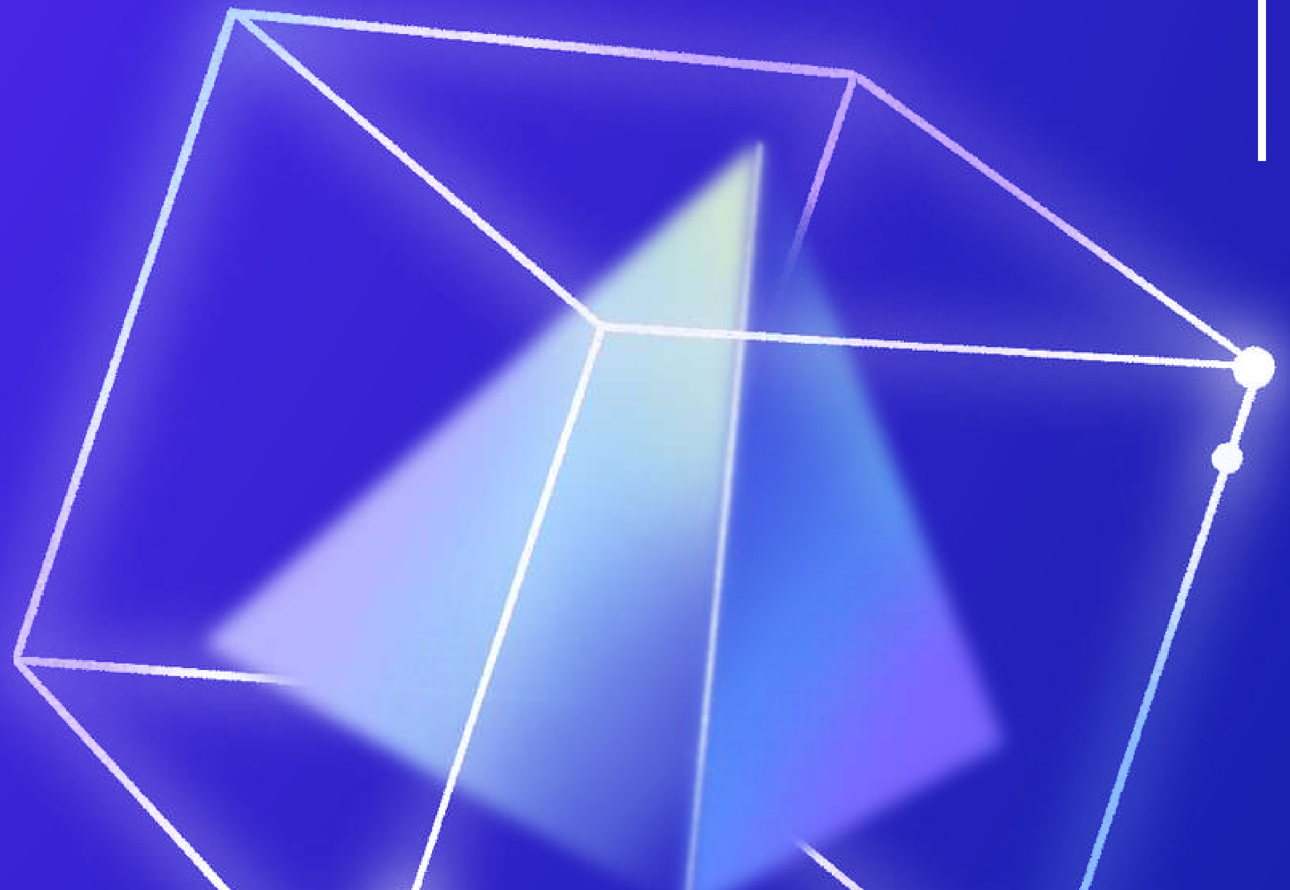
# PROJECT

## Based On “Cast”, “Crew”, “Genre”, “Overview” and “keyword” Movie Recommendation System & SMS Spam Detection

BY SHUBHAM VERMA(521243)
OMPRAKASH KUMAR(521207)
EEE DEPARTMENT
SEC(B) 3rd YEAR



# TABLE OF CONTENTS

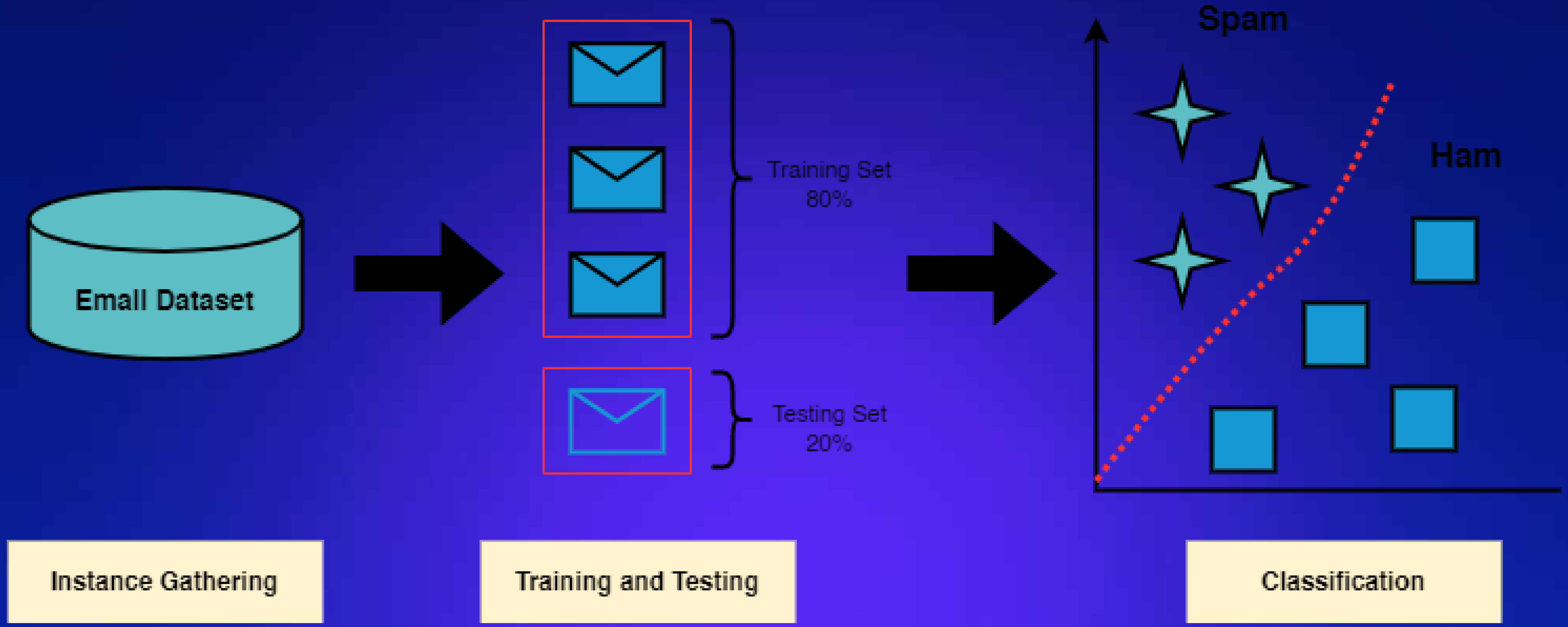


## Introduction

### SMS SPAM DETECTION

Our project focuses on using machine learning for SMS spam detection. In today's digital age, SMS spam poses privacy and security risks. We aim to develop a robust model to accurately distinguish spam from legitimate messages. Objectives include creating a labeled dataset, implementing ML algorithms, and evaluating model performance. Our solution uses supervised ML techniques on SMS data to enhance user privacy and security.

## WORK FLOW



Email Dataset
Training Set
80%
Testing Set
20%
Spam
Ham
Instance Gathering
Training and Testing
Classification

CORE IDEA

## 1. **Data Cleaning**:

- **Loading the dataset**: This step involves importing your dataset into your programming environment. In Python, this is typically done using libraries like pandas for tabular data or other specialized libraries for different data formats.
- **Handling missing values**: Missing data is a common issue in real-world datasets. Strategies for handling missing values include removing rows or columns with missing values, imputing missing values using mean or median, or more advanced techniques like predictive imputation.
- **Removing duplicates**: Duplicate records can skew analysis and modelling results. Identifying and removing duplicates ensures that each data point is unique and contributes meaningfully to the analysis with the help of pandas .
- **Converting data**: Categorical variables are need to be converted into numerical format for analysis and modelling. This could involve techniques like one-hot encoding for nominal variables or label encoding for ordinal variables.

## One-Hot Encoding:

▯ Usage: One-hot encoding is used for categorical variables where there is no ordinal relationship among the categories. Each category is represented by a binary vector where only one element is '1' (hot) and the rest are '0' (cold).

▯ Example: Consider a categorical variable "Colour" with three categories: Red, Green, and Blue. After one-hot encoding, each category becomes a binary feature: [1, 0, 0] for Red, [0, 1, 0] for Green, and [0, 0, 1] for Blue.

## Label Encoding:

▯ Usage: Label encoding is used for categorical variables with an ordinal relationship among the categories. Each category is assigned a unique integer label based on its order.

▯ Example: Consider a categorical variable "Size" with three categories: Small, Medium, and Large. After label encoding, Small may be encoded as 0, Medium as 1, and Large as 2.

Differences:

▯ Representation: One-hot encoding creates binary vectors for each category, resulting in a sparse matrix. Label encoding assigns integer labels to categories, resulting in a single column with integers.

## 2. **Exploratory Data Analysis (EDA)**:

**Analyzing data distribution**: Understanding the distribution of your data is crucial for selecting appropriate modelling techniques and interpreting results. For SMS spam detection, analyzing the distribution of spam vs. ham messages(()) helps gauge class imbalance.
 "spam" refers to unwanted or unsolicited messages, such as promotional emails or text messages,
 “ ham" messages refer to non-spam messages that users typically want to receive
**Visualizing data**: Data visualization techniques such as histograms, bar plots, and scatter plots help uncover patterns, trends, and anomalies in the data. Visualizations can reveal insights about message lengths, word frequencies, and other relevant features
 Visualization is a powerful tool for understanding and communicating patterns and insights in data. In Python, the matplotlib library is a popular choice for creating visualizations. Let's go through a basic example of how to visualize data using matplotlib.

## 3. **Text Preprocessing**:

- Text preprocessing refers to the process of cleaning and preparing text data for natural language processing (NLP) tasks. It involves transforming raw text into a format that is suitable for analysis by machine learning algorithms. Text preprocessing typically includes the following steps:

 **Tokenization**: Tokenization is the process of breaking down a text into smaller units called tokens. Tokens can be words, phrases, or even characters, depending on the granularity of analysis. The most common approach is to tokenize text into words using whitespace or punctuation as delimiters.

 **Lowercasing**: Convert all text to lowercase to ensure uniformity and avoid treating the same word differently due to differences in capitalization.

 **Removing Punctuation**: Remove punctuation marks such as commas, periods, exclamation marks, and question marks from the text. Punctuation usually does not carry meaningful information for many NLP tasks.

 **Removing Stop-words**: Stop-words are common words that occur frequently in a language but typically do not carry much meaning, such as "and", "the", "is". Removing stop-words helps reduce noise in the text data and can improve the performance of NLP models.

 **Stemming and Lemmatization**: Stemming and lemmatization are techniques used to reduce words to their base or root form. Stemming involves removing suffixes and prefixes from words to derive the root form, while lemmatization involves reducing words to their dictionary form (lemma). For example, "running" would be stemmed to "run" and lemmatized to "run".

 **Removing Numbers and Special Characters**: Remove numerical digits and other special characters from the text as they may not contribute to the analysis and could introduce noise.

 **Handling Contractions and Abbreviations**: Expand contractions (e.g., "can't" to "cannot") and replace abbreviations with their full forms (e.g., "U.S." to "United States") to ensure consistency in the text.

 **Normalization**: Normalize the text by removing extra spaces, correcting misspellings, and standardizing abbreviations or acronyms.

 **Feature Engineering**: Extract additional features from the text data, such as word frequency counts, n-grams (sequences of adjacent words), or part-of-speech tags, to capture more information for analysis.
 nltk is often used for more advanced text preprocessing tasks beyond basic cleaning and formatting.
 spacy: spacy is another popular NLP library in Python that provides efficient and high-performance text processing capabilities. spacy is known for its speed and accuracy in text processing tasks.
 scikit-learn: While primarily known for machine learning algorithms, scikit-learn also provides utilities for text preprocessing, such as TF-IDF vectorization, feature extraction, and feature selection

Text preprocessing is a critical step in the Natural Language Processing pipeline as it helps improve the quality of text data and enhances the performance of machine learning models trained on that data. The specific preprocessing steps applied may vary depending on the nature of the text data and the requirements of the NLP task at hand.

## 4. **Model Building**:

In SMS spam detection, model building involves creating and training machine learning or deep learning models to distinguish between spam and non-spam (ham) messages based on the content of SMS messages. Various techniques and libraries can be used for model building in SMS spam detection:

o **Feature Extraction**: Before building a model, text data needs to be Preprocessed and transformed into numerical features .

o **Model Selection**: Once the features are extracted, we can choose a suitable machine learning algorithm for classification. Common algorithms for text classification tasks include logistic regression, decision trees, random forests, support vector machines (SVM), naive Bayes, and more advanced techniques like gradient boosting and deep learning. `scikit-learn` Libraries , provide implementations of these algorithms for classification tasks.

o **Model Training**: After selecting a machine learning algorithm, the model needs to be trained on a labeled dataset of SMS messages. The dataset should be split into training and testing sets to evaluate the performance of the model. `scikit-learn` provides functions for splitting datasets and training models.

o **Evaluation**: Once the model is trained, it needs to be evaluated to assess its performance in classifying spam and ham messages. Common evaluation metrics for binary classification tasks include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). `scikit-learn` provides functions for computing these metrics.

o **Tuning and Optimization**: Depending on the performance of the initial model, hyperparameters may need to be tuned and optimization techniques applied to improve its performance. Techniques such as grid search and randomized search can be used to find the optimal hyperparameters for the model.

 Overall, libraries such as `NLTK`, `spacy`, `scikit-learn`, and `TensorFlow` or `PyTorch` (for deep learning) are commonly used for model building in SMS spam detection. These libraries provide a wide range of functionalities for text preprocessing, feature extraction, model selection, training, evaluation, and optimization, making them essential tools for building effective spam detection models.

## 5. **Evaluation of Model**:

- **Accuracy**: The proportion of correctly classified instances out of the total instances.
- **Precision**: The proportion of true positive predictions out of all positive predictions made by the model.
- **Recall**: The proportion of true positive predictions out of all actual positive instances in the dataset.
- **F1-score**: The harmonic mean of precision and recall, providing a single metric that balances both measures.
- **ROC-AUC**: Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at various threshold settings. The Area Under the ROC Curve (AUC) summarizes the ROC curve's performance into a single value.

These evaluation metrics can be calculated using libraries such as scikit-learn in Python. scikit-learn provides functions to compute these metrics from the model predictions and the true labels of the SMS messages.

## 6. **Improvement**:

Improvement in the context of model building refers to enhancing the performance or effectiveness of a machine learning model. This can involve improving its accuracy, precision, recall, F1 score, or other relevant evaluation metrics. There are several approaches to improving a model, including:

o **Feature Engineering**: Enhancing the quality of input features by creating new features, transforming existing features, or selecting the most informative features. Libraries such as `scikit-learn`, `feature-engine`, and `featuretools` can be used for feature engineering tasks.

o **Hyperparameter Tuning**: Fine-tuning the hyperparameters of the model to optimize its performance. Techniques such as grid search, randomized search, and Bayesian optimization can be used to search for the best combination of hyperparameters. Libraries such as `scikit-learn`, `Hyperopt`, and `Optuna` provide tools for hyperparameter tuning.

o **Ensemble Methods**: Combining multiple base models to create a stronger ensemble model. Ensemble methods such as bagging, boosting, and stacking can improve the robustness and generalization of the model. Libraries such as `scikitlearn` and `XGBoost` provide implementations of various ensemble methods.

o **Model Selection**: Experimenting with different types of machine learning algorithms or deep learning architectures to find the most suitable model for the task. Libraries such as `scikit-learn`, `TensorFlow`, and `PyTorch` offer a wide range of algorithms and models for different types of tasks.

o **Data Augmentation**: Generating additional training data by applying transformations such as rotation, translation, or noise addition to the existing data. Data augmentation techniques can help improve the model's ability to generalize to new data. Libraries such as `imgaug` and `albumentations` are commonly used for data augmentation in computer vision tasks, but similar techniques can be applied to text data as well.

o **Transfer Learning**: Leveraging pre-trained models on large datasets to extract useful features or fine-tune the model on a specific task. Transfer learning can be particularly effective when working with limited training data. Libraries such as `TensorFlow Hub` and `Hugging Face Transformers` provide pre-trained models for various NLP tasks.

