BioBytes Summer Project (BioSoc)

MID-TERM EVAL

Presented By : Group 3

Tanmay

Ishant

Asmita

Karan

Mentors:

Nischay Patel

Sikha Vamsi

OVERVIEW

01 Concepts Covered Till Now

02 Naive Bayes Classifier

03 Detailed Explanation of Imputations

04 Future Plan of the Project

CONCEPTS COVERED TILL NOW

Git/GitHub

Used to track changes made in our repo and also to push our code onto GitHub

Numpy

A library in Python. It provides support for multidimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

Pandas

A library in Python. It provides data structures like dataframes and functions needed to work with structured data seamlessly.

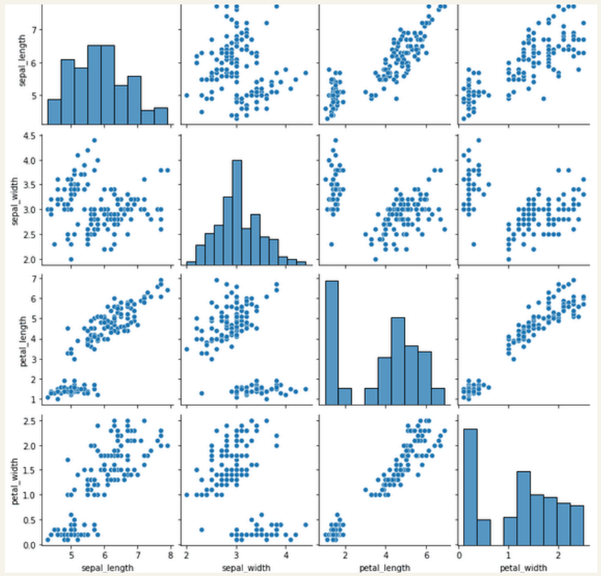
CONCEPTS COVERED TILL NOW

Matplotlib

A library in Python. It is used for data visualization and generating a wide range of plots and charts.

Seaborn

A library in Python. It is built on top of Matplotlib and focuses on making statistical plots more informative and aesthetically pleasing.



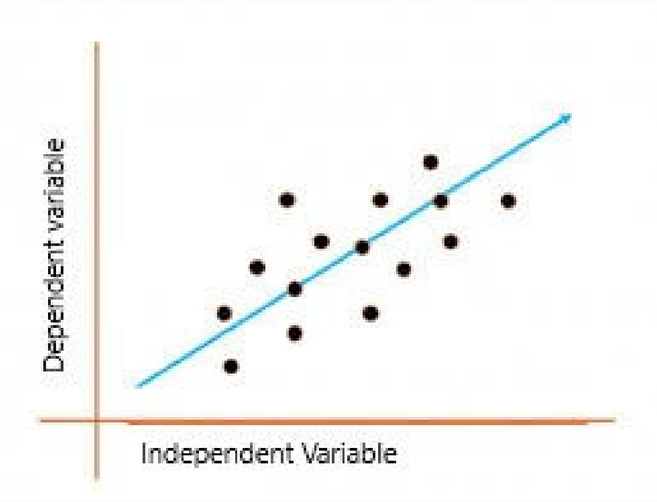
CONCEPTS COVERED TILL NOW

Linear Regression

- Linear Regression is a machine learning model which predicts the relationship between two variables by assuming they have a straight-line connection.
- It finds the best line that minimizes the differences between predicted and actual values.

Simple Linear Regression

If there is a single input variable X (independent variable), such linear regression is simple linear regression.



The blue line is referred to as the best-fit straight line. Based on the given data points, we attempt to plot a line that fits the points the best.

CONCEPTS COVERED TILL NOW

Linear Regression

Linear Regression uses the traditional slope-intercept form of a straight line to find the best fit line.

$$y = \mathbf{m}x + \mathbf{c}$$

The goal of the algorithm is to find the best values for **m** and **c** such that the error between the predicted values and actual values is minimized.

Cost Function

The cost function helps to work out the optimal values for **m** and **c** , which provides the best-fit line for the data points.

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

Using the MSE function, we'll update the values of **m** and **c** such that the MSE value settles at the minima.

CONCEPTS COVERED TILL NOW

Linear Regression

Gradient Descent

A regression model optimizes the gradient descent algorithm to update **m** and **c** by reducing the cost function by randomly selecting values and then iteratively updating them to reach the minimum cost function.

Evaluation Metrics

- R-Squared (R2) :

$$R2 = 1 - (RSS/TSS)$$

RSS - Sum of squares of Residuals
TSS - Sum of errors from the mean

- Root Mean Squared Error (RMSE) :

$$\sqrt{\sum_{i=1}^{n}\left(y_i^{Actual} - y_i^{Predicted}\right)^2 / n}$$

CONCEPTS COVERED TILL NOW

Multiple Linear Regression

If there is are multiple input variables (independent variables), such linear regression is simple linear regression.

$$y = c + m1x1 + m2x2 + m3x3 \ldots$$

Overfitting

When a model learns every pattern and noise in the data to such an extent that it affects the performance of the model on the unseen future dataset, it is referred to as overfitting. When a model has low bias and higher variance it ends up memorizing the data and causing overfitting.

CONCEPTS COVERED TILL NOW

Underfitting

When the model fails to learn from the training dataset and is also not able to generalize the test dataset, is referred to as underfitting. When a model has high bias and low variance it ends up not generalizing the data and causing underfitting. It is unable to find the hidden underlying patterns in the data.

Decision Tree

A decision tree is like a series of questions that leads to a final decision. By asking and answering the right questions, it helps predict outcomes based on the data you have.

- **Classification Trees**: Used when the outcome is a category
- **Regression Trees**: Used when the outcome is a number

CONCEPTS COVERED TILL NOW

KNN

k-Nearest Neighbors is a straightforward and effective algorithm for making predictions based on the closest data points in the training set.

- For classification, it takes a majority vote of the classes of the “k” nearest neighbors.
- For regression, it computes the average of the target values of the “k” nearest neighbors.

Random Forest

A machine learning algorithm designed for classification tasks.

