PAGE NO :
DATE : 03/3/25

LAB - 0

Method 1 :

Creating and initializing Data in dataframe directly.

code :

```
import pandas as pd.
data = {
        "Name" : ["Preeti", "Prajval", "Prem"],
        "USN" : ["1BM22CS208", "1BM22CS220",
                 "1BM22CS341"],
        "Grade" : ["O", "S", "A+"]
       }
df = pd.DataFrame(data, index = [1, 2, 3])
df
```

output :

	Name	USN	Grade
1	Preeti	1BM22CS208	O
2	Prajval	1BM22CS220	S
3	Prem	1BM22CS341	A+

Method 2 :

Import dataset from sklearn.datasets

code :

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns = iris.feature_
                                        names)
```

PAGE NO :
DATE :

```
df['target'] = iris.target_names[iris.target]
df.head()
```

output:

	sepal length (cm)	sepal width (cm)	petal length (cm)
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3
3	4.6	3.1	1.5
4	5.0	3.6	1.4

petal width (cm)	target
0.2	setosa
0.2	setosa
0.2	setosa
0.2	setosa
0.2	setosa.

Method 3:

Importing dataset in .csv form.

code:

```
file_path = '/content/sample_data/california_housing_train.csv'
df = pd.read_csv(file_path)
df.head()
```

PAGE NO :
DATE :

output:

	longitude	lat	hous_age	t_rooms	t_bedr	popula"
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0

Method 4:

Importing dataset from platforms like UCI, Kaggle etc

code:

```
df = pd.read_csv('/content/Dataset_of_Diabetes.csv')
print(df.head())
```

output:

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c
0	502	17975	F	50	4.7	46	4.9

Yahoo Finance api

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["Reliance.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01",
                   end="2023-10-01", group_by='ticker')
```

PAGE NO :
DATE :

```
print(data.head())
print(data.shape)
print(data.columns)
reliance_data = data['RELIANCE.NS']
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
reliance_data['Da
plt.figure(fig.size = (12, 6))
plt.subplot(2, 1, 1)
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.show()
reliance_data.to_csv('reliance_stock_data.csv')
```

3/3/25

10/3/2025

PAGE NO :
DATE :

Data Pre-processing is the transformations applied to data before feeding into algorithm.

1. Data cleaning: Handling Missing values,
   Handling categorical data,
   Handling Outliers.

2. Data Transformation: Min-max scaler,
   Standard scaler.

~~Implementation~~:

```
import pandas as pd.
# loading.
df = pd.read_csv('housing.csv')

# display information
df.head()  df.info()

# display statistical information
df.describe()

# display count of unique labels for 'Ocean Proximity'
df.count(column df['Ocean Proximity']).unique()
df['Ocean_proximity'].value_count()
# Display which attributes in dataset have missing value.
df.isnull.sum()
```

PAGE NO :
DATE :

1. Adult Income Dataset (adult.csv)
   - No columns had missing values

Diabetes Dataset
- No columns had missing values.

Handling Method : Since no missing values were found, no imputation was performed.

2. Which categorical columns did you identify in the dataset? How did you encode them?

Adult Income Dataset (adult.csv)
- Categorical columns:
  workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

3. Encoding Method : One-hot encoding was applied using pd.get_dummies(df, drop_first=True), which converted these categorical columns into numerical format.

Diabetes Dataset

- Categorical columns:
  Gender, class

Encoding : One-hot encoding was applied using pd.get_dummies(df, drop_first=True) to transform them into numeric format.

PAGE NO :
DATE :

3. What is difference b/w Min-Max. scaling and standard scaling? when would you use one over the other.

Feature	
Definition	Scales values b/w a fixed range.
Formula	
Effect	
when to use	

MinMax Scaling:

Scales values b/w a fixed range

~~$x_{scaled} = x_{max} - x_{min}$~~

$x\_std = (x - x.min) / (x_{max} - x.min)$

$x\_scaled = x\_std * (max - min) + min$

Preserves original data distribution but scales it within a limited range.

when preserving relationships and original range of data is important.

Standard scaling

centers data around mean with a standard deviation of 1

$z = (x - u) / s$

changes data distribution to have zero mean and unit variance.

PAGE NO :
DATE :

when data follows a normal distribution or contains outliers.