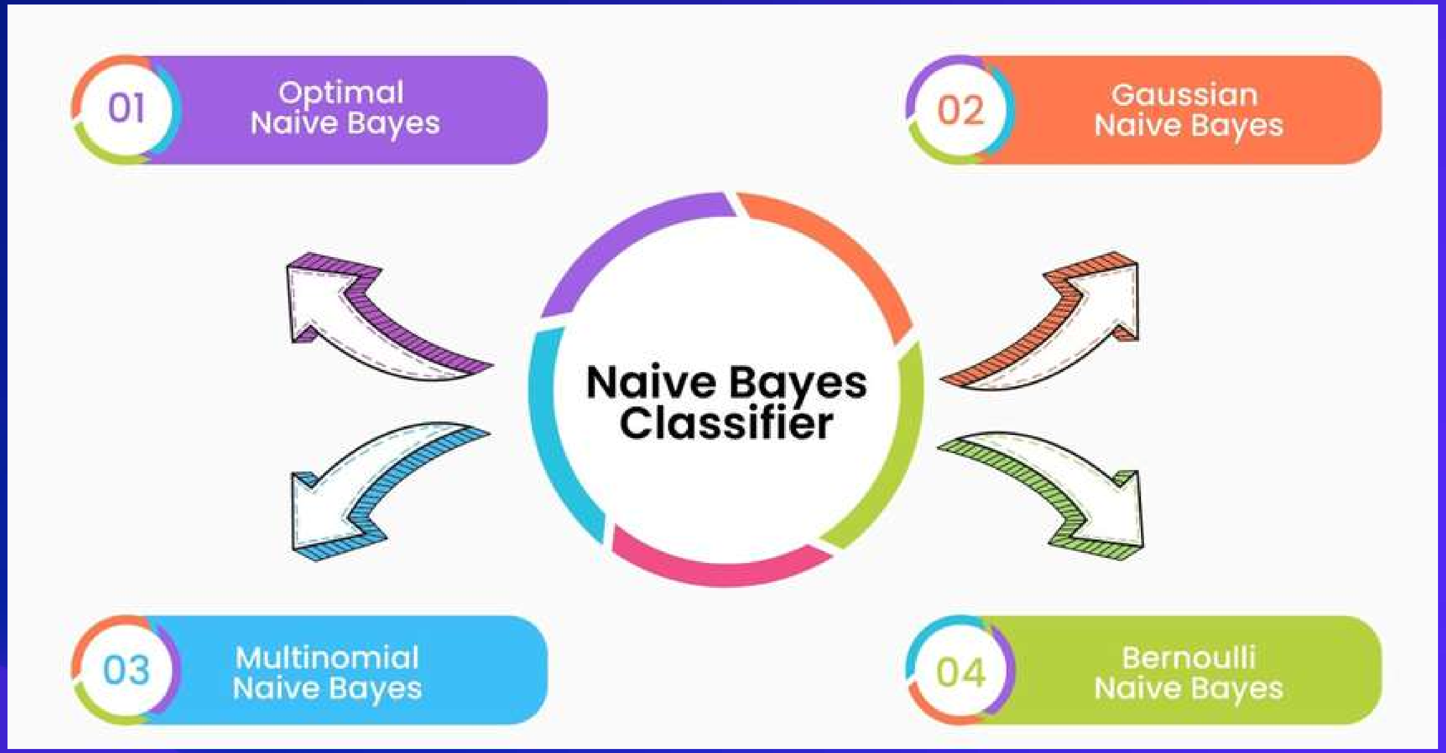
 Improvement in model performance often involves a combination of these approaches, and the choice of approach depends on the specific characteristics of the dataset and the task at hand. By iteratively experimenting with different techniques and evaluating their impact on the model's performance, you can gradually improve the effectiveness of your machine learning model.

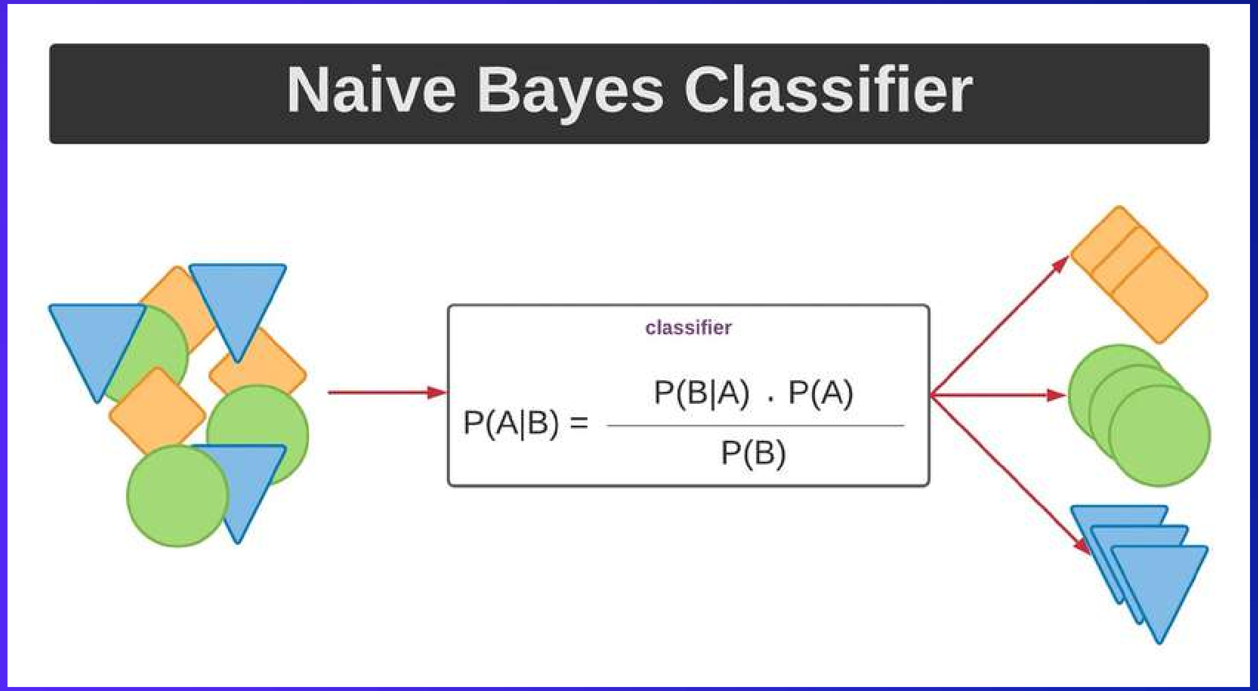
## About ML ALGO, LIBRARY USED

### NAIVE BAYES

Naive Bayes is a popular classification algorithm in machine learning, based on Bayes' theorem. It assumes that the features are independent of each other, hence the term "naive." Despite this simplifying assumption, Naive Bayes often performs well in practice, especially with text classification tasks like spam filtering and sentiment analysis.

The algorithm calculates the probability of a given instance belonging to each class based on the feature values. It selects the class with the highest probability as the predicted class for the instance. Naive Bayes is efficient, simple to implement, and works well with high-dimensional data, but it may not perform optimally if the independence assumption is violated or if the dataset is imbalanced.





- Exploratory Data Analysis (EDA) in machine learning involves analyzing and visualizing data to understand its structure, identify patterns, and detect anomalies before building a model. It helps in gaining insights into the data, selecting appropriate features, handling missing values, and determining the best approach for modeling. EDA often includes techniques like summary statistics, data visualization, correlation analysis, and dimensionality reduction.

In machine learning, the correlation coefficient measures the strength and direction of the linear relationship between two variables. It ranges from -1 to 1, where:

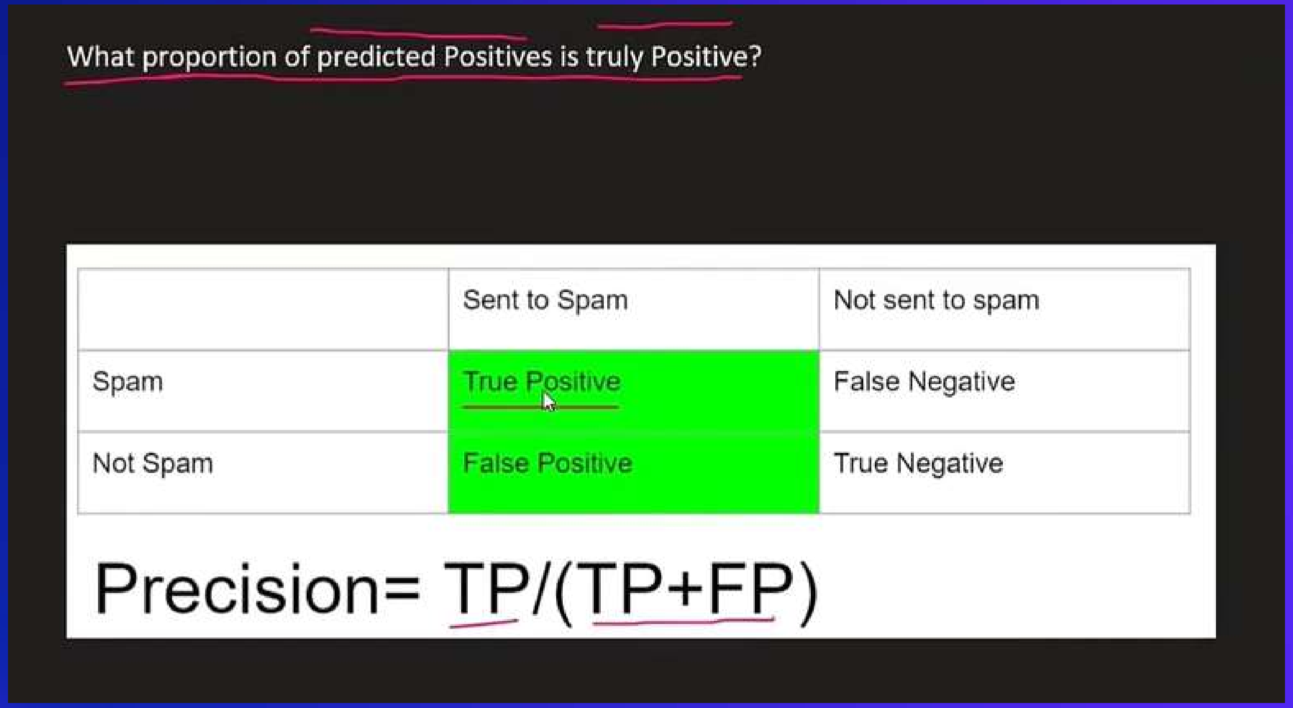
- 1 indicates a perfect positive linear relationship,
- -1 indicates a perfect negative linear relationship, and
- 0 indicates no linear relationship.

It helps in understanding how changes in one variable affect another, aiding feature selection and understanding the data's behavior.

- Multinomial Naive Bayes (MultinomialNB) is another variant of the Naive Bayes algorithm, specifically designed for classification tasks with discrete features. It's commonly used in text classification tasks, where the features represent the frequency or occurrence of words or other tokens.

- Accuracy score in machine learning is a metric used to measure the performance of a classification model. It calculates the percentage of correct predictions out of all predictions made by the model. In short, it tells you how often the model is correct. For example, if a model has an accuracy score of 0.8, it means it correctly predicts the label 80% of the time.
- Precision in machine learning is a metric that measures the accuracy of positive predictions made by a classification model. It is calculated as the ratio of true positive predictions to the total number of positive predictions, giving an indication of how many of the predicted positive cases are actually correct. In short, precision tells you how precise or accurate the model is when it predicts a positive outcome.

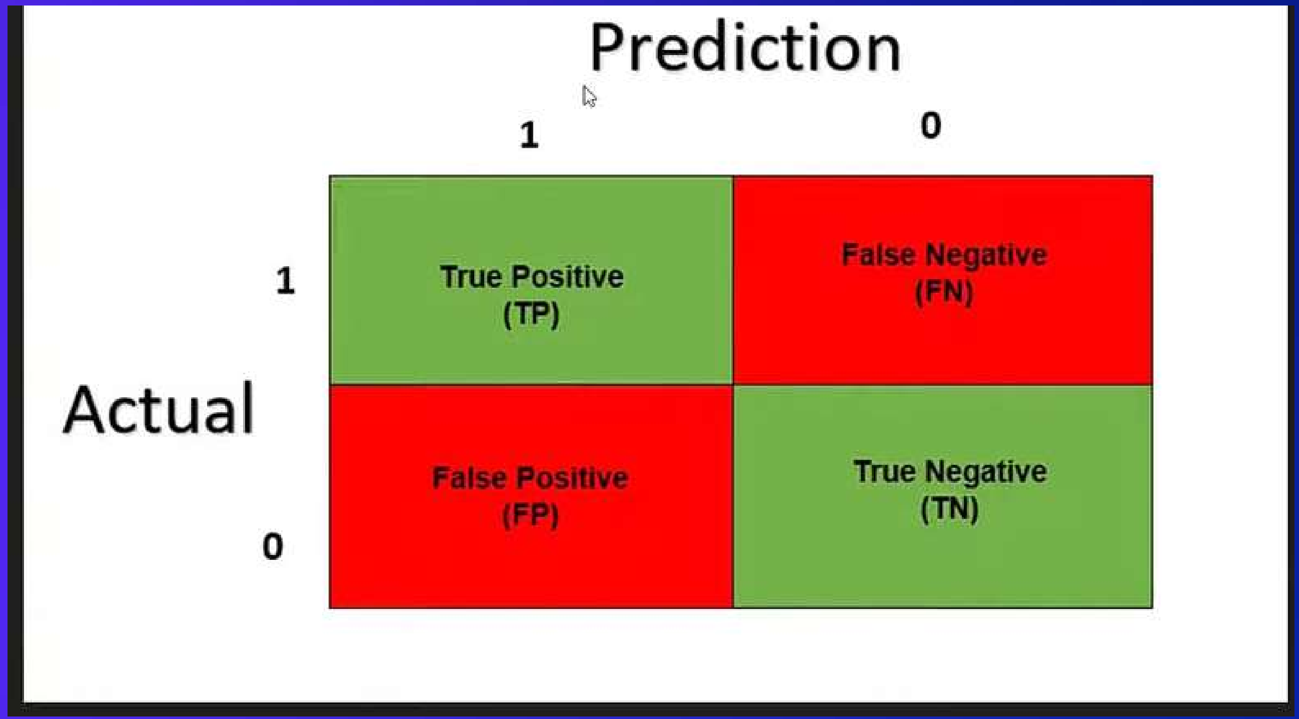
- A confusion matrix in machine learning is a table that shows the performance of a classification model. It compares the actual labels of the data with the predicted labels made by the model. It helps visualize the number of true positives, true negatives, false positives, and false negatives, providing insights into the model's accuracy and errors.



What proportion of predicted Positives is truly Positive?

| | Sent to Spam | Not sent to spam |
|---|---|---|
| Spam | True Positive | False Negative |
| Not Spam | False Positive | True Negative |

Precision= TP/(TP+FP)



| | | Prediction | |
|---|---|---|---|
| | | 1 | 0 |
| Actual | 1 | True Positive (TP) | False Negative (FN) |
| | 0 | False Positive (FP) | True Negative (TN) |

- TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer is a technique used in natural language processing (NLP) to convert text data into numerical vectors.

- Logistic regression is a machine learning algorithm used for binary classification tasks, where the output is either one of two classes, like "yes" or "no," "spam" or "not spam." Despite its name, it's used for classification, not regression.

- SVC stands for Support Vector Classifier, which is a type of supervised learning algorithm used for classification tasks.

- Decision Tree Classifier is a versatile algorithm that's easy to understand and implement, making it a popular choice for classification tasks in various domains.
- KNN is a simple yet effective algorithm for classification, particularly suitable for small to medium-sized datasets where distances between data points are meaningful.

- Random Forest Classifier harnesses the power of multiple decision trees to improve predictive accuracy and handle complex data with ease.
- AdaBoost Classifier builds a strong model by sequentially training weak learners on increasingly focused subsets of the data, resulting in improved accuracy.

- Bagging Classifier reduces variance and improves the stability of the model by averaging predictions from multiple models trained on different subsets of the data.
- Extra Trees Classifier is a variant of Random Forest that introduces additional randomness during tree construction, potentially resulting in faster training and a more diverse ensemble of trees.
- Gradient Boosting Classifier sequentially improves the model by minimizing errors using a series of weak learners, resulting in a powerful and accurate classifier.
- XGBoost Classifier is a powerful algorithm that leverages gradient boosting to produce accurate predictions efficiently, making it a popular choice for classification problems in machine learning.
- Scaling using MinMaxScaler is a preprocessing technique in machine learning used to scale features to a specific range, typically between 0 and 1.
- A Voting Classifier is an ensemble learning technique in machine learning where multiple individual classifiers are combined to make predictions. Each classifier gives its prediction, and the final prediction is determined by a majority vote (for classification tasks) or averaging (for regression tasks) among the individual classifiers. This helps improve prediction accuracy and robustness by leveraging the collective wisdom of multiple models.

## CODE EXPLANATION

- Data cleaning

- df.info(): This method provides information about the Data-Frame's structure,
- df.drop(): This method is used to remove rows or columns from the DataFrame based on specified labels.
- df.sample(): This method is used to generate a random sample of rows from the DataFrame.
- df.isnull().sum() is used to count the number of missing values (null values) in each column of a DataFrame df
- df.duplicated().sum() is used to count the number of duplicate rows in a DataFrame df.
- df.drop_duplicates(): This method removes duplicate rows from the DataFrame. By default, it keeps the first occurrence of each duplicated row

- EDA

- df.head() will display the first few rows of the DataFrame df
- df['target'].value_counts() is used to count the occurrences of each unique value in the 'target' column of the DataFrame
- Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
- pyplot: matplotlib.pyplot is a module within Matplotlib that provides a collection of functions that enable users to create plots easily

- NLTK is used:

- ▯ Tokenization: Breaking text into words or sentences.
- ▯ Stemming and Lemmatization: Reducing words to their base or root form.
- ▯ Part-of-Speech (POS) Tagging: Assigning grammatical categories (such as noun, verb, adjective) to words in a sentence.
- ▯ Named Entity Recognition (NER): Identifying and categorizing named entities (such as person names, organization names, etc.) in text.
- ▯ Parsing: Analyzing the syntactic structure of sentences.
- ▯ WordNet Integration: Accessing WordNet, a lexical database of English words, and utilizing its features such as synonyms, antonyms, and hypernyms.
- ▯ Text Classification: Classifying text into predefined categories or labels.
- ▯ Sentiment Analysis: Analyzing the sentiment or opinion expressed in text.
- ▯ Language Detection: Identifying the language of a given text.
- ▯ Corpus Handling: Working with text corpora for training and evaluating NLP models.

- Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics

-  sns.heatmap(numeric_df.corr(), annot=True)   Using the  Seaborn  create a heatmap of the correlation matrix for a DataFrame
-  string.punctuation is a string constant defined in Python's built-in string module. It contains a string of punctuation characters commonly used in English text
-  nltk.download('stopwords')   Downloading package stopword
-  nltk.corpus   is a module in the NLTK (Natural Language Toolkit) library for Python. It provides access to a collection of textual corpora (plural of corpus) and lexical resources that are commonly used in natural language processing (NLP) tasks
-  nltk.stem.porter   Stemming is the process of reducing words to their base or root form, which helps in reducing the dimensionality of the feature space and improving the performance of text processing and analysis tasks.

- transform_text function you provided earlier is a text preprocessing function that performs several common text cleaning operations.

-  Wordcloud is a popular visualization technique used to represent text data. It displays a collection of words, where the size of each word corresponds to its frequency or importance in the text.

-  collections module in Python provides a collection of specialized container datatypes that are alternatives to the built-in containers such as lists, dictionaries, and sets.

-  sklearn.feature_extraction.text is a module in the scikit-learn library (often abbreviated as sklearn) that provides tools for feature extraction from text data. It includes methods for converting raw text data into numerical feature vectors, which can be used as input for machine learning algorithms.

- sklearn.model_selection is a module in the scikit-learn library that provides tools for model selection and evaluation in machine learning
-
- □ sklearn.naive_bayes module in scikit-learn provides implementations of various Naive Bayes classifiers
-
- □ sklearn.metrics module in scikit-learn provides a wide range of performance metrics for evaluating the performance of machine learning models
-
- □ from sklearn.naive_bayes import GaussianNB as GNB, MultinomialNB as MNB, BernoulliNB as BNB
- □ from sklearn.metrics import accuracy_score as acc, confusion_matrix as cm, precision_score as ps
- □ gnb, mnb, bnb = GaussianNB(), MultinomialNB(), BernoulliNB()

- y_pred1 = gnb.fit(X_train, y_train).predict(X_test)
- □ print(acc(y_test, y_pred1))
- □ print(cm(y_test, y_pred1))
- • print(ps(y_test, y_pred1))  fit() method is used to fit the Gaussian Naive Bayes classifier (gnb) to the training data (X_train, y_train) and then predict() method is used to make predictions on the test data (X_test).
- □
- y_pred3 = bnb.fit(X_train, y_train).predict(X_test)
- print(acc(y_test, y_pred3))
- print(cm(y_test, y_pred3))
- • print(ps(y_test, y_pred3))  The Bernoulli Naive Bayes classifier (bnb) is fitted to the training data (X_train, y_train) using the fit() method, and then predictions are made on the test data (X_test) using the predict() method.

- from sklearn.linear_model import LogisticRegression as LR
- from sklearn.svm import SVC
- from sklearn.naive_bayes import MultinomialNB as MNB
- from sklearn.tree import DecisionTreeClassifier as DT
- from sklearn.neighbors import KNeighborsClassifier as KNN
- from sklearn.ensemble import RandomForestClassifier as RF
- from sklearn.ensemble import AdaBoostClassifier as AdaBoost
- from sklearn.ensemble import BaggingClassifier as Bagging
- from sklearn.ensemble import ExtraTreesClassifier as ExtraTrees
- from sklearn.ensemble import GradientBoostingClassifier as GBoost
- from xgboost import XGBClassifier as XGB

clfs = {'SVC': svc, 'KN': knc, 'NB': mnb, 'DT': dtc, 'LR': lrc, 'RF': rfc, 'AdaBoost': abc, 'BgC': bc, 'ETC': etc, 'GBDT': gbdt, 'xgb': xgb}

• The function train_classifier takes the classifier clf, training data X_train and y_train, and test data X_test and y_test as input.

• It fits the classifier to the training data, makes predictions on the test data, and then returns the accuracy and precision scores calculated using the abbreviated function names acc and ps.

• acc represents accuracy_score and ps represents precision_score.

The function train_classifier is called with the parameters svc (which represents the Support Vector Classifier), training data X_train and y_train, and test data X_test and y_test.

- This function fits the Support Vector Classifier to the training data, makes predictions on the test data, and returns the accuracy and precision scores.
- • Two empty lists accuracy_scores and precision_scores are initialized to store the accuracy and precision scores, respectively.
- • A loop iterates over the items of the dictionary clfs, where each item consists of a classifier name (name) and the corresponding classifier object (clf).
- • For each classifier, the train_classifier function is called with the classifier object and the training and test data.
- • The accuracy and precision scores returned by the train_classifier function are printed along with the classifier name.
- • The accuracy and precision scores are appended to the accuracy_scores and precision_scores lists, respectively.

A DataFrame named performance_df is created using pd.DataFrame().
- The DataFrame is constructed using a dictionary where:
- 'Algorithm': Represents the keys of the clfs dictionary (classifier names).
- 'Accuracy': Represents the accuracy scores stored in the accuracy_scores list.
- 'Precision': Represents the precision scores stored in the precision_scores list.
- The DataFrame is then sorted by the 'Precision' column in descending order using sort_values().

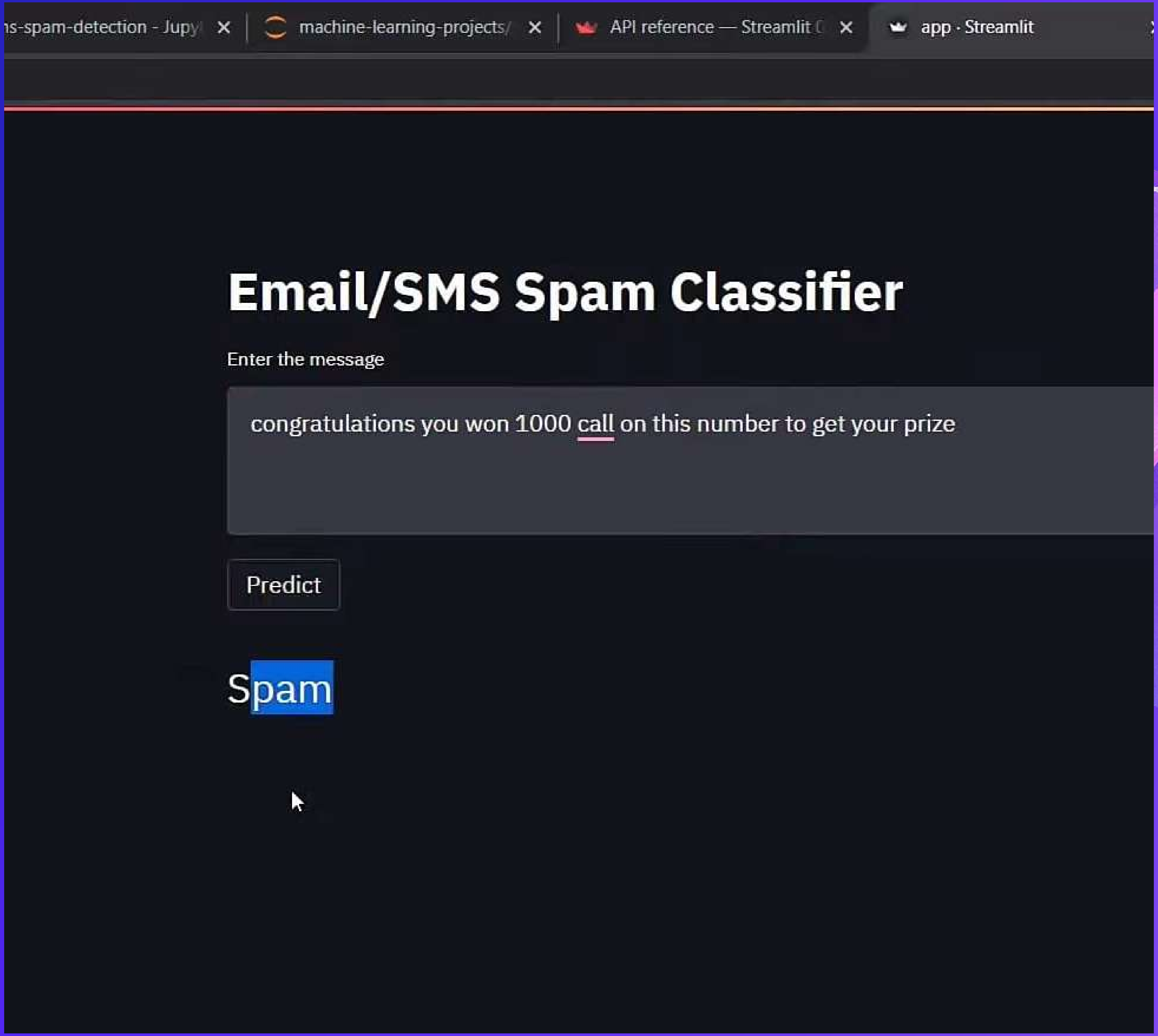
The pd.melt() function is used to reshape the DataFrame performance_df.
- The id_vars parameter specifies the column(s) to be kept as identifier variables (not melted).
- In this case, the identifier variable is specified as "Algorithm", meaning it will remain unchanged while the other columns are melted.
- sns.catplot() is used to create a categorical plot (bar plot).
- The x-axis represents the 'Algorithm' column, the y-axis represents the 'value' column, and the 'variable' column is used to distinguish between accuracy and precision scores.
- The data for the plot is provided by the DataFrame performance_df1.

A DataFrame is created using pd.DataFrame() with the specified columns.
• The 'Algorithm' column is assigned the keys of the clfs dictionary.
• The 'Accuracy_max_ft_3000' column is assigned the accuracy scores stored in the accuracy_scores list.

A DataFrame named temp_df is created using pd.DataFrame() with the specified columns.
• The 'Algorithm' column is assigned the keys of the clfs dictionary.
• The 'Accuracy_scaling' column is assigned the accuracy scores stored in the accuracy_scores list.
• The 'Precision_scaling' column is assigned the precision scores stored in the precision_scores list.
• The DataFrame is sorted by the 'Precision_scaling' column in descending order using sort_values().

The VotingClassifier class from the sklearn.ensemble module is imported with an alias VC.
• Three classifiers (SVC, MultinomialNB, ExtraTreesClassifier) are instantiated with their respective parameters.
• The fit() method is called on the voting classifier object to train the ensemble model using the training data X_train and y_train.
• Predictions are made on the test data X_test using the predict() method of the voting classifier object, and the results are stored in y_pred.
• A list of tuples named estimators is created. Each tuple contains a string identifier for the estimator and the corresponding classifier object.
• The fit() method is called on the classifier (clf) to train it using the training data (X_train, y_train), and then the predict() method is immediately called on the test data (X_test) to make predictions.
•
• The pickle.dump() function is used to serialize (i.e., save) the tfidf object to a file named 'vectorizer.pkl' in binary mode ('wb').
• Similarly, the mnb object is serialized and saved to a file named 'model.pkl' in binary mode.

# WEBSITE DEMO



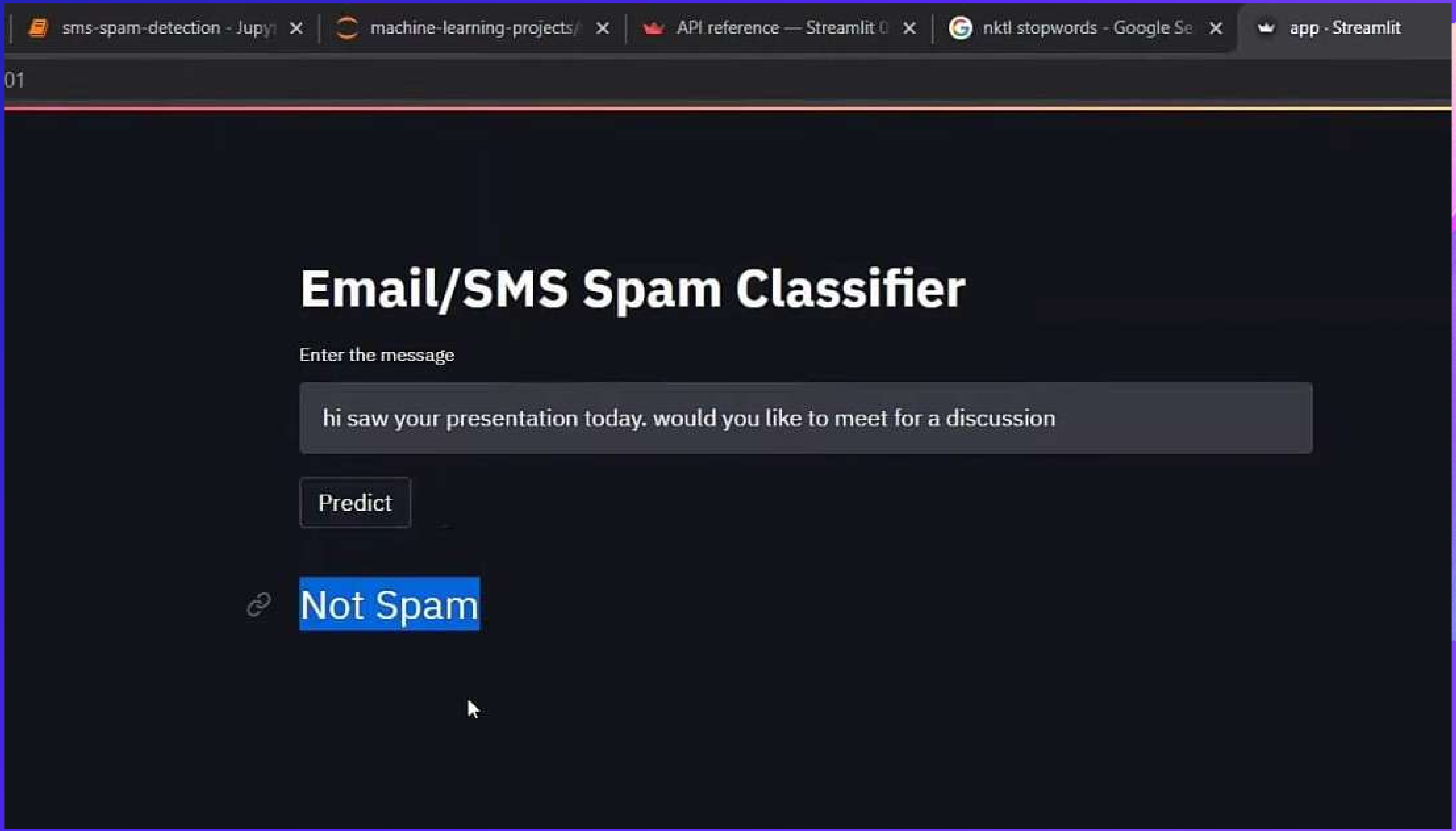
## Email/SMS Spam Classifier

Enter the message

congratulations you won 1000 call on this number to get your prize

Predict

Spam


sms-spam-detection - Jupy
machine-learning-projects/
API reference — Streamlit
nktl stopwords - Google Se
app · Streamlit
01

Email/SMS Spam Classifier

Enter the message

hi saw your presentation today. would you like to meet for a discussion

Not Spam

## GITHUB LINK

- Dataset

#spam.csv https://drive.google.com/file/d/1xS-uDIw9UXMKlf4EDHgu8oEAQWVF6kSW/view

- PyCharm

#website https://github.com/verma3shubham/sms_detect_pycharm

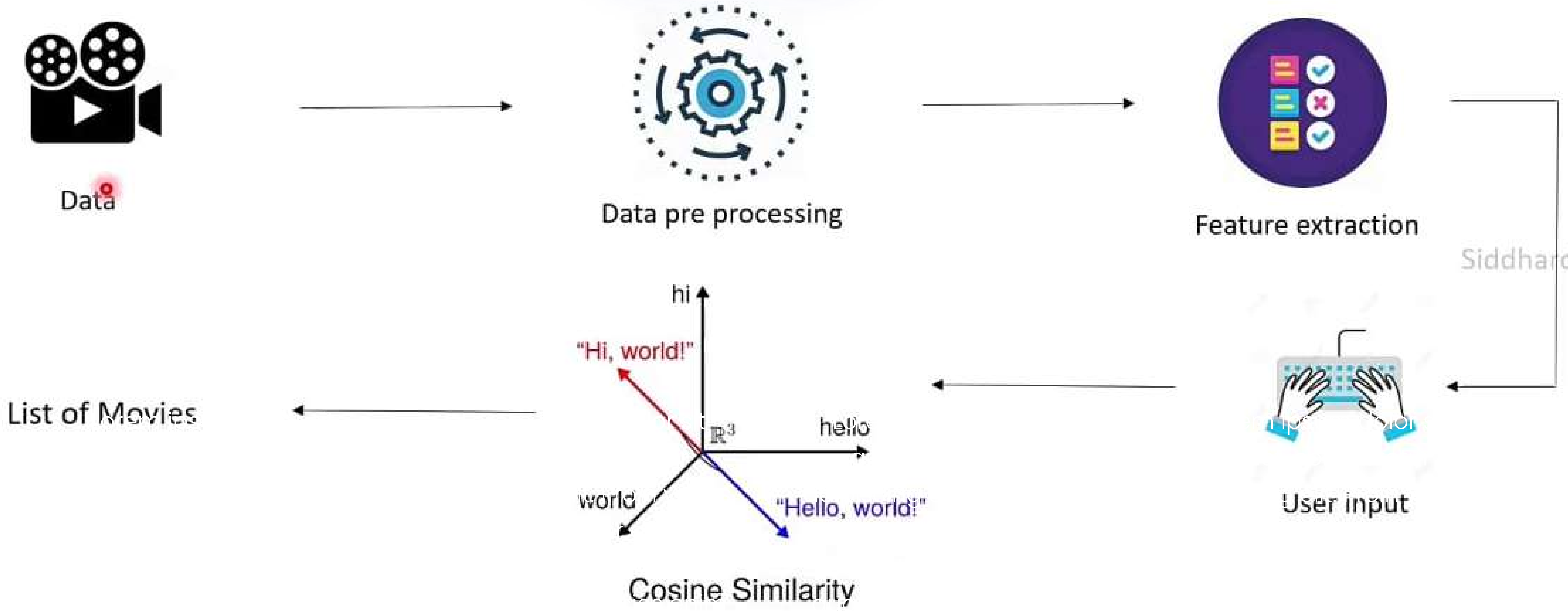
- JUPYTER

#code https://github.com/omprakash1207/sms-spam-detection/blob/main/sms_spam_detection.ipynb