It uses a collection of decision trees to make predictions.

Each tree is trained on a different subset of the training data and makes predictions independently.

CONCEPTS COVERED TILL NOW

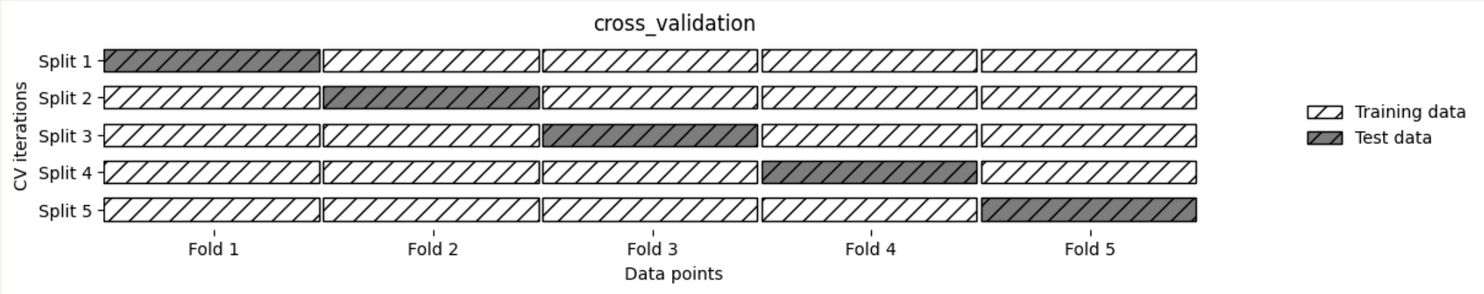
Gradient Boosting

Like Random Forest in that it uses a collection of decision trees to make predictions.

New trees are added to the model sequentially, each on correcting the errors of the combined existing trees.

Cross Validation

Splits the data into parts and uses each part as train and test data one by one to get better accuracy.



CONCEPTS COVERED TILL NOW

Logistic Regression

A regression algorithm which does classification.

Calculates probability of belonging to a particular class.

Sigmoid Function

The sigmoid function is a mathematical function that maps any real-valued number to a value between 0 and 1.

$$\sigma = \frac{1}{(1+e^{-x})}$$

Cost Function

$$cost = -\frac{1}{m}\sum_{i=1}^{m}[y * log(a) + (1-y) * log(1-a)]$$

Logistic Regression

Model

```python
def sigmoid(x):
    return 1/(1 + np.exp(-x))

def model(X, Y, learning_rate, iterations):

    m = X_train.shape[1]
    n = X_train.shape[0]

    W = np.zeros((n,1))
    B = 0

    cost_list = []

    for i in range(iterations):

        Z = np.dot(W.T, X) + B
        A = sigmoid(Z)

        # cost function
        cost = -(1/m)*np.sum( Y*np.log(A) + (1-Y)*np.log(1-A))

        # Gradient Descent
        dW = (1/m)*np.dot(A-Y, X.T)
        dB = (1/m)*np.sum(A - Y)

        W = W - learning_rate*dW.T
        B = B - learning_rate*dB

        # Keeping track of our cost function value
        cost_list.append(cost)

        if(i%(iterations/10) == 0):
            print("cost after ", i, "iteration is : ", cost)

    return W, B, cost_list
```

W - Weights Matrix

B - Bias

Z - Linear combination of weights, features, and bias

A - Probability matrix by applying sigmoid to Z

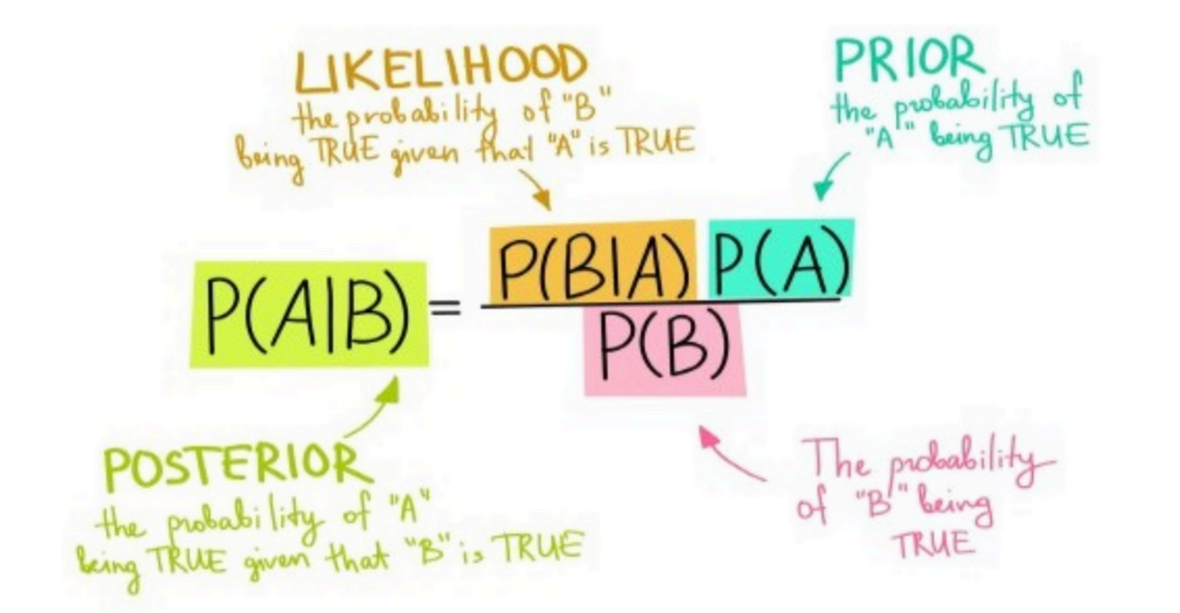
dW and dB - Gradients of cost wrt W and B

These gradients guide the updates to the weights and bias to minimize the cost function, optimizing the probability predictions.

The model returns W (the learned coefficients for each feature) and B (the learned offset term). These are then used to predict the probabilities for new data.