- Min - max
  → you need data to be within fixed range
  → the dataset does not contain extreme outliers.

- Standard scale
  → the dataset has varying units and a Gaussian
  → There are significant outliers, as standardization is less sensitive to them.

Snehal B
10/3/25

PAGE NO :
DATE : 17/03/25

LAB2

ID3 algorithm implementation

```
import pandas as pd
import numpy as np
import math
from graphviz import Digraph

def cal_entropy(data, target):
    total_samples = len(data)
    class_counts = data[target].value_counts()
    entropy = 0
    for count in class_counts:
        probability = count / total_samples
        entropy -= probability * math.log2(probab
    return entropy

def cal_info_gain(data, feature, target):
    total_samples = len(data)
    weighted_entropy = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        subset_entropy = cal_entropy(subset, target)
        weighted_entropy += (len(subset) /
            total_samples) * subset_entropy
    return cal_entropy(data, target) - weight
                                    entropy

def build_tree(data, target, features, parent_
                node_class = None):
    if len(data[target].unique()) <= 1:
        return data[target].unique()[0]
```

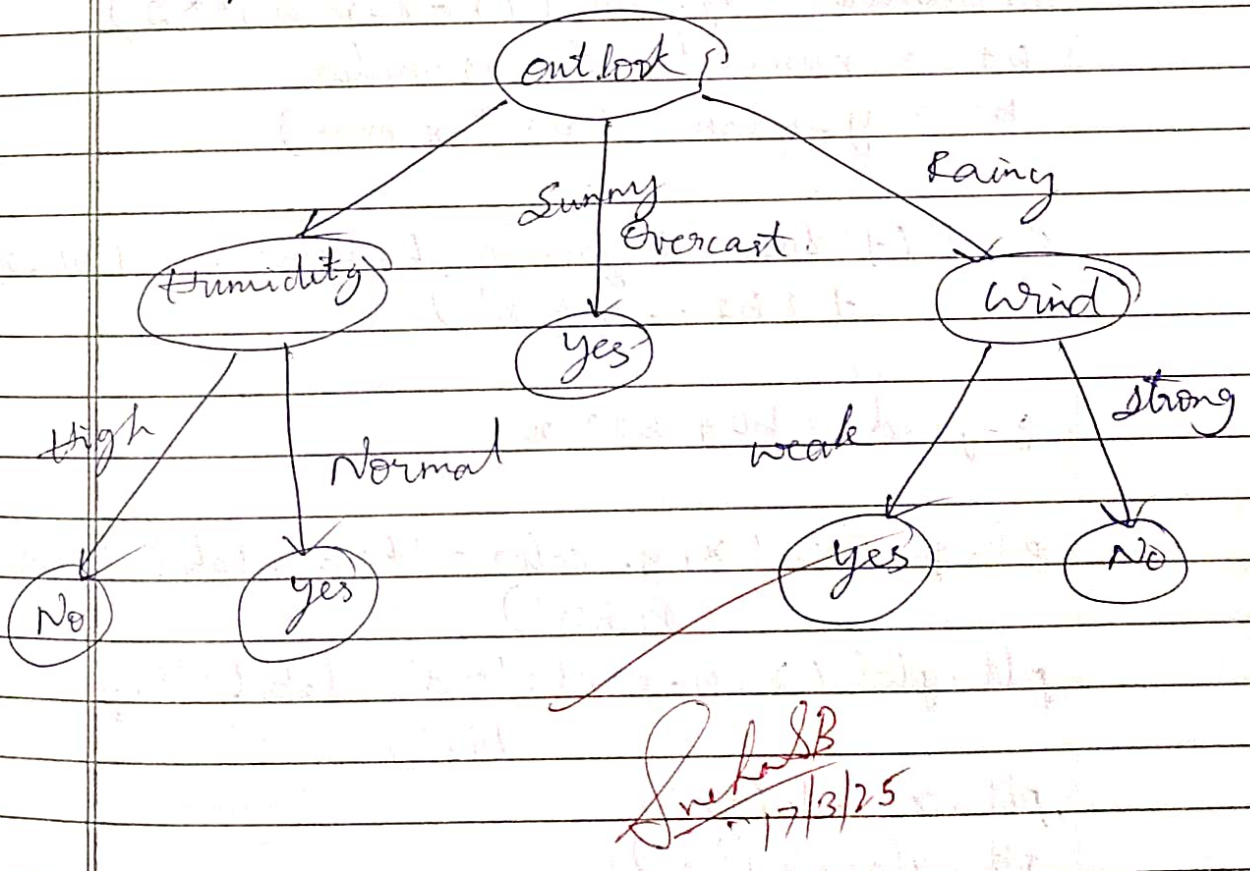
PAGE NO :
DATE :

```
if len(features) == 0:
    return parent_node_class

parent_node_class = data[target].mode()[0]

best_feature = max(features, key = lambda
    feature : calculate_info_gain(data, feature
    target))
tree = {best_feature : {}}
features.remove(best_feature)

for value in data[best_feature).unique():
    subset = data[data[best_feature] == value]
    subtree = build_tree(subset, target, features.
        copy(), parent_node_class)

    return tree

def visualize_tree(tree, dot=None, node_name
                                    =None):

    if not dot:
        dot = Digraph(comment = 'Decision
                                    tree')

    if isinstance(tree, dict):
        feature = list(tree.keys())[0]
        dot.node(node_name, label = feature)

    else:
        dot.node(node_name, label = str(tree
    return dot
```