# RECOMMENDATION SYSTEM

1. Content Based Recommendation System
2. Popularity Based Recommendation System
3. Collaborative Based Recommendation System

# WORK FLOW



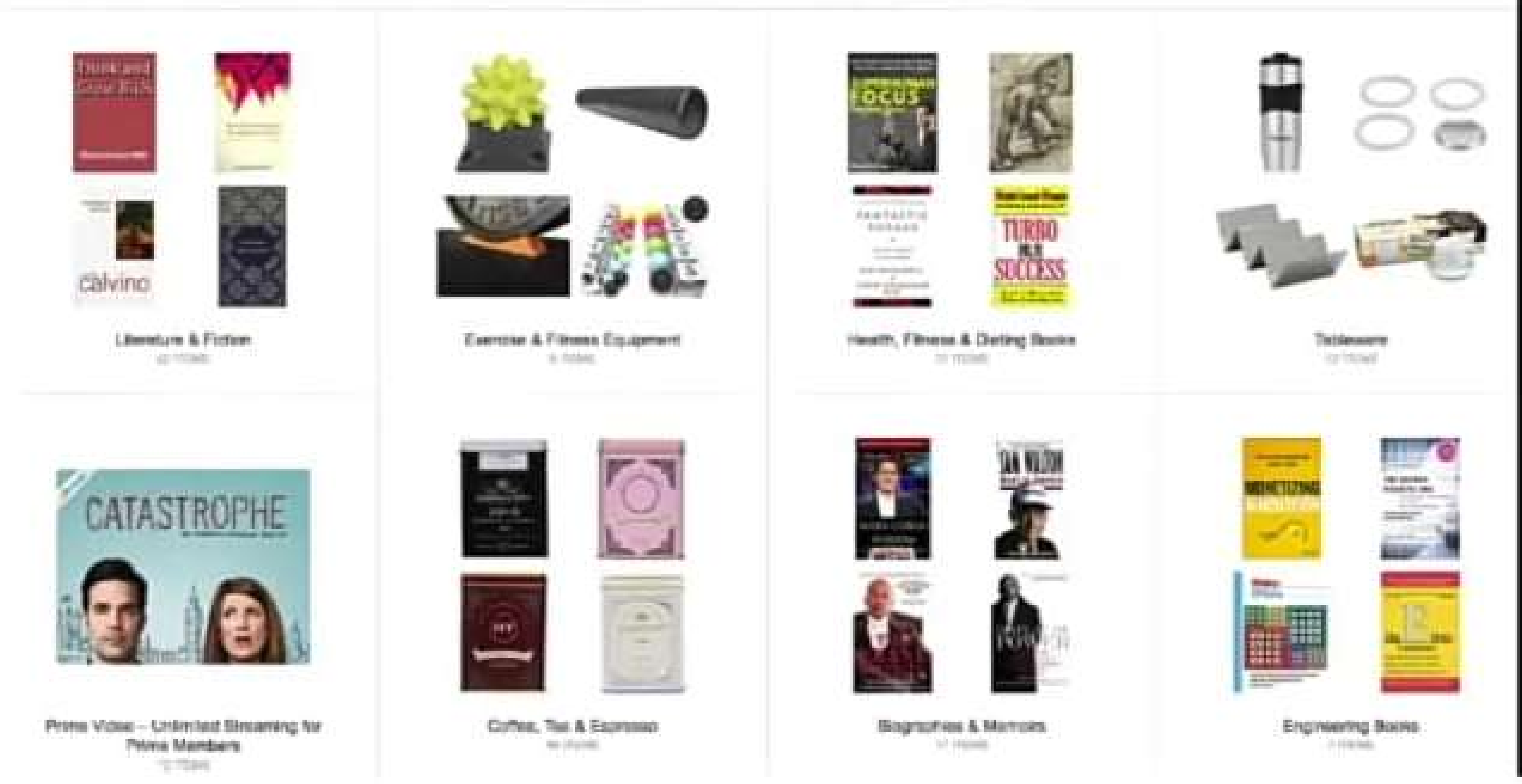
Cosine Similarity Algorithm

# INtro

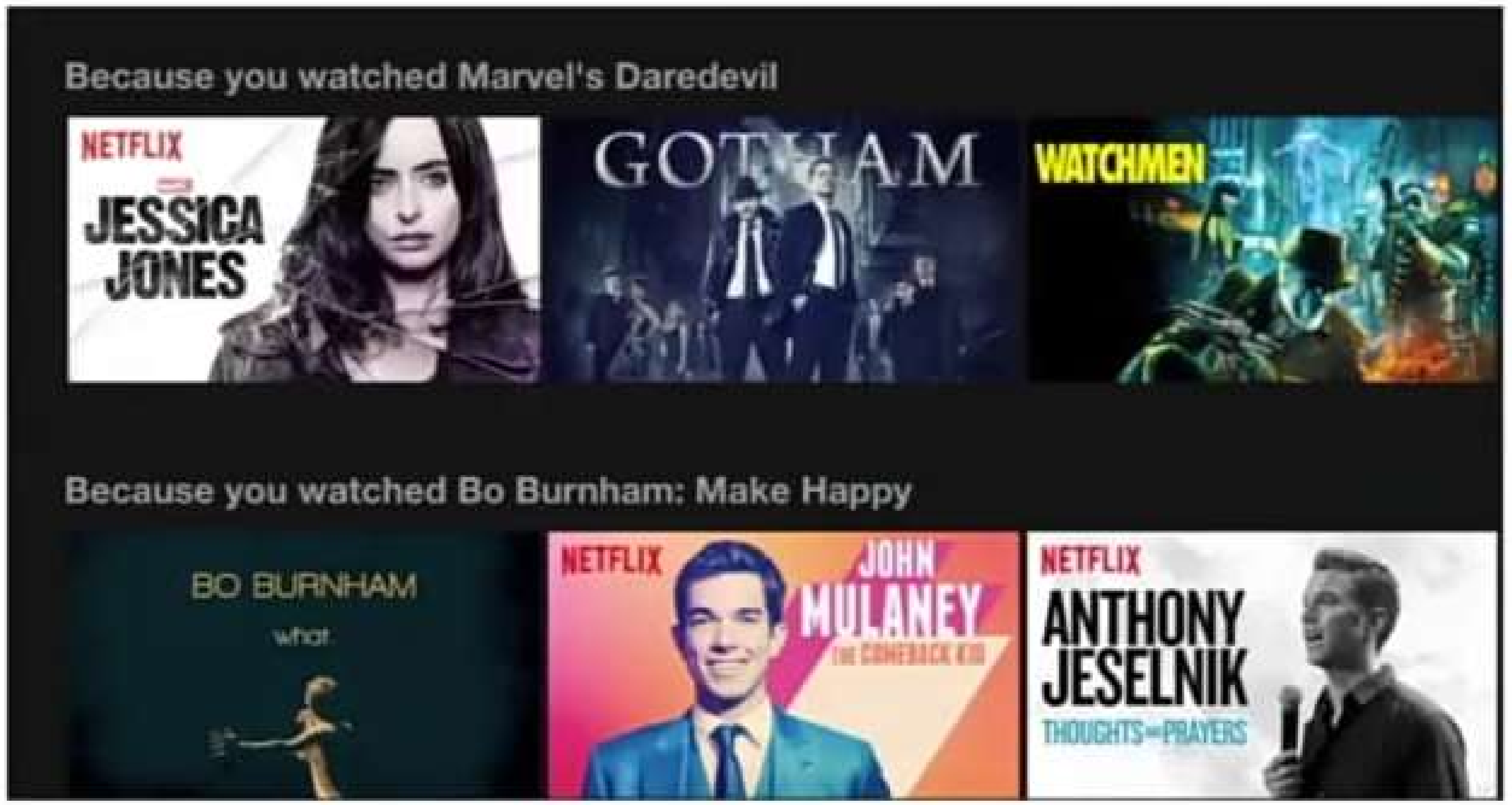
## What's Common?

1. Amazon

Recommended for you, Thomas

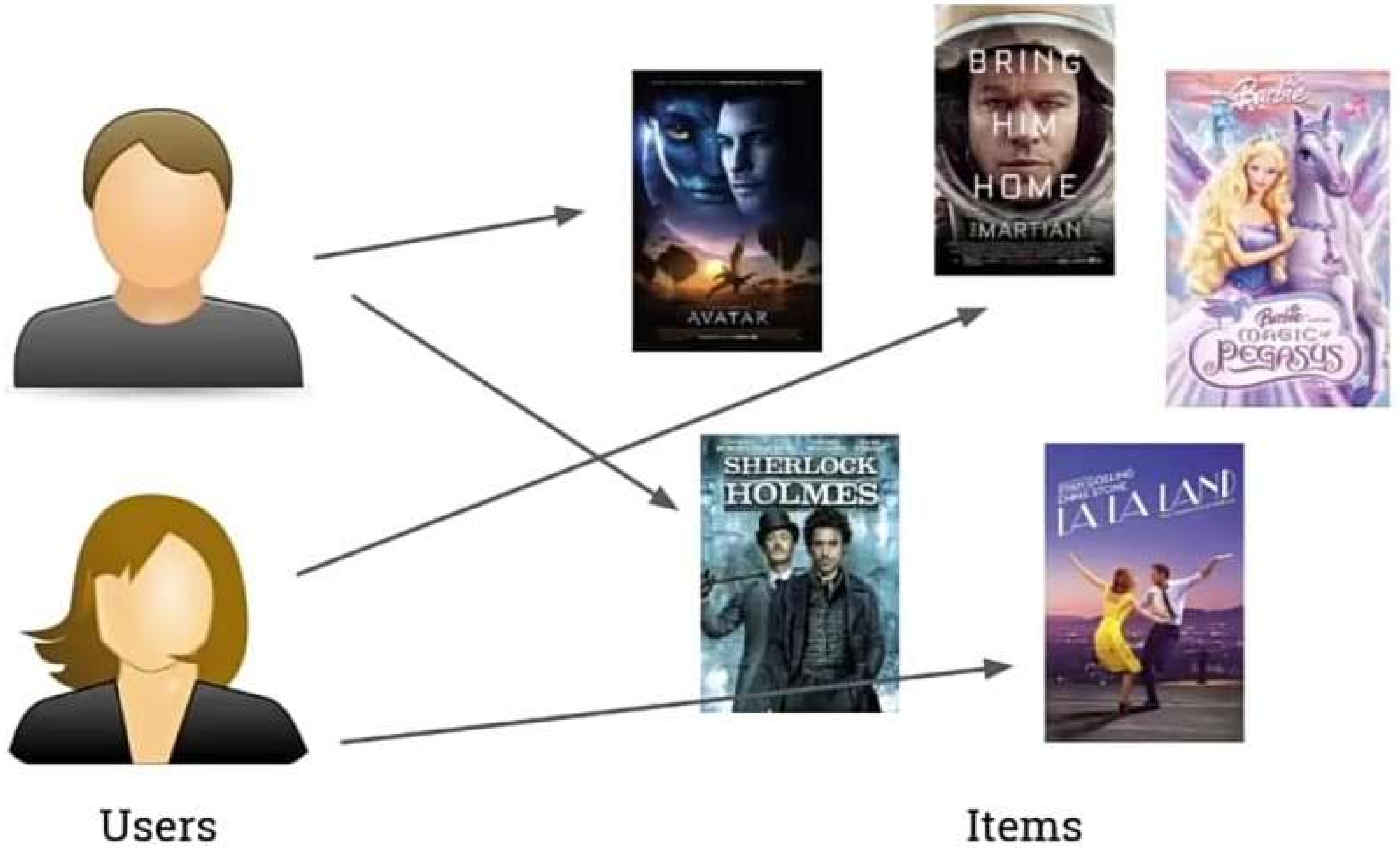

calvino
Literature & Fiction
Exercise & Fitness Equipment
Health, Fitness & Dieting Books
Tableware
CATASTROPHE
Prime Video – Unlimited Streaming for Prime Members
Coffee, Tea & Espresso
Biographies & Memoirs
Engineering Books