The probabilities can then be used to classify the data by setting a threshold value.

NAIVE BAYES CLASSIFIER



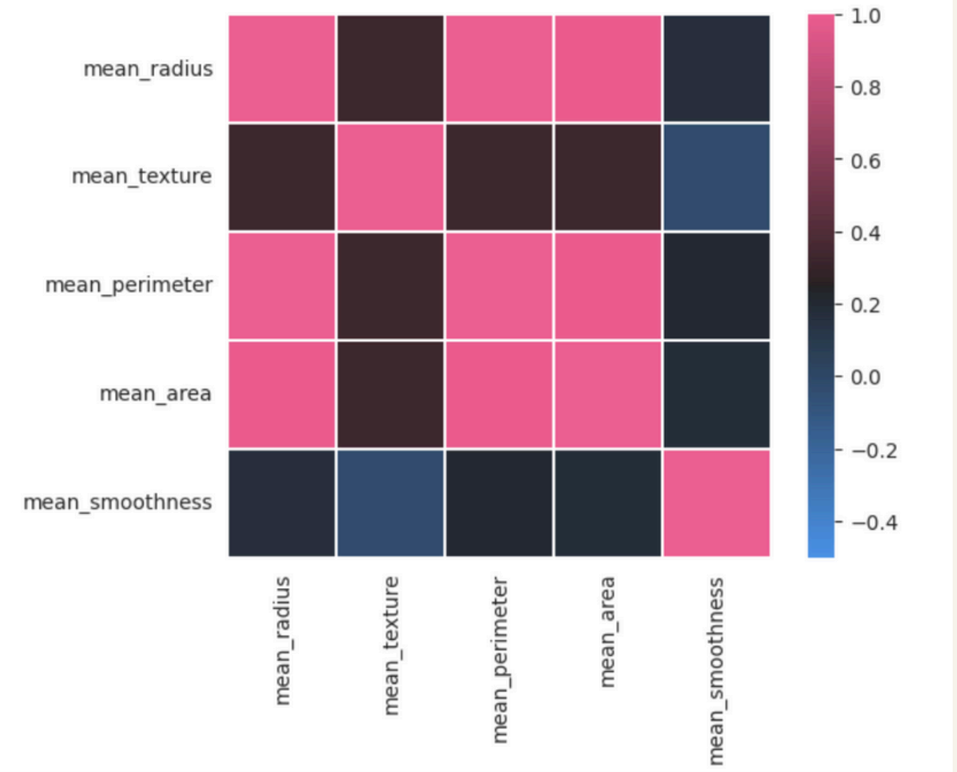
INTRODUCTION

Naive Bayes is a simple probabilistic classifier based on Bayes' theorem with strong (naive) independence assumptions between features.

Example: Cancer Dataset

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
5	12.45	15.70	82.57	477.1	0.12780	0
6	18.25	19.98	119.60	1040.0	0.09463	0
7	13.71	20.83	90.20	577.9	0.11890	0
8	13.00	21.82	87.50	519.8	0.12730	0
9	12.46	24.04	83.97	475.9	0.11860	0

```
corr = data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250,354,80,60,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
```



Correlation Matrix to understand how the features are related

Clearly mean_radius, mean_perimeter and mean_area are closely related(quite intutitive also)

So we will drop mean_area and mean_perimeter from out dataset to create the training data

Main Goal :

If we are given a feature vector x1,x2,x3 we have to predict the class in the target variable(which is here 0 or 1)

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)}$$

Posterior_probability

We will calculate the posterior probablity of each class in the target variable(here 0 or 1) and the class which has the maximum posterior_probablity is our prediction

Why do we need to make the naive assumption :

Consider a senerio where we have the feature vector (X=x1,x2,x3) in our test data but there is no such row in our training dataset then the posterior probability will always be zero if the features are dependent no matter which class the target variable(Y) belong to, but this is a major setback to our model.

But if the features are independent then we can write:

$$P(X|y) = P(x_1, x_2, \ldots, x_n|y)$$

$$= P(x_1|x_2, \ldots, x_n, y) * P(x_2|x_3, \ldots, x_n, y) \ldots P(x_n|y)$$

How to calculate the likelihood probability for each class[P(X/Y)] :

Many times it may happen that we have continuous features, so we cannot directly calculate the likelihood probability [P(X/Y)]. This problem can be eliminated by following two ways:

Fitting the data in a known distribution like Gaussian Distribution

This way we can calculate the likelihood using the probability density function of that distribution

Probability density function for Gaussian distribution →

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Converting continuous feature into categorical ones

Say we have a marks column in our training dataset which is continuous, then we can convert it into categorical by making labels like marks from 0-20 == 0, 20-40 == 1 and so on

Python Implementation from scratch

Calculate P(Y=y) for all possible y

```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

Approach 1: Calculate P(X=x|Y=y) using Gaussian dist.