PAGE NO :
DATE :

```
data = pd.read_csv("weather.csv")

target = 'Playtennis'
features = list(data.columns)
features.remove(target_column)
tree = build_tree(data, target, features)
dot = visualize_tree(tree)
dot.render('decision_tree')
dot
```

output:



Sneha SB
17/3/25

PAGE NO :
DATE : 24/3/23

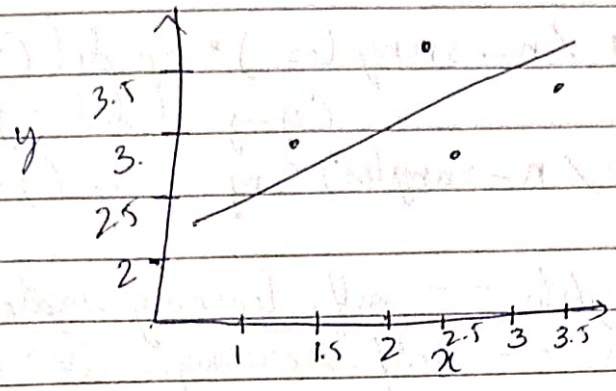
LAB - 3

LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

x_mean = np.mean(x)
y_mean = np.mean(y)

numerator = np.sum((x - x_mean) * (y - y_mean))
denominator = np.sum((x - x_mean) ** 2)
b1 = numerator / denominator
b0 = y_mean - (b1 * x_mean)

print("Linear Regression Equation : y = {b0:.4f}
      + {b1:.2f} x")

y_pred = b0 + b1 * x

plt.scatter(x, y, color = 'blue', label = 'Data
      Points')
plt.plot(x, y_pred, color = 'red', label = 'Regression
      Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

PAGE NO :
DATE :

output: $y = 2.20 + 0.60x$



LOGISTIC REGRESSION

```
import numpy as np.

class LogisticRegression:
    def __init__(self, learning_rate = 0.01,
                 n = 1000):
        self.learning_rate = learning_rate
        self.num_iterions = num_iterations
        self.weights = None
        self.bias = None.

def sigmoid (self, z):
    return 1/(1 + np.exp(-z))

def fit (self, X, y):
    n_samples, n_features = X.shape
    self.weights = np.zeros(n_features)
    self.bias = 0

for _ in range (self.num_n):
    linear_model = np.dot(X, self.weights)
                   + self.bias
```

PAGE NO :
DATE :

```
y_predicted = self.sigmoid (linear_model)

dw = (1 / n_samples) * np.dot (X.T,
                    (y_predicted - y))
db = (1 / n_samples) * np.sum (y_predicted - y)

self.weights -= self.learning_rate * dw
self.bias -= self.learning_rate * db

def predict (self, X):
    linear_model = np.dot (x, self.weights) +
                        self.bias
    y_p = self.sigmoid (linear_model)
    y_p_cls = [1 if i > 0.5 else 0 for i
                        in y_p]

    return np.array (y_p_cls)

if __name__ == "__main__":

    x = np.array ([[1, 2], [2, 3], [3, 4], [4, 5]
                    [5, 6]])
    y = np.array ([0, 0, 1, 1, 1])

    model = LogisticRegression ()
    model.fit (x, y)

    y_pred = model.predict (x)

    print ("Predictions :", y_pred)
```