2. Netflix

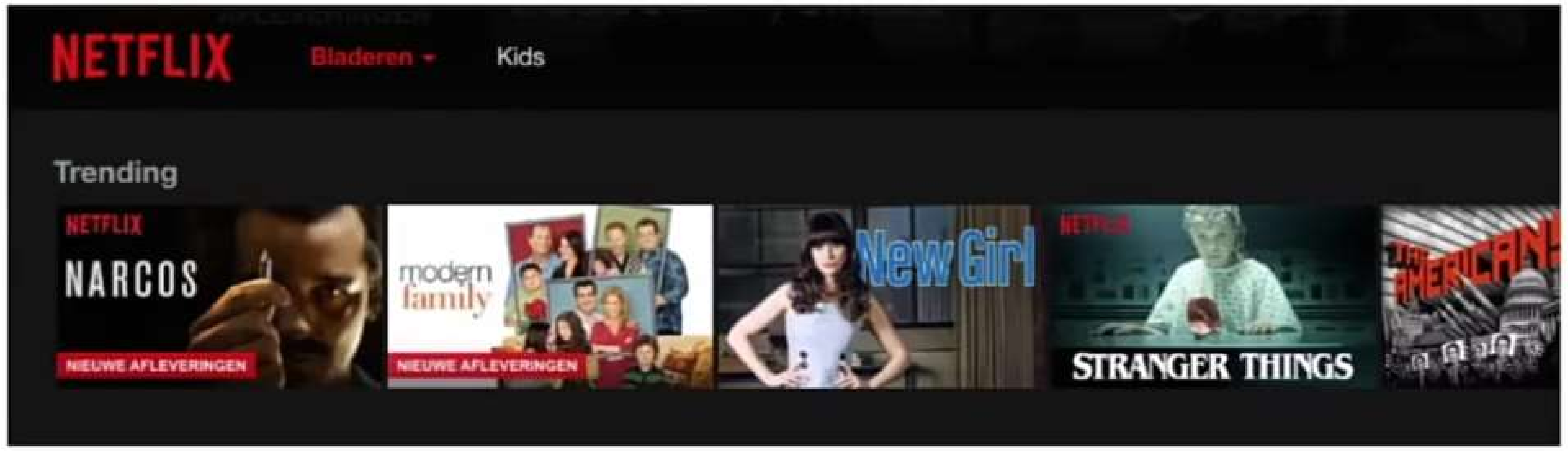

Because you watched Marvel's Daredevil

Because you watched Bo Burnham: Make Happy

## Recommendation Systems

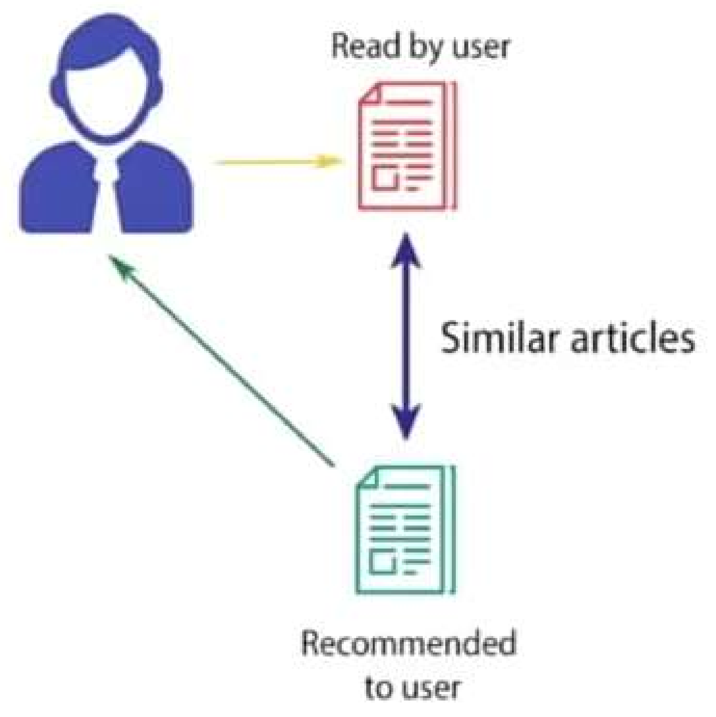


## Implementing A Recommender System

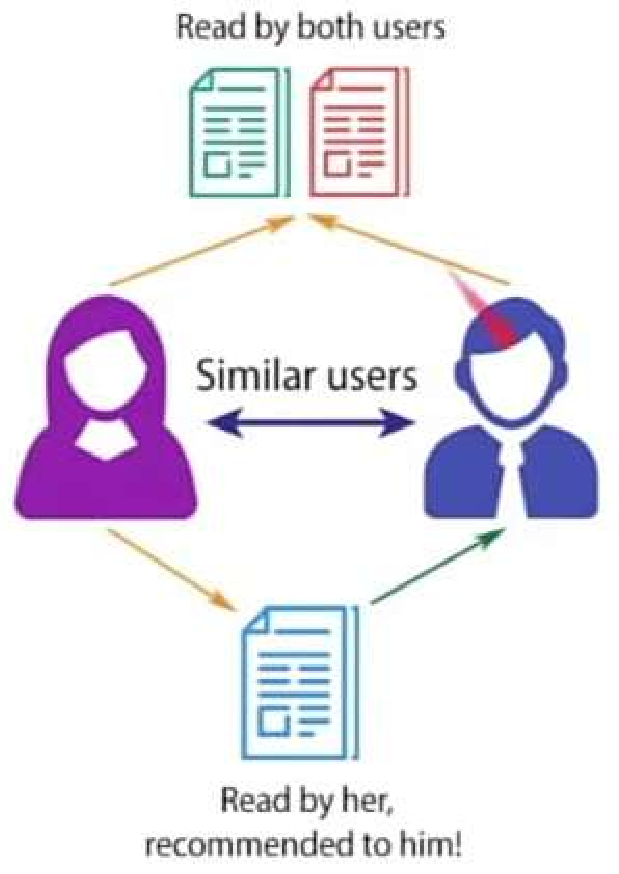


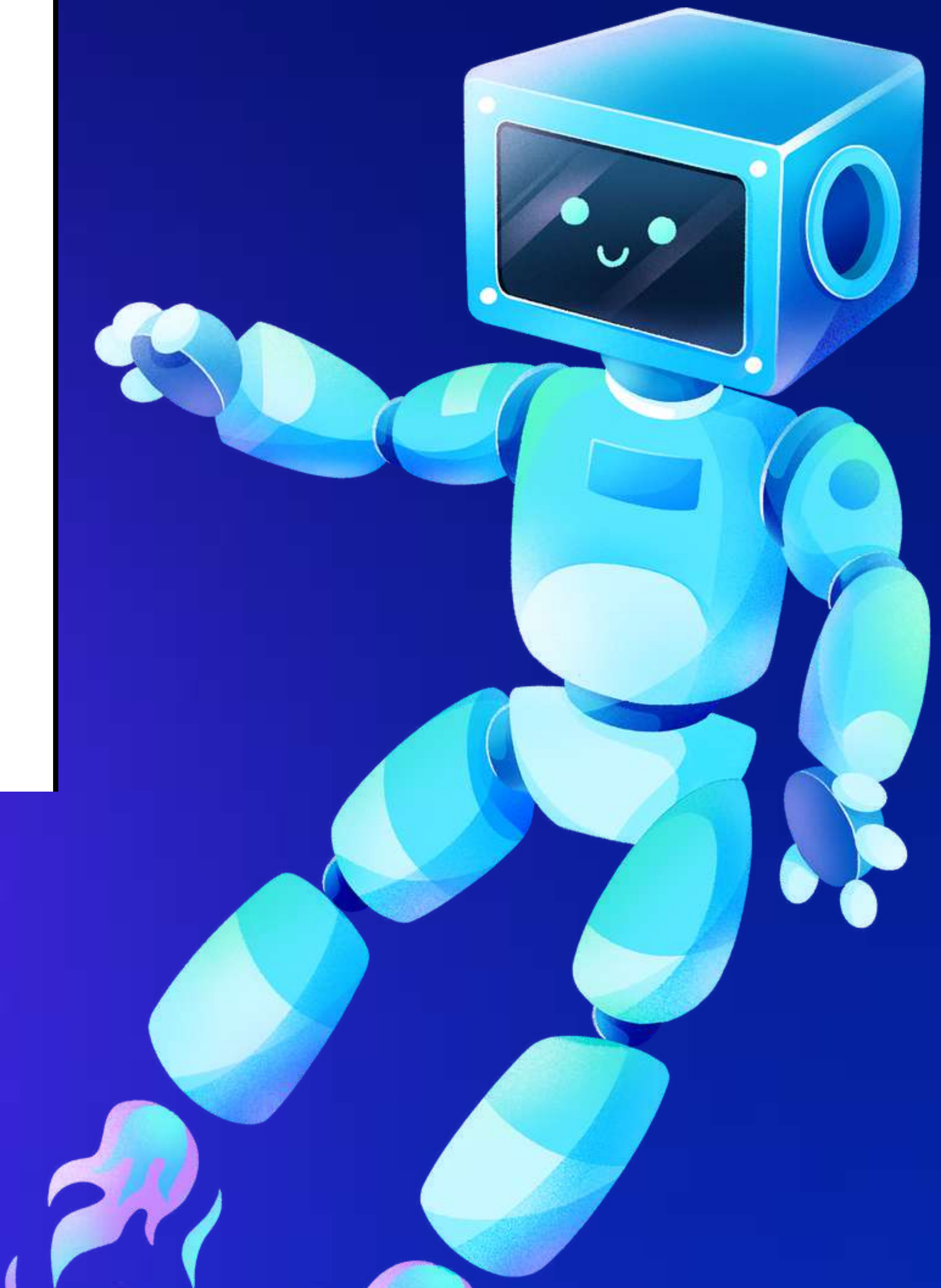
## Implementing A Recommender System

### 2. Content Based

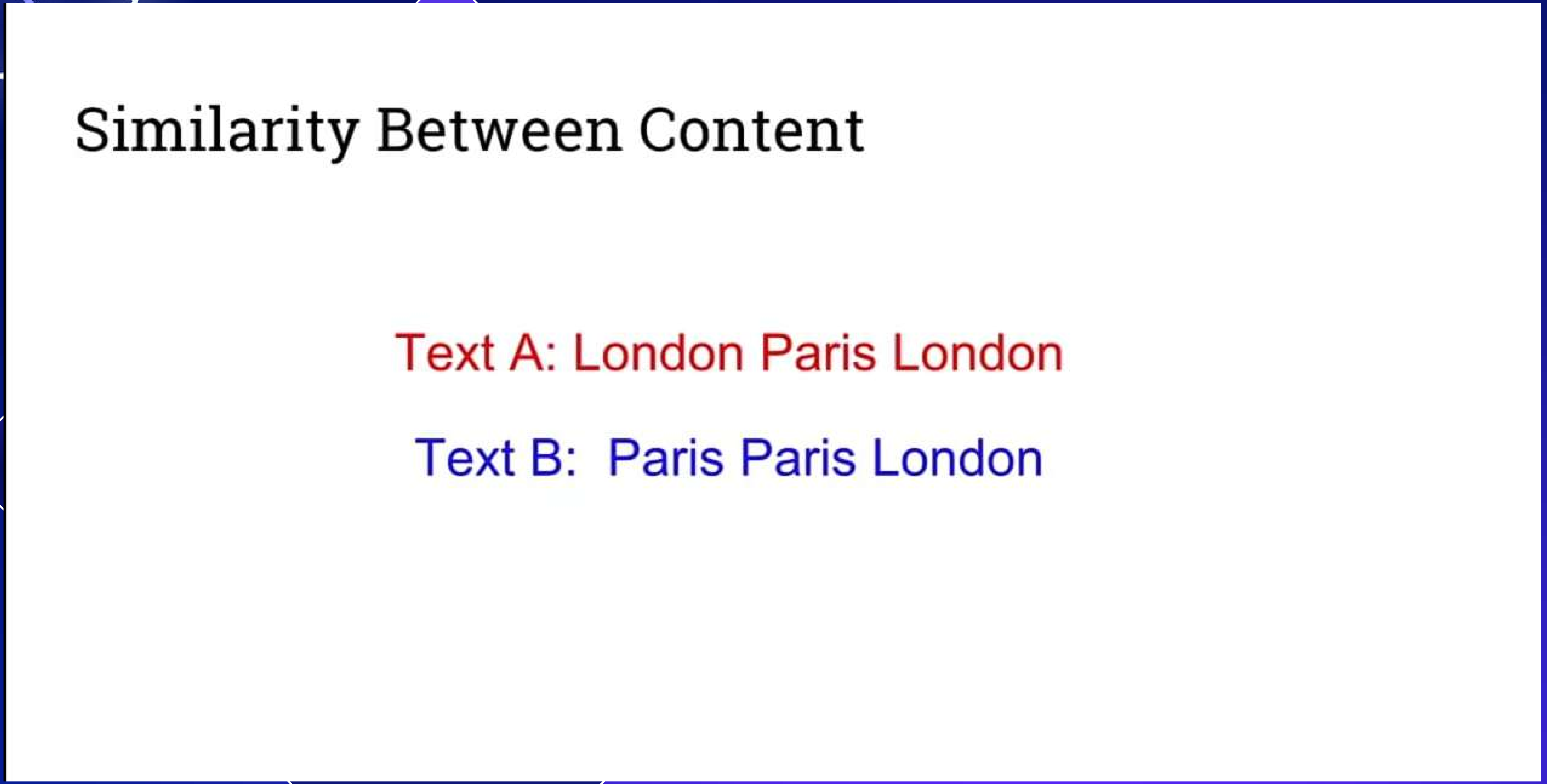


### 3. Collaborative Filtering





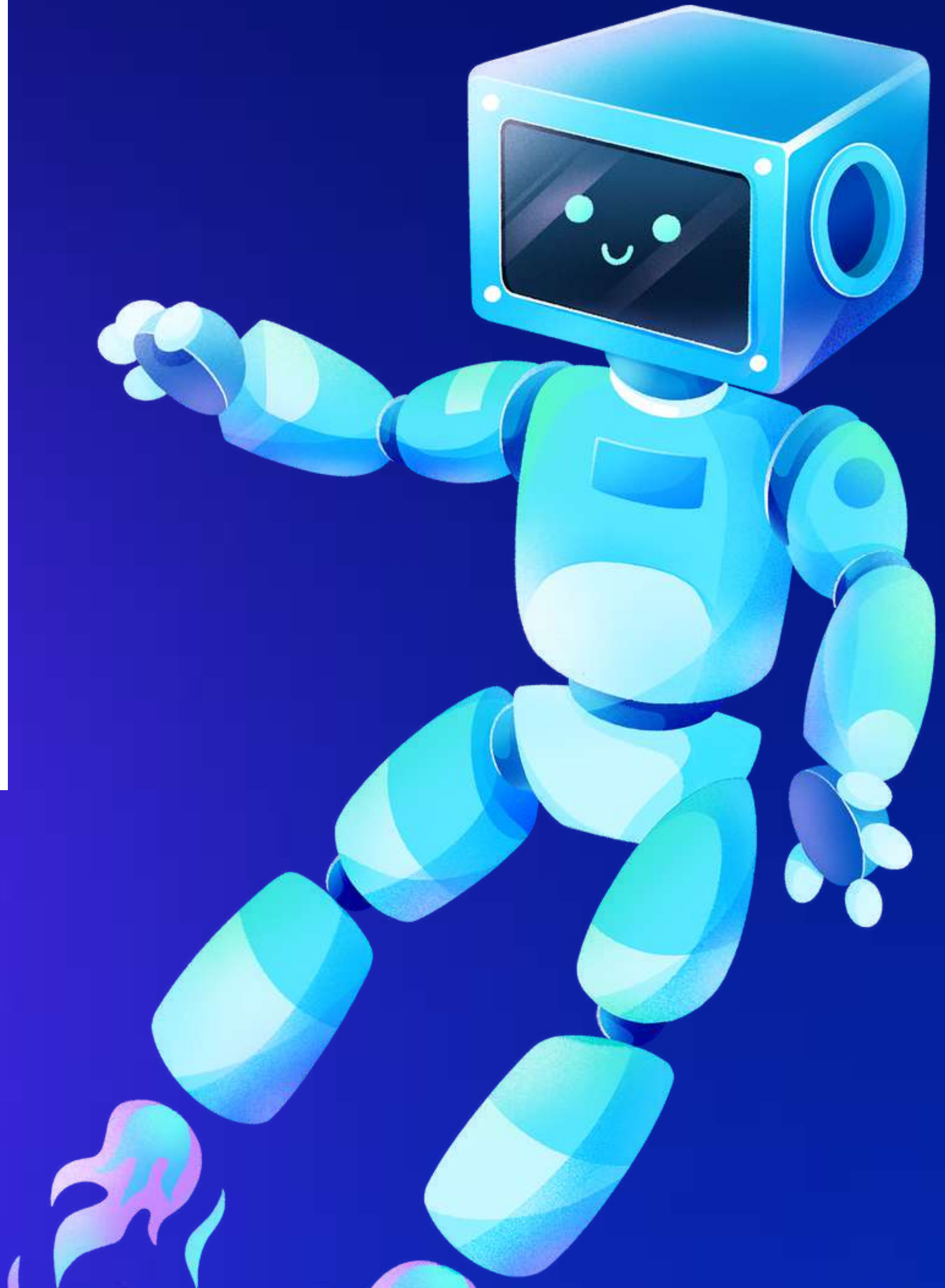
# CORE IDEA

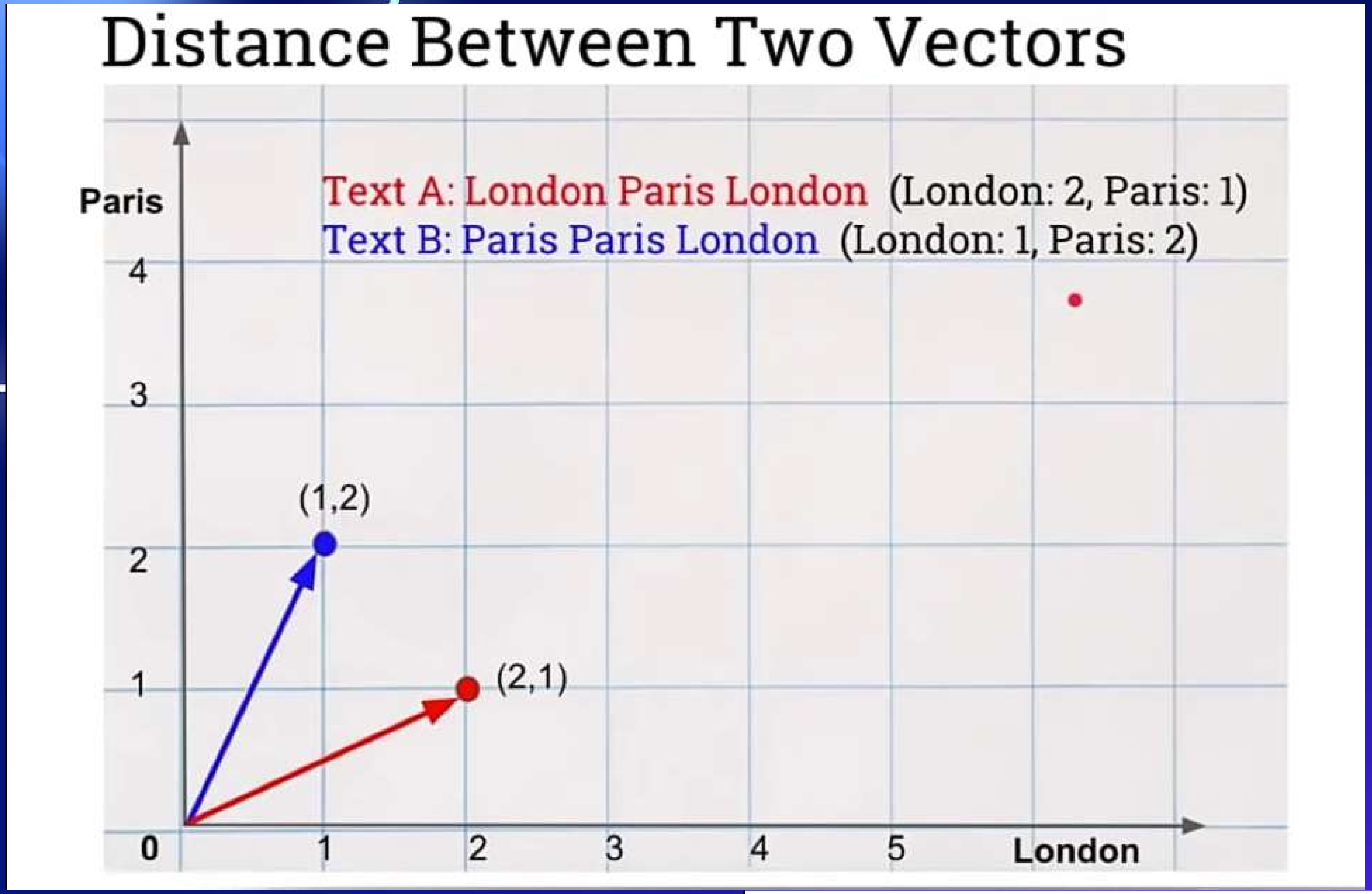


Similarity Between Content

Text A: London Paris London

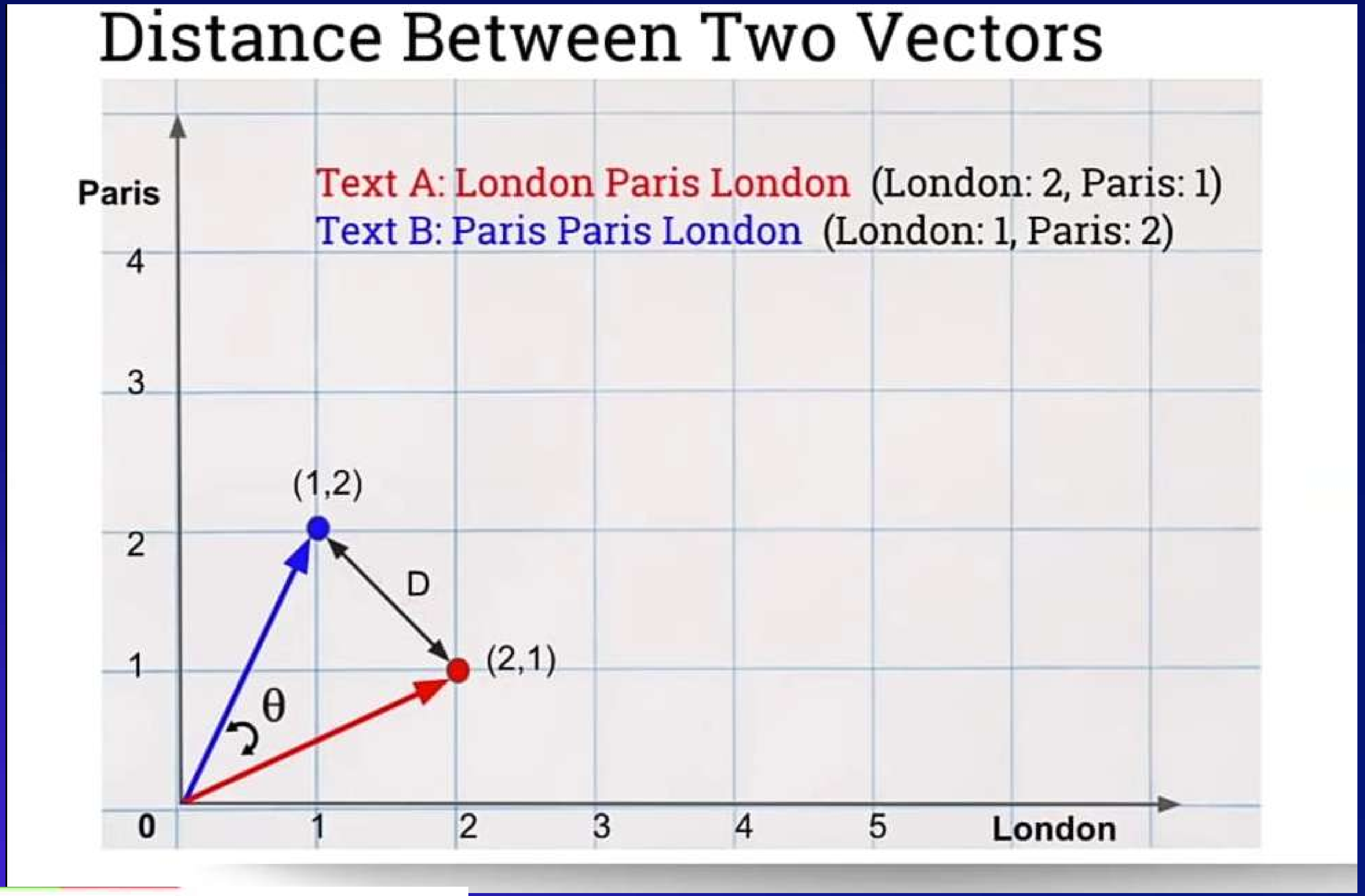
Text B:  Paris Paris London