```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

Final Naive Bayes Algorithm

Calculate P(X=x1|Y=y)P(X=x2|Y=y)...P(X=xn|Y=y) * P(Y=y) for all y and find the maximum

```python
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df, features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

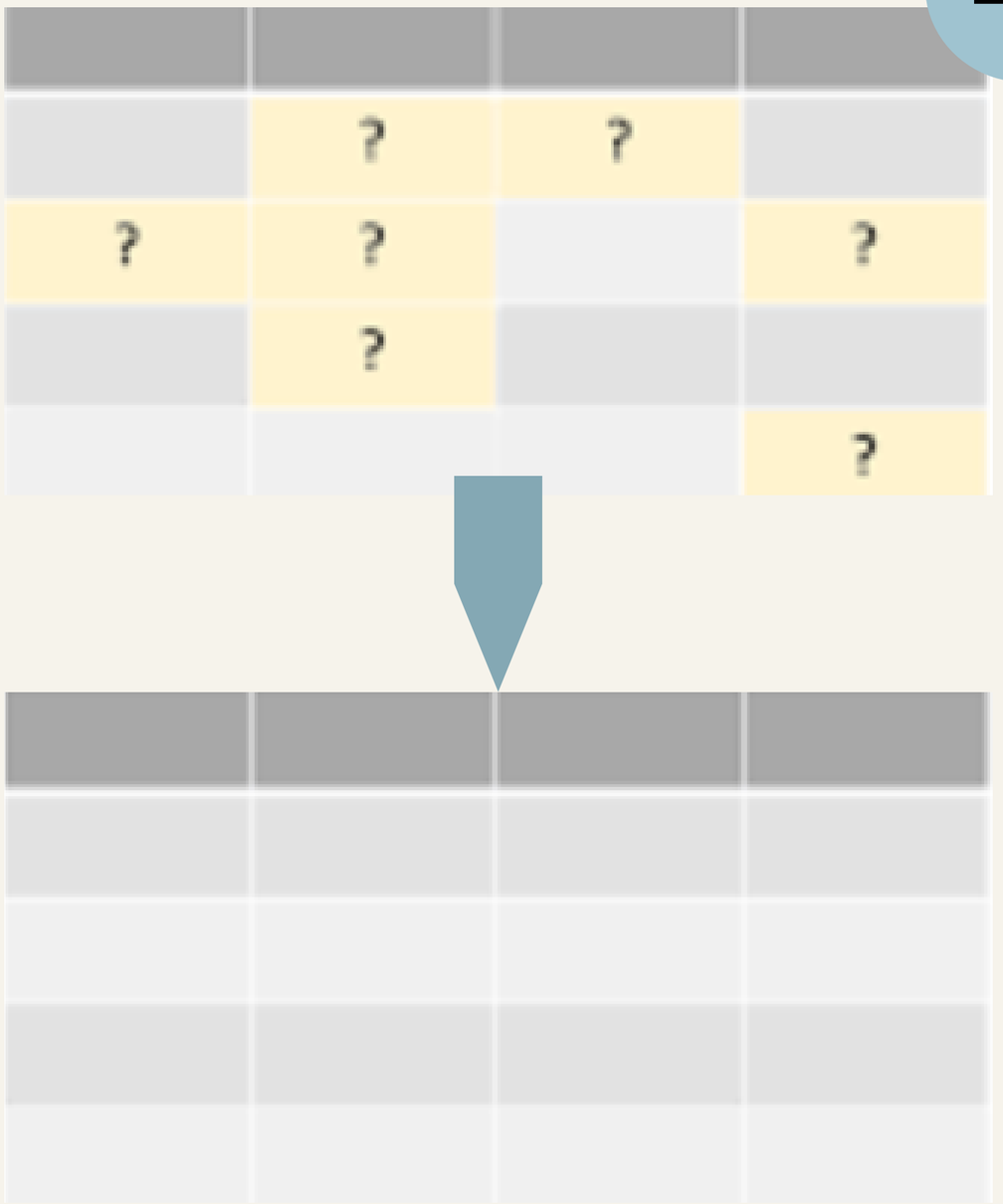
MISSING VALUE IMPUTATION

INTRODUCTION

Imputation refers to the process of replacing missing data with substituted values. When data is missing in a dataset, imputation techniques are used to fill in those gaps so that the dataset can be used effectively for analysis or machine learning models.

Why Data Might be Missing:

1. Human Error: Mistakes in data entry or recording.
2. Data Corruption: Loss of data due to software or hardware issues.
3. Non-responses: Participants skipping questions in surveys or studies.
4. Data Collection Issues: Incomplete data collection processes



Why is it Important to Handle Missing Data:

- Improves Model Performance: Complete data leads to more accurate and robust models.
- Preserves Data Utilization: Keeps the dataset size intact, retaining valuable information.
- Reduces Bias: Minimizes bias introduced by missing values.
- Enhances Predictive Power: Ensures models work effectively with real-world unseen data.
- Maintains Data Integrity: Preserves relationships and patterns in the data.

Different Ways of Handling Missing Values:

- Dropping Columns with Missing Values
- A better option-Imputation
- An Extention to Imputation

Dropping Columns with Missing Values:

The simplest option(though not very efficient) is to drop columns with missing values.

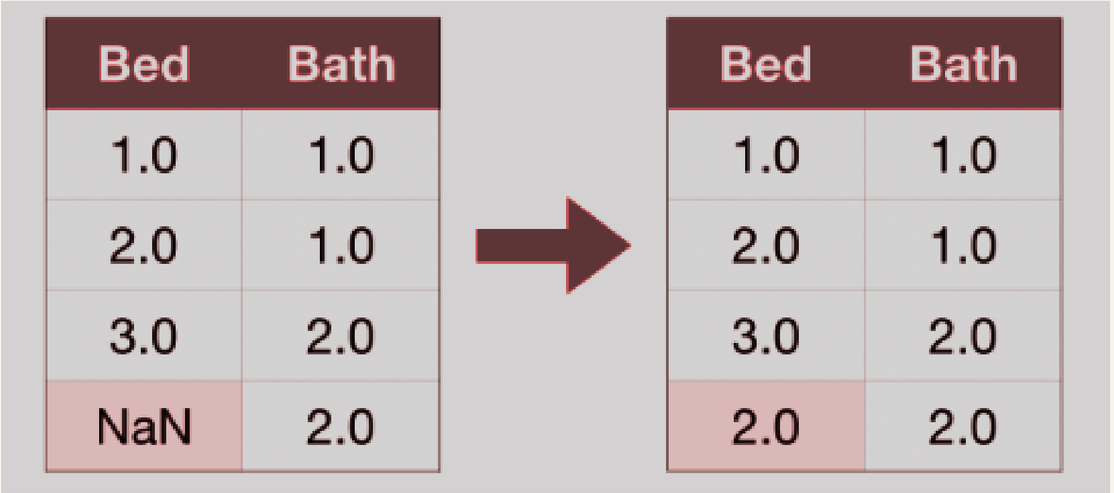
Unless most values in the dropped columns are missing, the model loses access to a lot of (potentially useful!) information with this approach.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bath
1.0
1.0
2.0
2.0

Imputation:

Imputation fills in the missing values with some number. For example, we can fill in the mean value along each column.



Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
2.0	2.0

There are various ways of imputing missing numerical values.
Two simple approaches are:

1.) Using fillna() method in Pandas

2.) Using SimpleImputer in Sklearn

Imputation

1.) Using fillna() method in Pandas

The fillna() method in pandas is used to fill missing values in a DataFrame or Series. It allows you to replace NaN (Not a Number) values with a specified value, method, or by using forward or backward fill

1.) Fill with a specific value:

```
df['column'].fillna(value=0, inplace=True)
```

Original

One	Two
0	2
1	3
NaN	0
2	1

data.fillna(0)

One	Two
0	2
1	3
0	0
2	1

2.) Fill with the mean, median, or mode of the column:

```
df['column'].fillna(df['column'].mean(), inplace=True)
```

3.) Using forward and backward fill

Forward fill is used to fill missing values in a DataFrame or Series by carrying forward the last valid observation.

Backward fill, or backward propagation, in pandas is a method used to fill missing values in a DataFrame or Series by carrying backward the next valid observation.

```
df.fillna(method='ffill', inplace=True)
df.fillna(method='bfill', inplace=True)
```

Imputation

2.) Using SimpleImputer in Sklearn:

SimpleImputer is a Scikit-learn class for handling missing data by filling in or imputing missing values with various strategies. It is part of the sklearn.impute module and is used to replace missing values (NaN) in a dataset with specific values such as the mean, median, mode, or a constant.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')
dfstd.marks = imputer.fit_transform(dfstd['marks'].values.reshape(-1,1))[:,0]
dfstd
```

	marks	gender	result
0	85.000000	M	verygood
1	95.000000	F	excellent
2	75.000000	F	None
3	85.833333	M	average
4	70.000000	M	good
5	85.833333	M	None
6	92.000000	F	verygood
7	98.000000	M	excellent

An Extention to Imputation

Imputation is the standard approach, and it usually works well. However, imputed values may be systematically above or below their actual values (which weren't collected in the dataset). Or rows with missing values may be unique in some other way. In that case, your model would make better predictions by considering which values were originally missing in the dataset.

In this approach, we impute the missing values, as before. And, additionally, for each column with missing entries in the original dataset, we add a new column(with a dummy variable) to show if that particular feature was missing or not.

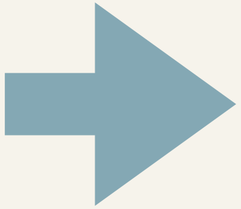
In some cases, this will meaningfully improve results. In other cases, it doesn't help at all.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bed	Bath	Bed_was_missing
1.0	1.0	FALSE
2.0	1.0	FALSE
3.0	2.0	FALSE
2.0	2.0	TRUE

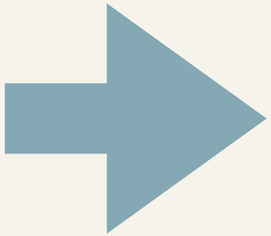
Python Implementation:

Adding new columns to keep track of which datapoints are imputed



```
# Make copy to avoid changing original data (when imputing)
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()

# Make new columns indicating what will be imputed
for col in cols_with_missing:
    X_train_plus[col + '_was_missing'] = X_train_plus[col].isnull()
    X_valid_plus[col + '_was_missing'] = X_valid_plus[col].isnull()
```

Now using SimpleImputer as we have done before



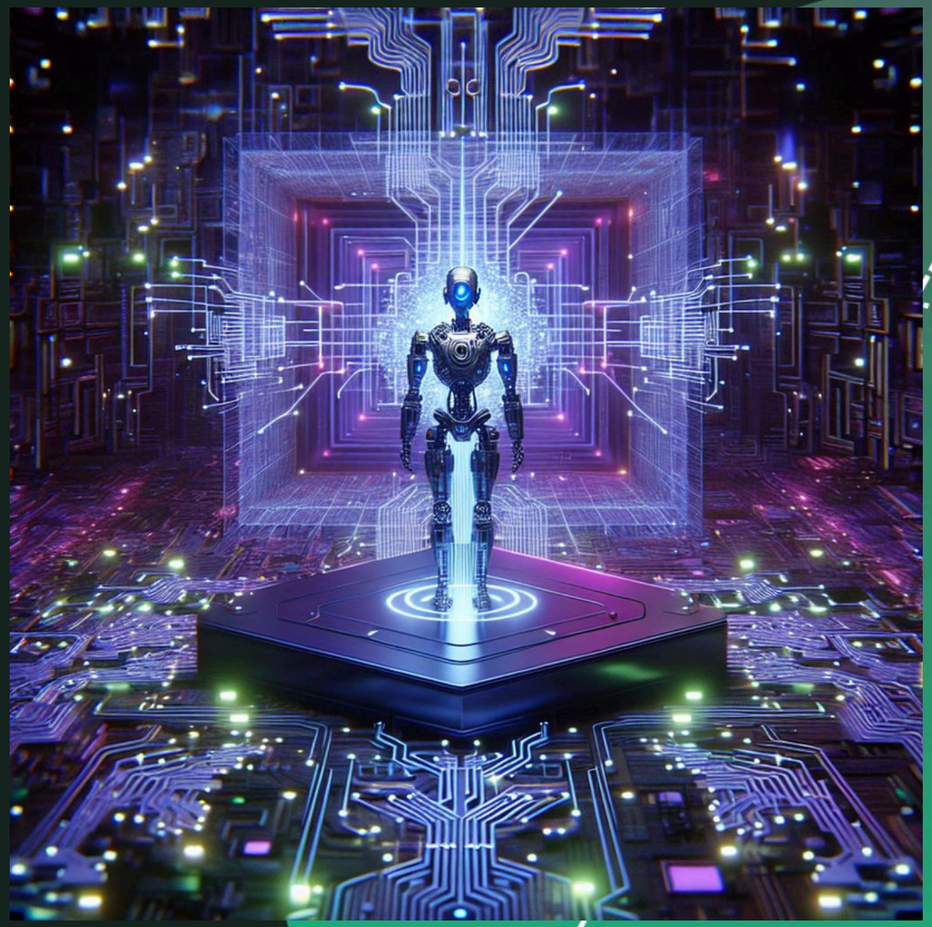
```
t Imputation
ıy_imputer = SimpleImputer()
.mputed_X_train_plus = pd.DataFrame(my_imputer.fit_transform(X_train_plus))
.mputed_X_valid_plus = pd.DataFrame(my_imputer.transform(X_valid_plus))

t Imputation removed column names; put them back
.mputed_X_train_plus.columns = X_train_plus.columns
.mputed_X_valid_plus.columns = X_valid_plus.columns
```