PAGE NO :
DATE :

output : [0 1 1 1 1]

LAB - 01

KNN algorithm:

```
import numpy as np
import pandas as pd
from collections import Counter

data = {'feature 1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        'feature 2': [2, 4, 1, 3, 5, 7, 6, 8, 10, 9],
        'class' : ['A', 'A', 'B', 'B', 'A', 'B', 'A',
                   'B', 'B', 'A']}
df = pd.DataFrame(data)

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def knn(x_train, y_train, test_point, k = 3):
    distances = [euclidean_dist(test_point, x) for
                 x in X_train]
    k_indices = np.argsort(distances)[:k]
    k_n_l = [y_train[i] for i in k_indices]
    most_common = Counter(k_n_l).most_common(1
    return most_common[0][0]

x = df[['feature 1', 'feature 2']].values
y = df['class'].values

test_point = np.array([6, 6])
predicted_class = knn(X, y, test_point, k=3)
```

test_points = [[6,6], [7,3], [9,10]]

output:

Predicted class for [6,6] : A
Predicted class for [9,3] : B
Predicted class for [2,3] : A

SVM

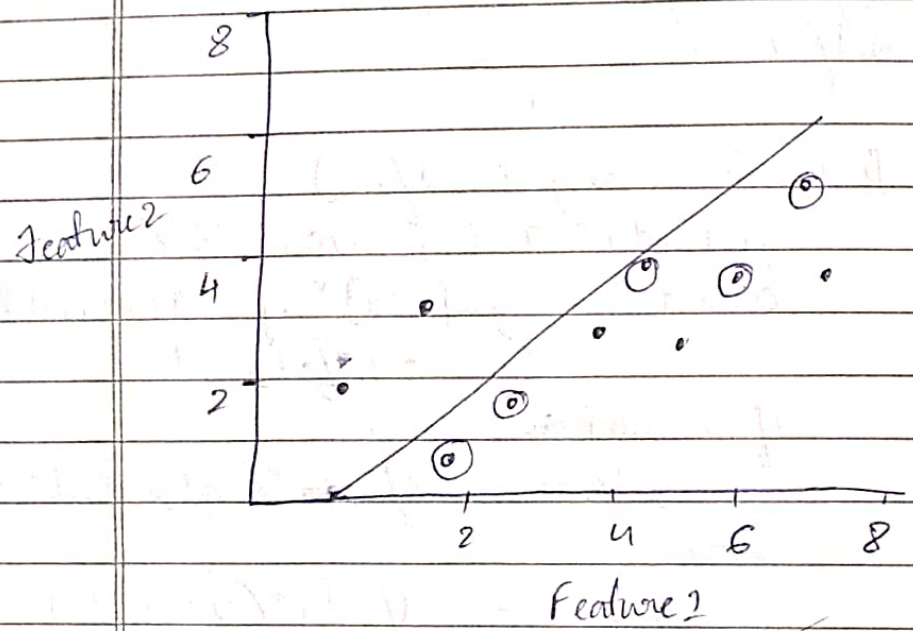
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = {'feature1': [1,2,3,4,3,6,7,8,7,10],
        'feature2': [2,4,1,3,5,7,6,8,10,9],
        }
df = pd.DataFrame(data)

x = df[['feature1', 'feature2']].values
y = df['class'].values

class SVM:
    def __init__(self, learning = 0.001, l = 0.01,
                 n = 1000):
        self.lr = learning
        self.l = l
        self.n = n
        self.w = None
        self.b = None
```

PAGE NO :
DATE :