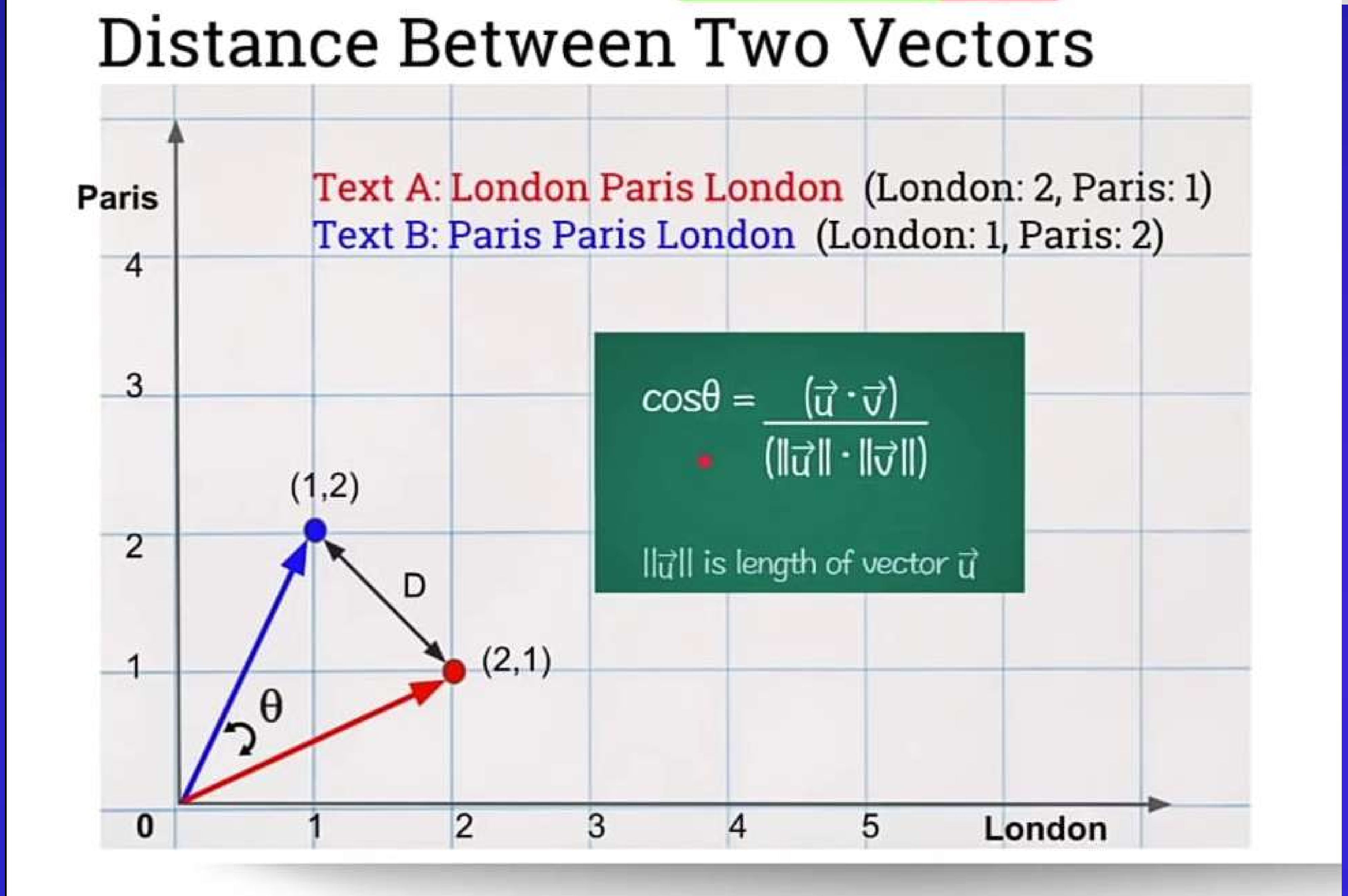
## Distance Between Two Vectors

Text A: London Paris London (London: 2, Paris: 1)
Text B: Paris Paris London (London: 1, Paris: 2)



## Distance Between Two Vectors

Text A: London Paris London (London: 2, Paris: 1)
Text B: Paris Paris London (London: 1, Paris: 2)



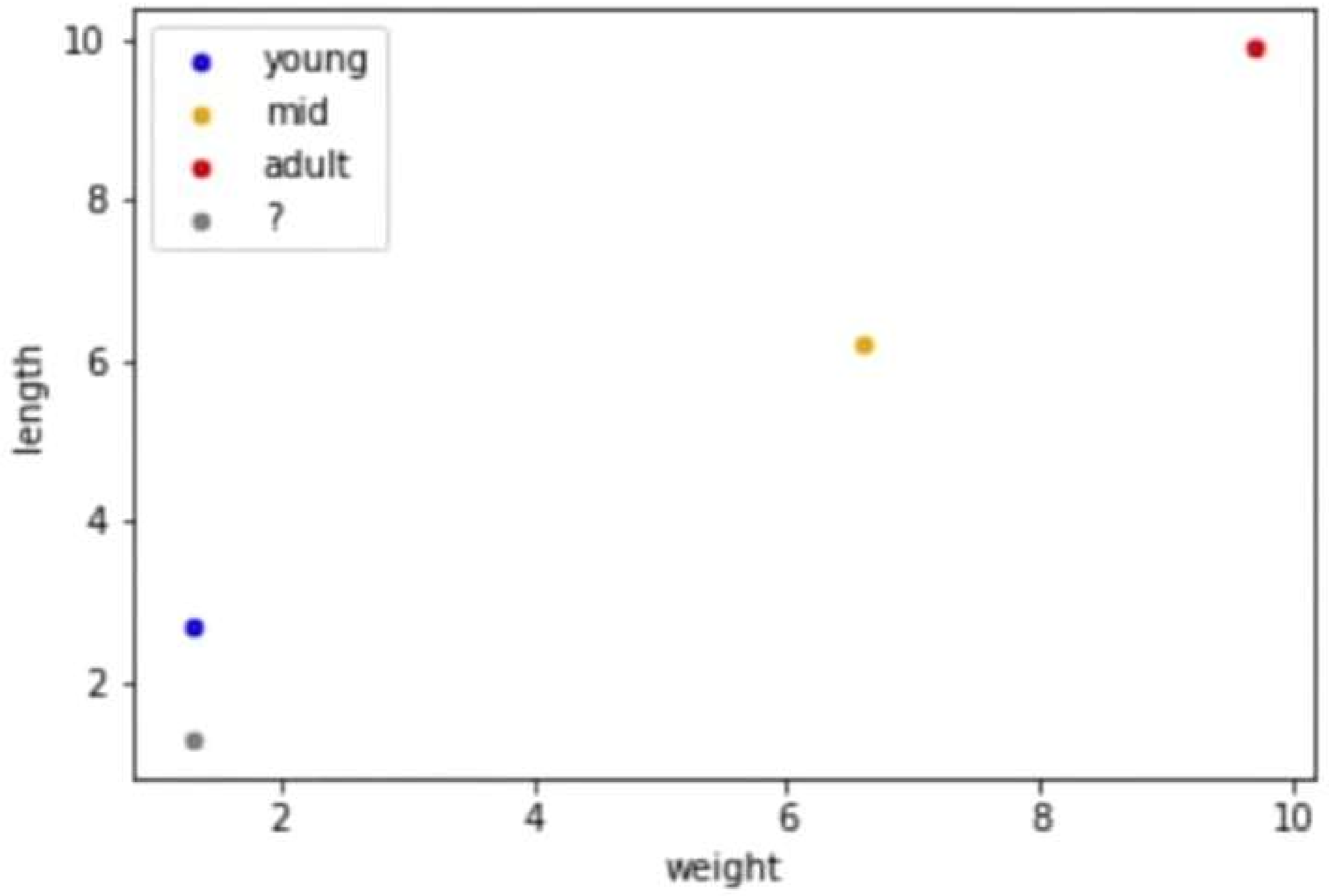
## Distance Between Two Vectors

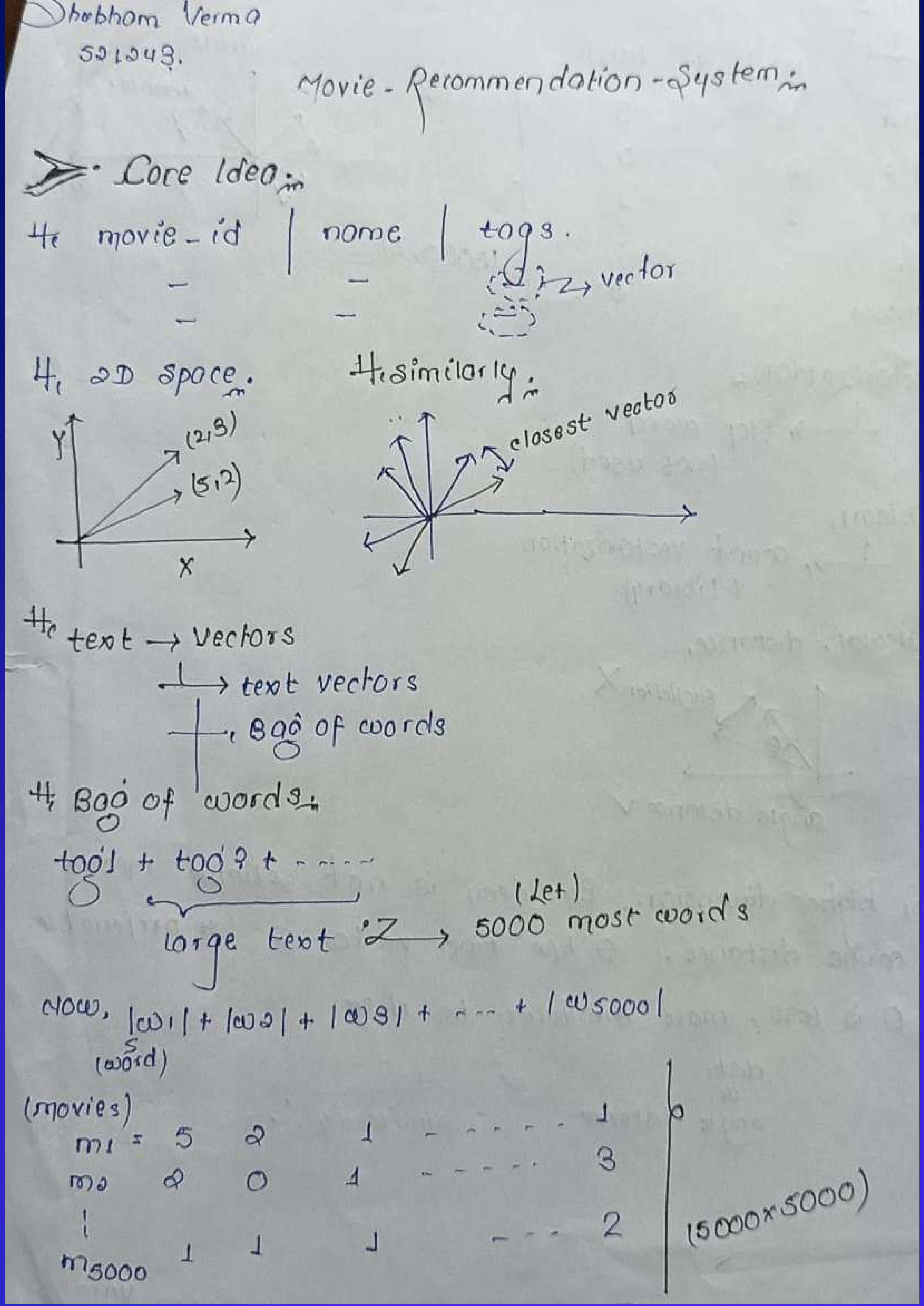
Text A: London Paris London (London: 2, Paris: 1)
Text B: Paris Paris London (London: 1, Paris: 2)

$$\cos\theta = \frac{(\vec{u} \cdot \vec{v})}{(\|\vec{u}\| \cdot \|\vec{v}\|)}$$

$\|\vec{u}\|$ is length of vector $\vec{u}$

## When To Use Angular Distance?





Shubham Verma
521243.