Future Plans

Group 3
Biobytes BioSoc

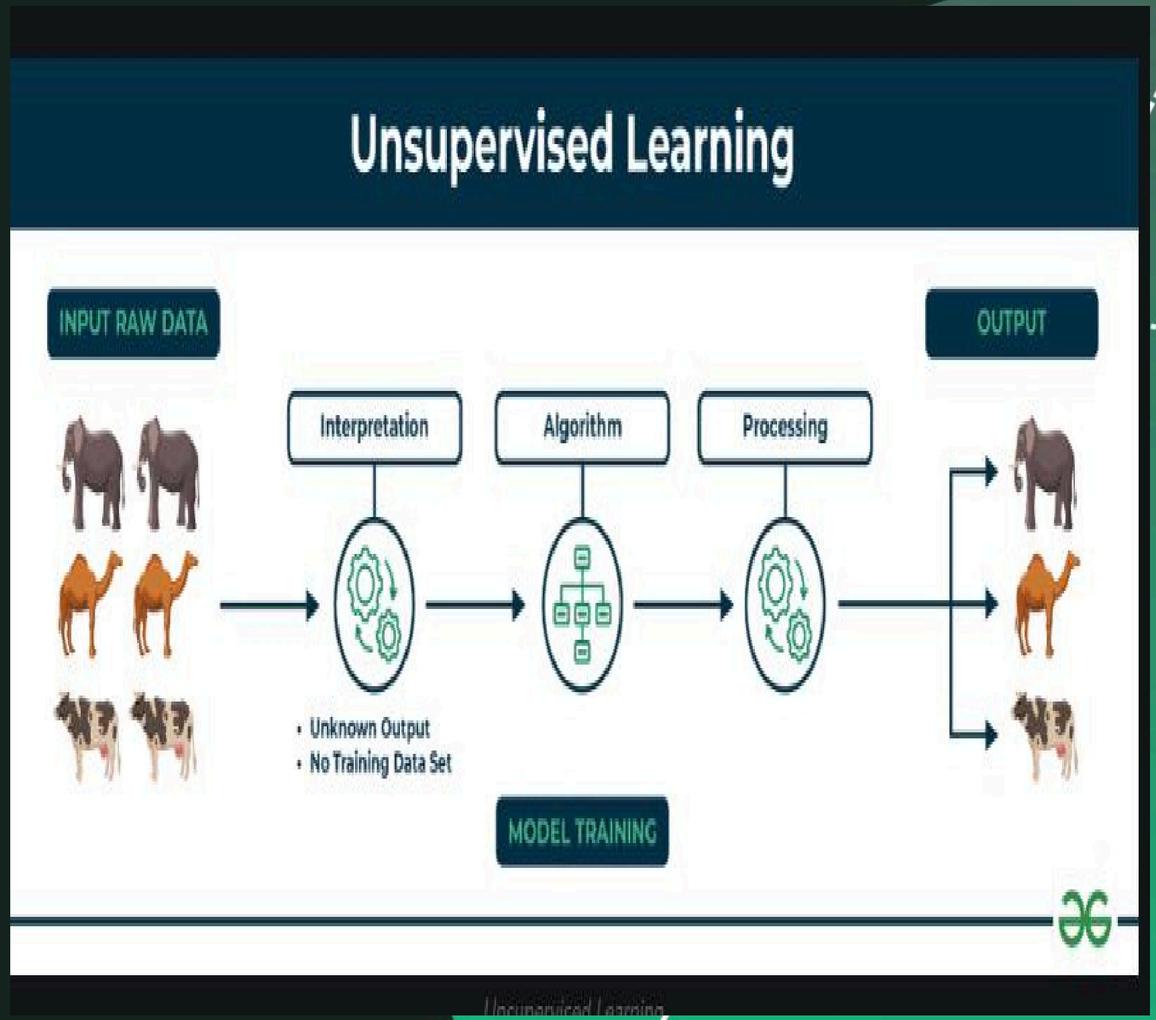
Unspervised Machine Learning

Unsupervised learning is a type of machine learning that learns from unlabeled data. This means that the data does not have any pre-existing labels or categories. The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.



Unspervised Machine Learning

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.

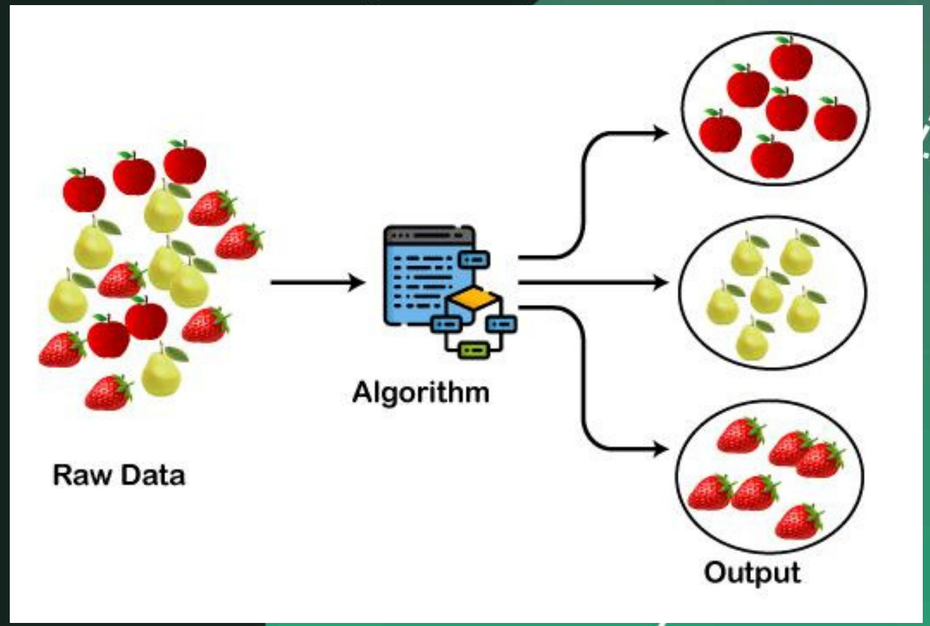


TYPES

1.Clustering
2.Association
3.Dimensionality reduction

Clustering

"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."
It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