```
def fit(self, X, y):
    n_s, n_f = x.shape
    y_ = np.where(y<=0, -1, 1)

    self.w = np.zeros(n_f)
    self.b = 0

    for _ in range(self.n):
        for idx, x_i in enumerate(x):
        condition = y_[idx] * (np.dot(x_i, self.w)
                               - self.b) >= 1

        if condition:
            self.w = self.w - lr * (2 * self.l * self.w)
        else:
            self.w = self.w - lr * (2 * self.lamda *
                                    self.w - np.dot(x_i,
                                                    y_[idx]))
            self.b = - self.lr * y_[idx]

def predict(self, x):
    app = np.dot(x, self.w) - self.b
    return np.sign(app)

clf = SVM()
clf.fit(X, y)

test_points = [[6,6], [2,3], [9,10]]
for point in test_points:
    prediction = clf.predict(np.array(point))
    print(f"Predicted class for {point}:
                                {prediction}")
```

PAGE NO :
DATE :

output : [6, 6) : 1
[2, 3) : -1
[9, 10) : 1



Lab 8B
7/4/25

LAB-06

PAGE NO :
DATE : 21/04/25

Random Forest

```
Function Random_Forest (Dataset, D, Integer
N_Trees, Integer m_features):
    Forest ← []
    For i from 1 to N_trees do:
        Sample Di ← Bootsrap_sample(D)
        Tree Ti ← Build_dec_tree (Di, m_features)
        add Ti to forest
    return Forest

Function Build_Decision_Tree.(Data Di, Integer m):
    If stopping_condition_met(Di):
        Return Leaf_Node (Most_common_label(Di))

    Features ← Random_Subset (All_features, m)
    Best_feature, threshold ← Find_Best_Split
                .(Di, Features)

    Left_data, Right_data ← Split_Data (Di,
                Best_feature, threshold)

    Left_node ← Build_Decision (Left_data, m)
    Right_node ← Build_Decision (Right_data, m)

    Return Node (Best_feature, threshold, left_node,
                Right_node)

Function Predict_Random (Forest, Instance x):
    Predictions ← [Predict_tree (Ti, x) for each
                Ti in Forest]
    Return Majority_vote (Predictions)
```

LAB-07

PAGE NO :
DATE :

AdaBoost (Boosting)

Function : AdaBoost (Dataset D, Integer T):
Initialize weights $w_i = 1/n$ for each training sample (x_i, y_i)

classifiers ← []
alphas ← []

for t from 1 to T do:
classifier h_t ← Train-weak-learner (D, weights)
Error ε_t ← $\sum [w_i * I(h_t(x_i) \neq y_i)]$
alpha α_t ← $0.5 * \log((1 - \varepsilon_t)/\varepsilon_t)$

for each i from 1 to n do:
$w_i \leftarrow w_i * \exp(-\alpha_i * y_i * h_t(x_i))$

Normalize weights: $w_i \leftarrow w_i / \sum w_i$

Append h_t to classifiers
Append α_t to Alphas

Return classifiers, alphas

Function Predict_adaBoost (classifiers, alphas, Instance x):

Total ← 0
for t from 1 to T do:
Total ← Total + $\alpha_t * h_t(x)$
Return Sign (Total)

LAB-08

PAGE NO :
DATE :

K - Means Clustering.

Function K_Means (Dataset D, Integer K,
Integer max_iter):
centroids ← Initialize_random (D, K)

For it from 1 to max_it do:
clusters ← assign_points_To_centroids (D,
centroids)
New_centroids ← compute_mean_of_clusters
(clusters)

If converged (centroids, New_centroids)
then:
Break

centroids ← New_centroids.

Return clusters, centroids

Function assign_points_to_centroids (Data D,
centroids):
clusters ← empty dictionary
For each point x_i in D do:
closest ← Find_Nearest_centroid (x_i,
centroids)
add x_i to clusters [closest]
Return clusters

Function compute_mean_of_clusters (clusters):
For each cluster:
compute mean of all points return updated
centroid

PAGE NO :
DATE :

LAB-09

Principal Component Analysis (PCA):

Function PCA (Data D, Integer K):
    standardized ← standardize_Data(D)

    CovMatrix ← Compute_covariance_matrix (standardized)
    Eigenvalues, Eigenvectors ← Eigen_Decomposition (CovMatrix)

    Sorted ← Sort_Eigenvectors_By_Eigenvalues (Eigenvalues, Eigenvectors)
    Top-k_vectors ← Select_Top_k_Eigenvectors (sorted, k)

    Reduced Data ← Project_Data (standardized, Top-k-vectors)
    Return ReducedData

Function Standardize_Data (Data D):
    For each feature:
        subtract mean and divide by standard deviation
    Return standardized D

Function Project_Data (Data, Eigenvectors):
    Return Matrix_Multiplication (Data, Eigenvectors)