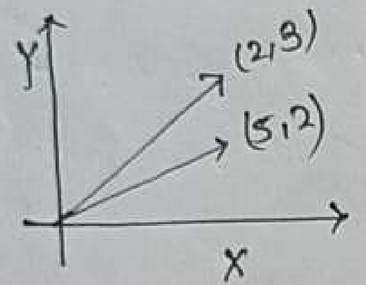
Movie-Recommendation-System:

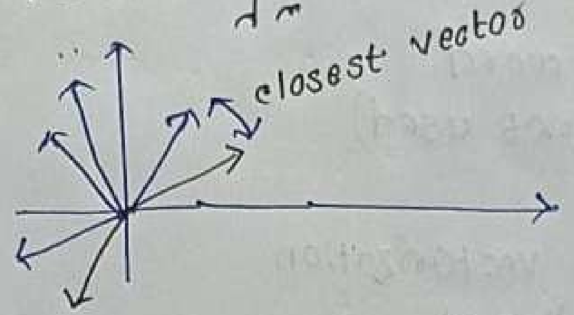
➤ Core Idea:

# movie-id | name | tags.

– | – | $\rightarrow$ vector
– | – |

# 2D space.   # similarity:





# text $\rightarrow$ vectors

$\rightarrow$ text vectors

$\rightarrow$ Bag of words

# Bag of words:

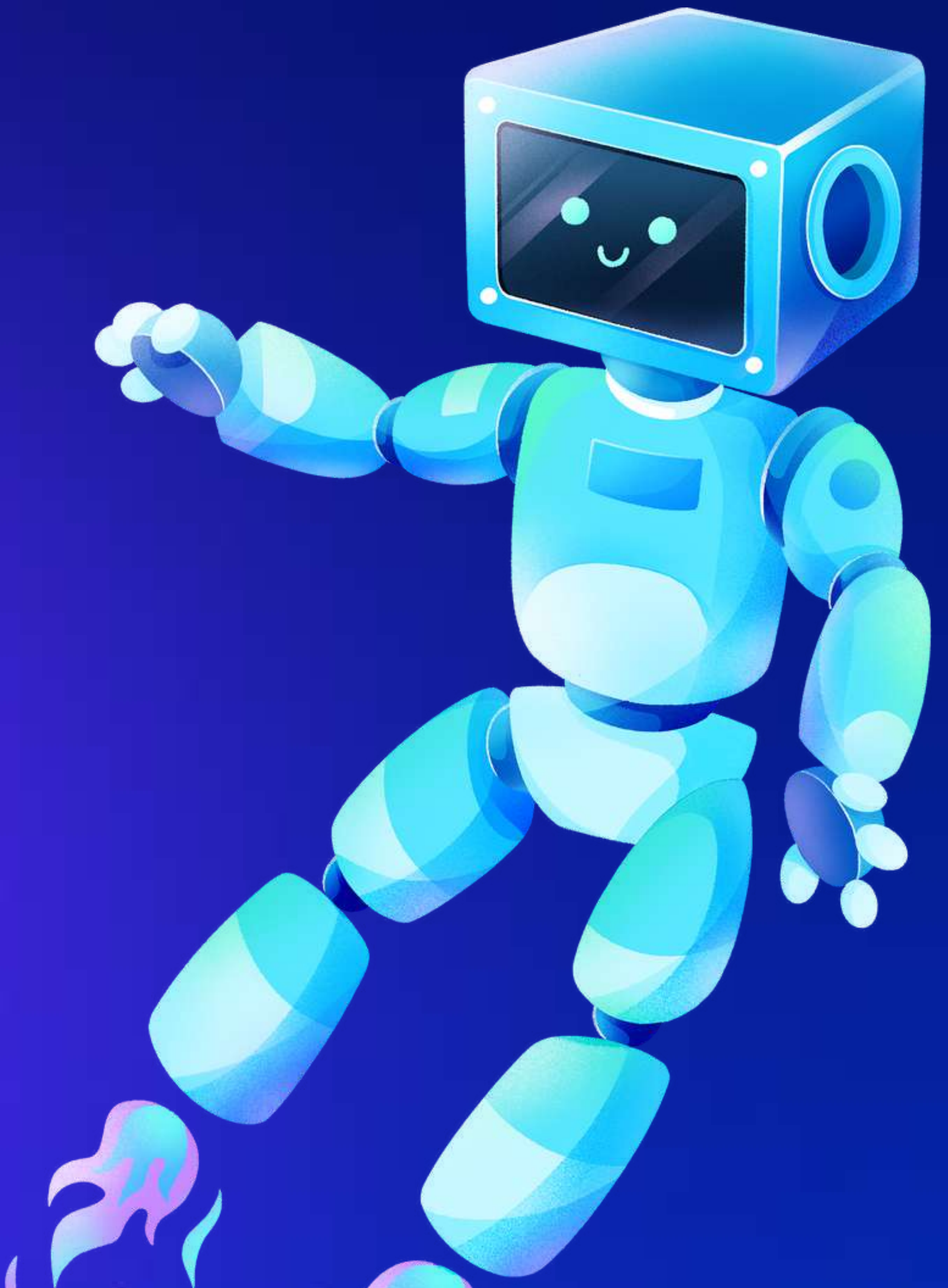
$\underbrace{\text{tag1} + \text{tag2} + \cdots}_{\text{large text}} \Rightarrow$ (Let) 5000 most words

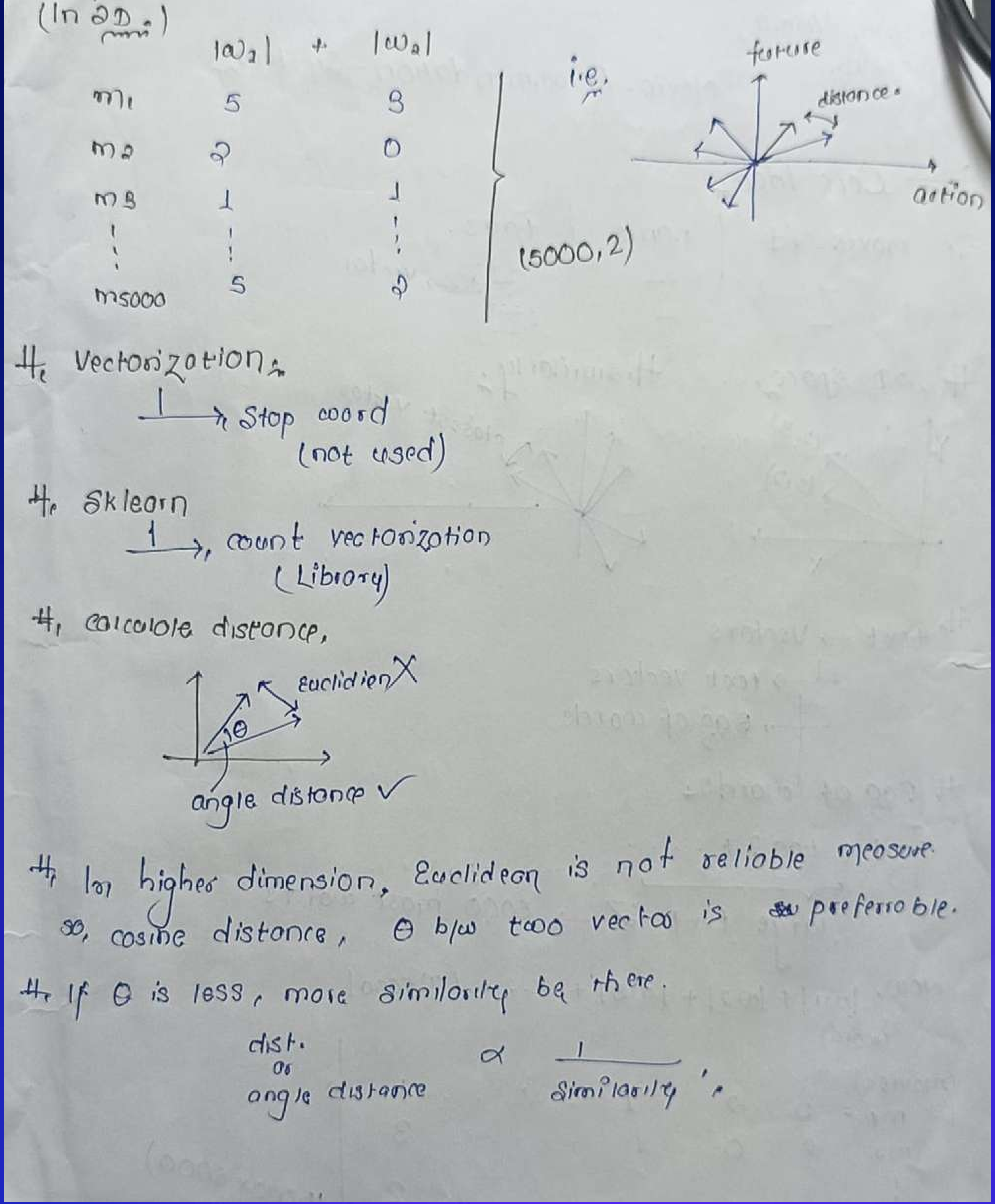
Now, $|w_1| + |w_2| + |w_3| + \cdots + |w_{5000}|$

(words)

$$\begin{array}{l|cccccc|}
\text{(movies)} & & & & & & \\
m_1 = & 5 & 2 & 1 & \cdots & 1 & 0 \\
m_2 & 2 & 0 & 1 & \cdots & 3 & \\
\vdots & & & & & & \\
m_{5000} & 1 & 1 & 1 & \cdots & 2 &
\end{array}$$

(5000×5000)



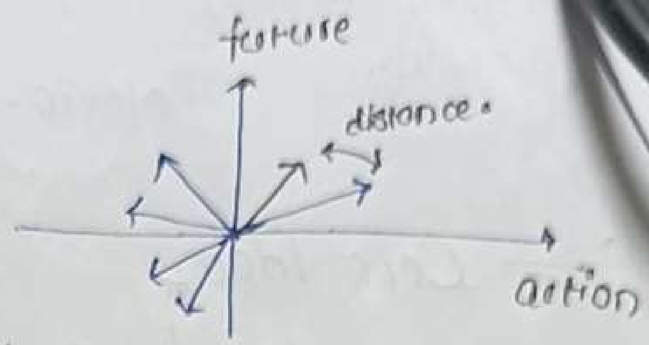


(In 2D)

| | $\lvert w_1 \rvert$ | $\lvert w_2 \rvert$ |
|---|---|---|
| m1 | 5 | 3 |
| m2 | 2 | 0 |
| m3 | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| m5000 | 5 | 2 |

(5000, 2)

i.e.



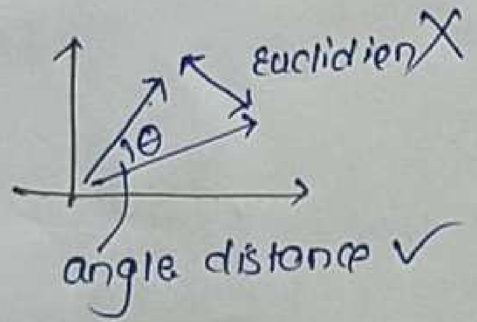
# Vectorization:

$\xrightarrow{1}$ stop word (not used)

# Sklearn

$\xrightarrow{1}$ count vectorization (Library)

# calculate distance.

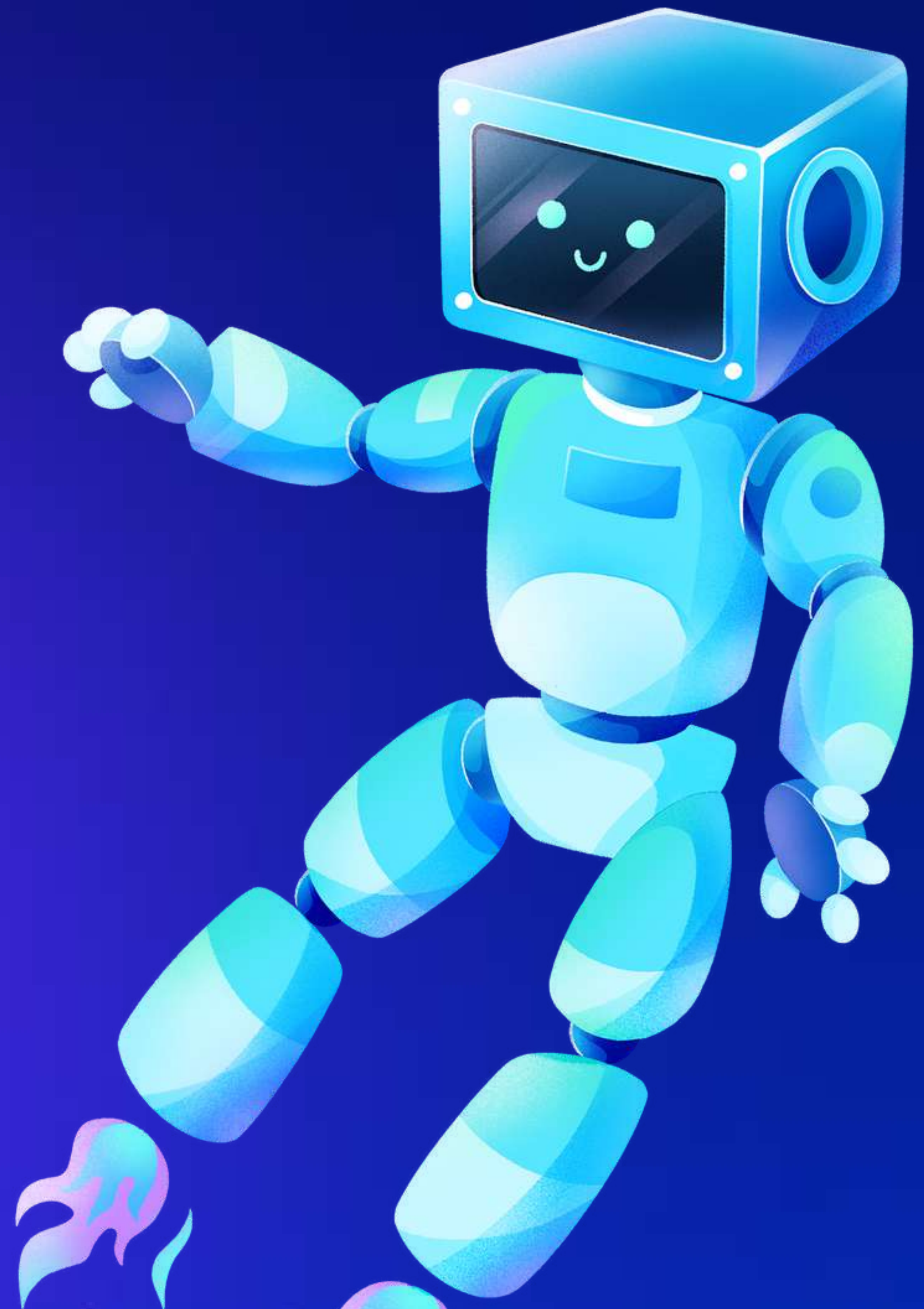


# In higher dimension, Euclidean is not reliable measure.

so, cosine distance, θ b/w two vector is preferrable.

# If θ is less, more similarity be there.

$$\text{dist. or angle distance} \propto \frac{1}{\text{similarity}}$$



# NUMPY

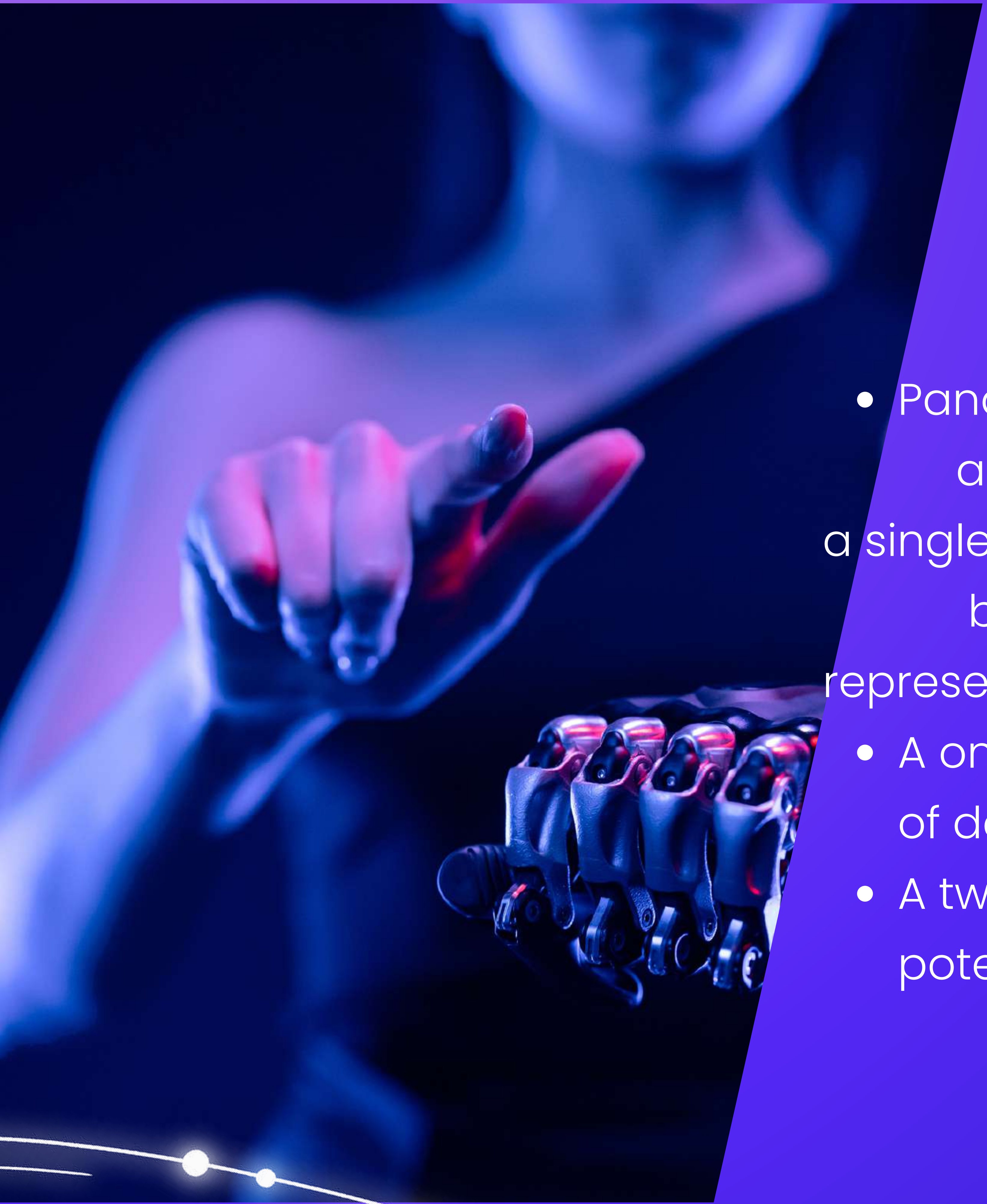
## Why Use NumPy ?