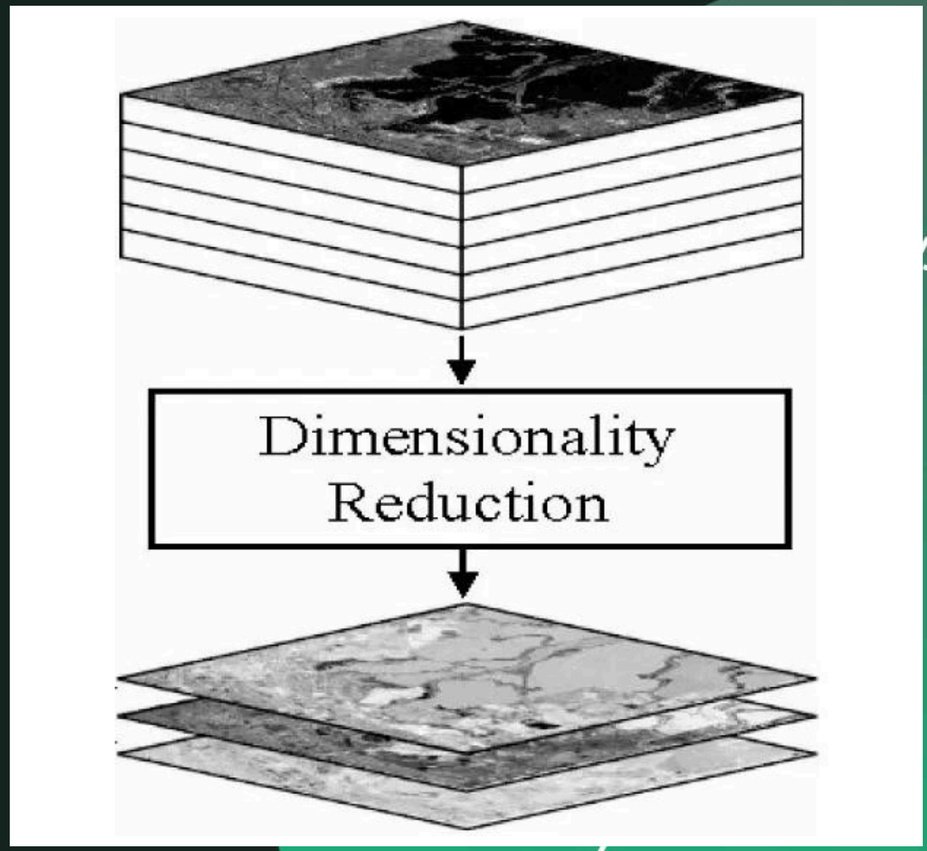


Association

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

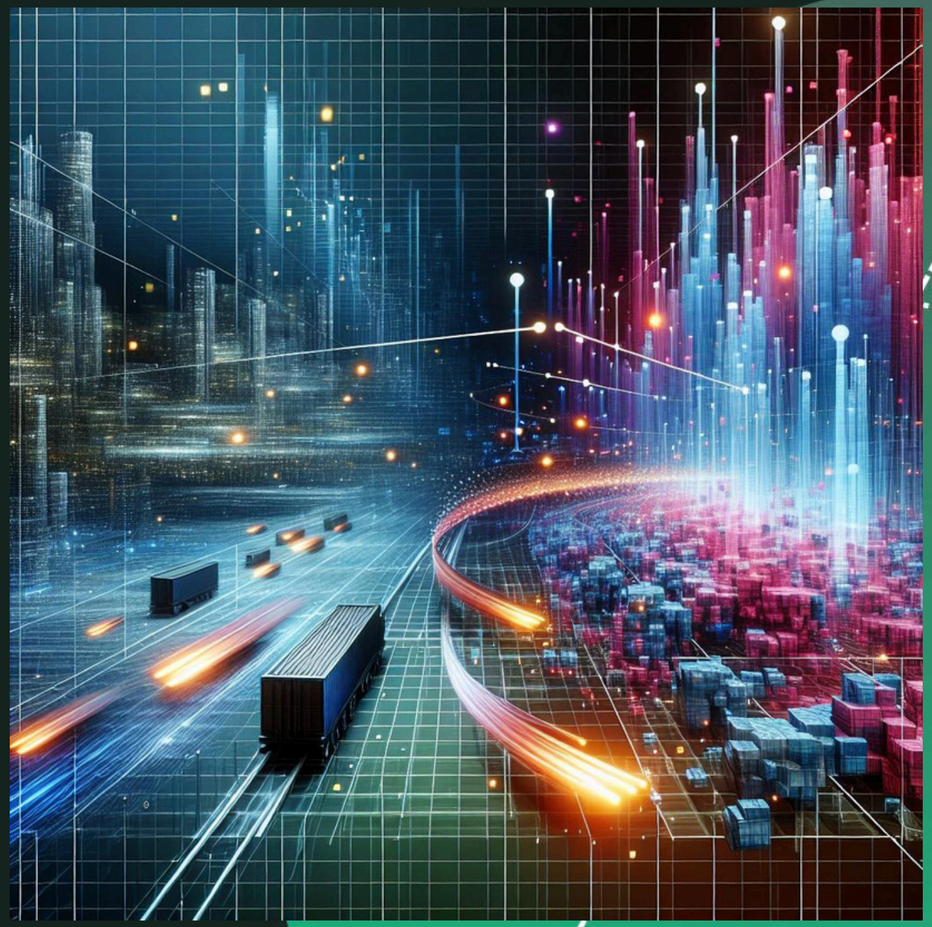
Dimensionality reduction

Dimensionality reduction is a technique used to reduce the number of features in a dataset while retaining as much of the important information as possible. In other words, it is a process of transforming high-dimensional data into a lower-dimensional space that still preserves the essence of the original data.