- NumPy provides efficient storage
- It also provides better ways of handling data for processing
- it is fast
- it is easy to learn
- NumPy uses relatively less memory to store data

# PANDAS

## What is PANDAS ?

- Pandas is an open source data analysis library written in Python
- It leverages the power and speed of numPy to make data analysis and preprocessing
- It provides rich and highly robust data operations

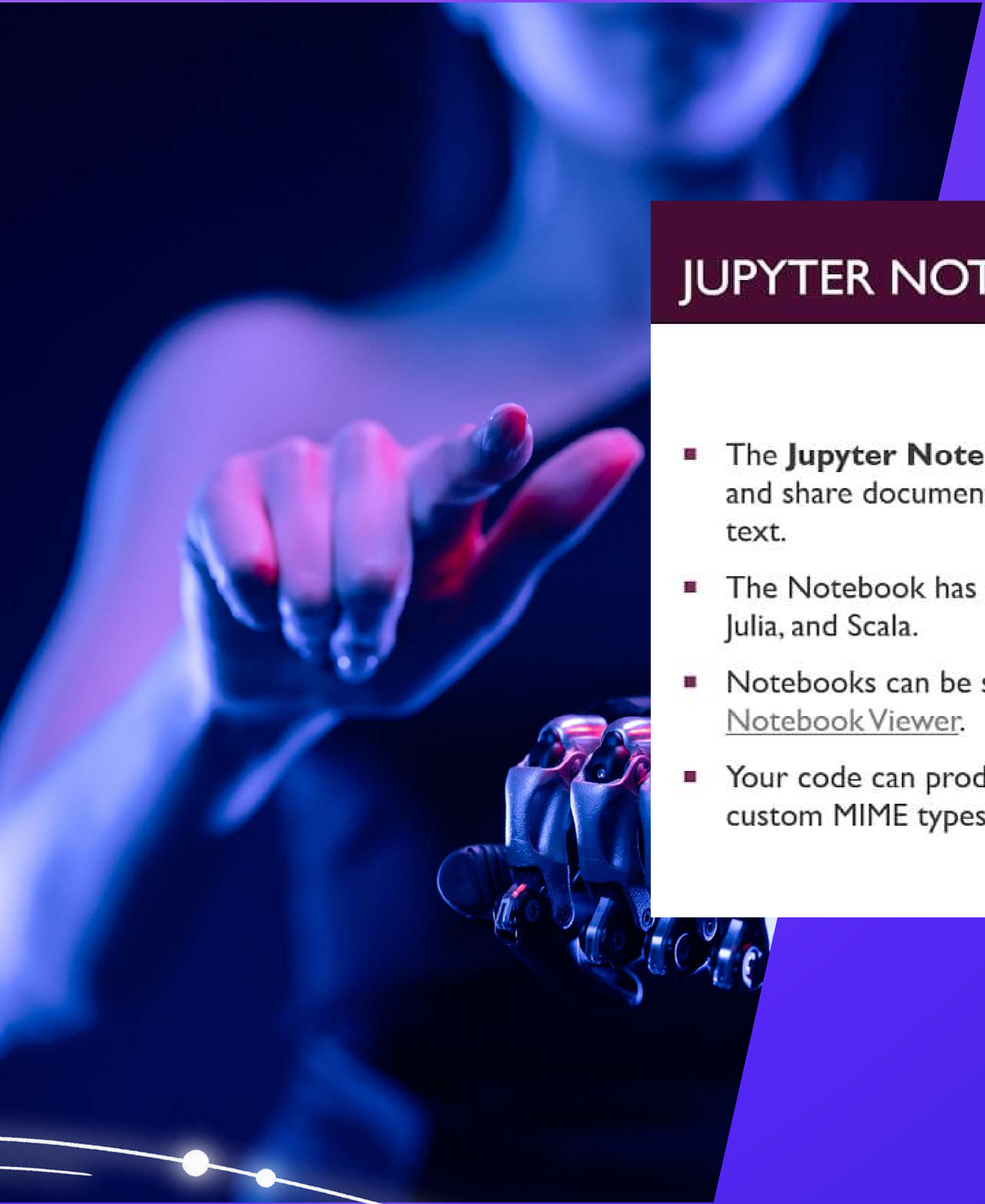


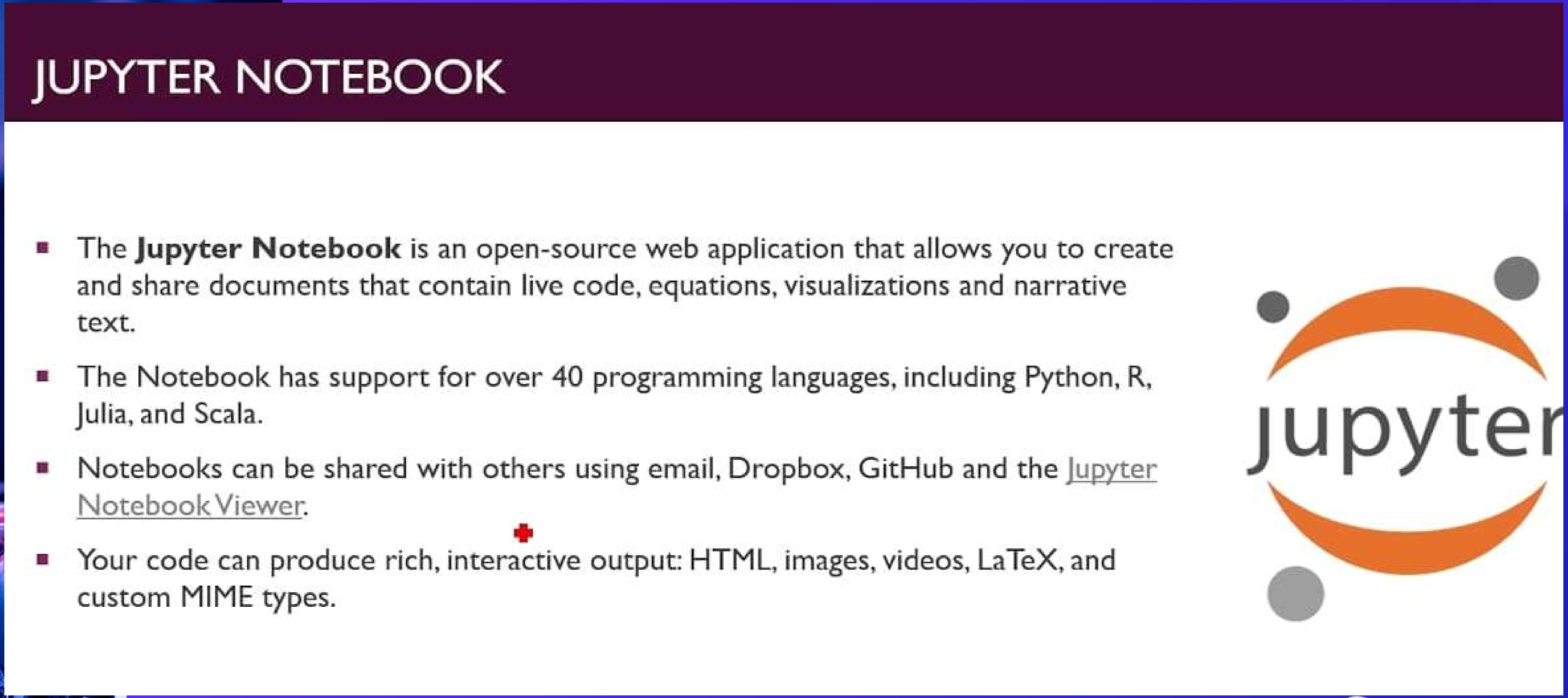
# PANDAS DATA STRUCTURE

- Pandas has two types of data structure:

  a)> Series - It's a one dimensional array with indexes, it stores a single column or row of data in a DataFrame

  b)> DataFrame - It's a tabular spreadsheet like structure representing rows each of which contains one or multiple columns
- A one-dimenasional array(labeled) capable of holding any type of data-Series
- A two-dimensional data(labeled) structure with columns of potentially different typesd of data-DataFrame





## JUPYTER NOTEBOOK

- The **Jupyter Notebook** is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- The Notebook has support for over 40 programming languages, including Python, R, Julia, and Scala.
- Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.
- Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

## CODE EXPLANATION

- import numpy as np: NumPy - library for numerical computations with arrays and matrices.
- import pandas as pd: Pandas - library for data manipulation, particularly with DataFrames.

- movies = pd.read_csv('tmdb_5000_movies.csv'): Loads data from a CSV file ('tmdb_5000_movies.csv') into a Pandas DataFrame named 'movies'.
- credits = pd.read_csv('tmdb_5000_credits.csv'): Loads data from a CSV file ('tmdb_5000_credits.csv') into a Pandas DataFrame named 'credits'.

- movies = movies.merge(credits, on='title'): Merges the 'movies' DataFrame with the 'credits' DataFrame based on the 'title' column.

- movies = movies[['movie_id','title','overview','genres','keywords','cast','crew']]: Selects specific columns ('movie_id', 'title', 'overview', 'genres', 'keywords', 'cast', 'crew') from the 'movies' DataFrame.

- movies.isnull().sum(): Counts the number of missing values (NaN) in each column of the 'movies' DataFrame.

- movies.dropna(inplace=True): Removes rows containing any missing values from the 'movies' DataFrame.

- movies.iloc[0].genres: Accesses the 'genres' data for the first row in the 'movies' DataFrame using integer location indexing.
- def convert(obj): Converts a string representation of a list of dictionaries into a list of names extracted from those dictionaries.
- movies['genres'] = movies['genres'].apply(convert): Applies the 'convert' function to each element in the 'genres' column of the 'movies' DataFrame.
- movies['keywords'] = movies['keywords'].apply(convert): Applies the 'convert' function to each element in the 'keywords' column of the 'movies' DataFrame.
- def convert3(obj): Extracts up to three names from a string representation of a list of dictionaries, limiting to the first three elements.
- movies['cast'] = movies['cast'].apply(convert3): Applies the 'convert3' function to each element in the 'cast' column of the 'movies' DataFrame, extracting up to three names from each list of dictionaries.

- def fetch_director(obj): Extracts the name of the director from a string representation of a list of dictionaries containing job roles.

- movies['crew'] = movies['crew'].apply(fetch_director): Applies the 'fetch_director' function to each element in the 'crew' column of the 'movies' DataFrame, extracting the name of the director from each list of dictionaries.

- movies['overview'] = movies['overview'].apply(lambda x: x.split()): Splits each overview text into a list of words and assigns the result to the 'overview' column of the 'movies' DataFrame.

- movies['genres'] = movies['genres'].apply(lambda x: [i.replace(" ","") for i in x]): Removes spaces from each element in the 'genres' list for every row in the 'movies' DataFrame.
- movies['keywords'] = movies['keywords'].apply(lambda x: [i.replace(" ","") for i in x]): Removes spaces from each element in the 'keywords' list for every row in the 'movies' DataFrame.
- movies['cast'] = movies['cast'].apply(lambda x: [i.replace(" ","") for i in x]): Removes spaces from each element in the 'cast' list for every row in the 'movies' DataFrame.
- movies['crew'] = movies['crew'].apply(lambda x: [i.replace(" ","") for i in x]): Removes spaces from each element in the 'crew' list for every row in the 'movies' DataFrame.

- movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']: Concatenates the 'overview', 'genres', 'keywords', 'cast', and 'crew' columns for each row in the 'movies' DataFrame and assigns the result to a new 'tags' column.

- new_df = movies[['movie_id','title','tags']]: Creates a new DataFrame 'new_df' containing only the 'movie_id', 'title', and 'tags' columns from the 'movies' DataFrame.

- new_df['tags'] = new_df['tags'].apply(lambda x: " ".join(x)): Joins the elements in the 'tags' list for each row in the 'new_df' DataFrame into a single string separated by spaces.

- def stem(text): Stems each word in a given text using the Porter stemming algorithm.

- from sklearn.feature_extraction.text import CountVectorizer: Imports the CountVectorizer class from the scikit-learn library for converting text data into a matrix of token counts.
- cv = CountVectorizer(max_features=5000, stop_words='english'): Initializes a CountVectorizer object 'cv' with a maximum of 5000 features and English stop words removed.

- vectors = cv.fit_transform(new_df['tags']).toarray(): Converts the 'tags' column of the 'new_df' DataFrame into a matrix of token counts using the CountVectorizer object 'cv', and then converts it into a dense array representation.

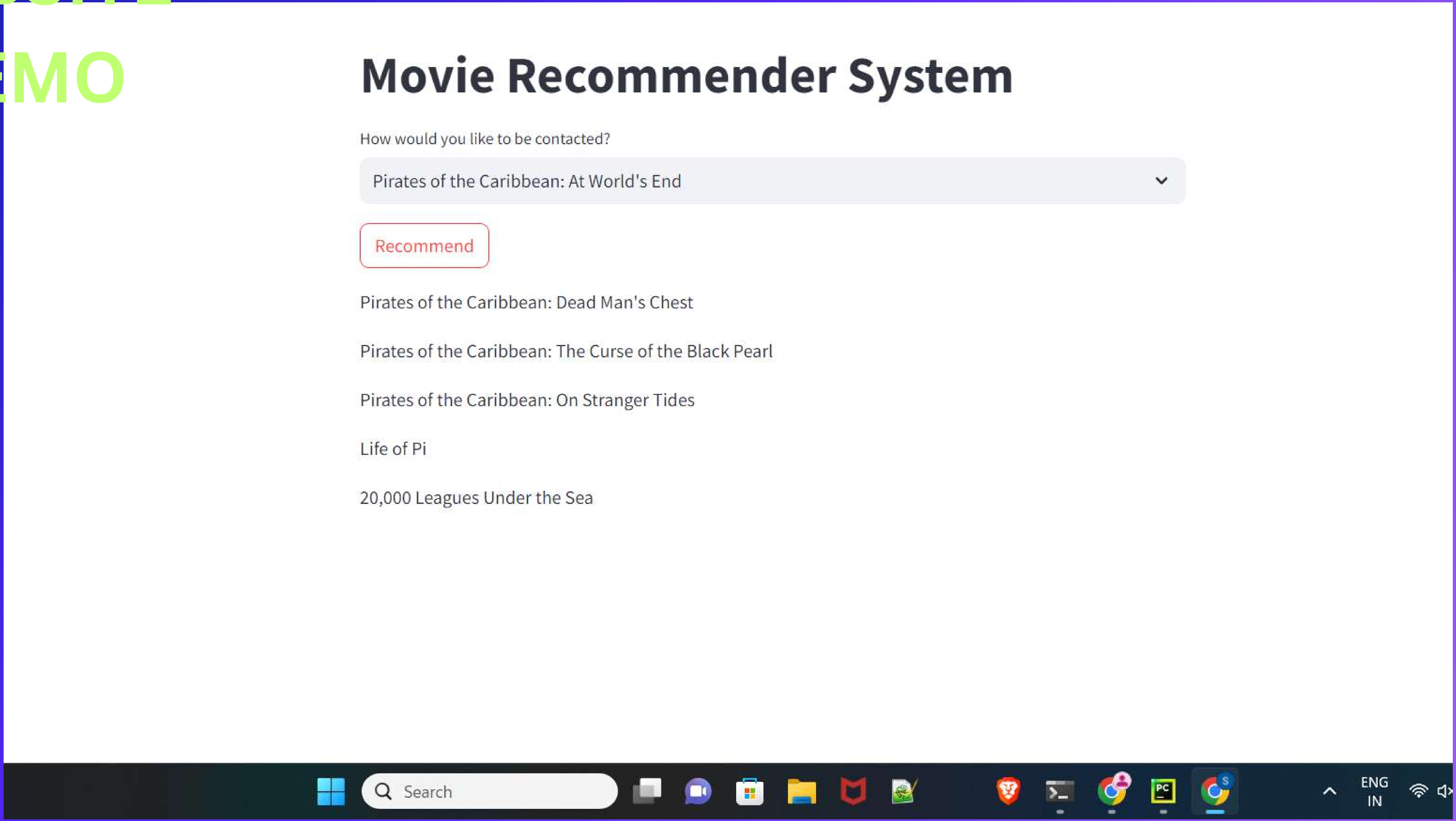
- from sklearn import metrics: Imports the metrics submodule from scikit-learn for evaluating model performance.
- from sklearn.metrics.pairwise import cosine_similarity: Imports the cosine_similarity function from scikit-learn for computing cosine similarity between samples.

- similarity = cosine_similarity(vectors): Computes the cosine similarity between all pairs of samples in the 'vectors' matrix.

- sorted(list(enumerate(similarity[0])), reverse=True, key=lambda x: x[1])[1:6]: Sorts the similarity scores of the first row in descending order, excluding the first element (self-similarity), and retrieves the top 5 most similar indices.

- def recommend(movie): Finds the index of the input movie in the DataFrame, computes the similarity distances for that movie, and prints the titles of the top 5 recommended movies based on similarity scores.

# WEBSITE DEMO



# Movie Recommender System

How would you like to be contacted?

Pirates of the Caribbean: At World's End

Recommend

Pirates of the Caribbean: Dead Man's Chest

Pirates of the Caribbean: The Curse of the Black Pearl

Pirates of the Caribbean: On Stranger Tides

Life of Pi

20,000 Leagues Under the Sea

## GITHUB LINK

- Dataset

#tmdb_5000_credits.csv
https://drive.google.com/file/d/1jXr68cbwM-S9iM_IdgOWHH7e4lyJMIuF/view?usp=sharing

#tmdb_5000_movies.csv
https://drive.google.com/file/d/1ZqoccAOaxX8WWrdtisTGY68EPFaGGgE0/view?usp=sharing

- JUPYTER

#code https://github.com/omprakash1207/sms-spam-detection/blob/main/movies_recomendation.ipynb

cgoogle_colab

#code https://colab.research.google.com/drive/1X2CmteVFkEb7H6XOEfEU_tvpodm2edB1?usp=sharing

# THANK YOU!

# RESOURCE PAGE