Importance of Dimensionality reduction?

1.It reduces the time and storage space required.
2.It helps Remove multi-collinearity which improves the interpretation of the parameters of the machine learning model.
3.It becomes easier to visualize the data when reduced to very low dimensions such as 2D or 3D.
4.It removes irrelevant features from the data, Because having irrelevant features in the data can decrease the accuracy of the models and make your model learn based on irrelevant features.



BIOLOGICAL APPLICATION

1.DNA seqeuncing

Machine learning algorithms can be used to analyze large sets of genomic sequencing data. Supervised learning methods for gene identification requires the input of labeled DNA sequences which specify the start and end locations of the gene.The algorithm then uses this model to learn the general properties of genes such as DNA-sequencing patterns and the location of stop codons

DRUG DISCOVERY

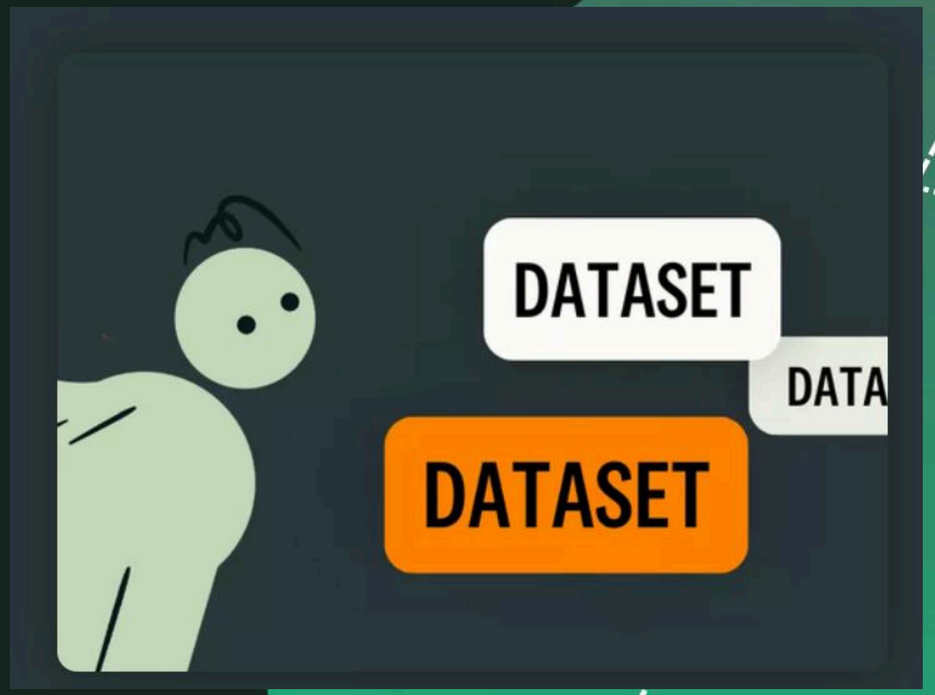
ML algorithms can be used to virtually screen large chemical libraries to identify potential drug candidates, reducing the number of compounds that need to be synthesized and tested in the lab.

Cardiovascular Risk Detection

In particular, ML has been proposed for cardiac imaging applications such as automated computation of scores, differentiation of prognostic phenotypes, quantification of heart function and segmentation of the heart.

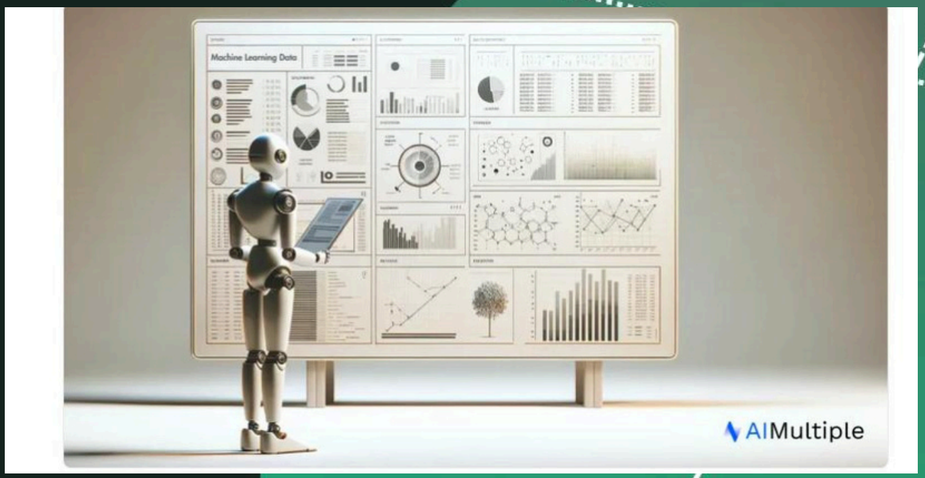
CREATE DATASETS

Machine learning (ML) can be used to create datasets from online biological databases:
1. Data Extraction
Collecting Data:
Web Scraping: ML tools can automatically gather data from web pages.
APIs: ML can help automate the download of data using provided APIs.
2. Data Cleaning
Fixing Errors: ML algorithms can identify and correct errors or inconsistencies in the data.



CREATE DATASETS

3. Data Integration
Combining Data: ML can merge data from different sources, ensuring it fits together properly.
5. Data Analysis
Finding Patterns: ML algorithms can analyze the data to find meaningful patterns or insights.
6. Data Storage
Organizing Data: ML can help structure and store data efficiently for easy access and further analysis.



THANK YOU