





""" Aim: You have a dataset of customer information and their purchasing behavior.

The dataset includes columns such as customer\_id, age, gender, annual\_income, purchase\_amount, and purchase\_date. You need to preprocess this data to prepare] it for machine learning tasks. The preprocessing steps include:"""

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

#Rohitkumar Pandey

data = {

'customer\_id': [1, 2, 3, 4, 5],

'age': [25, 45, np.nan, 35, 50],

'gender': ['Male', 'Female', 'Female', np.nan, 'Male'],

'annual\_income': [50000, 60000, 45000, 80000, 120000],

'purchase\_amount': [200, 150, 300, 400, np.nan],

'purchase\_date': ['2023-01-01', '2023-02-15', '2023-01-20', '2023-03-05',

'2023-02-25']

}

df=pd.DataFrame(data)

print(f"Our Dataframe: \n {df}")

#Handling missing values.

print("Handling missing values.")

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

df['gender'].fillna(df['gender'].mode()[0], inplace=True)

df['purchase\_amount'].fillna(df['purchase\_amount'].median(),inplace=True)

print(f" Dataframe after Handling Missing Values : \n {df}")

#Encoding categorical variables.

print("Encoding categorical variables.")

df['gender']=df['gender'].map({'Male':0,'Female':1})

print(f" Dataframe afterEncoding categorical variables : \n {df}")

#Normalizing numerical variables.

print("Normalizing numerical variables.")

scaler=StandardScaler()

df[['age','annual\_income','purchase\_amount']]=scaler.fit\_transform(df[['age',

'annual\_income','purchase\_amount']])

print(f" Dataframe afterEncoding categorical variables : \n {df}")

#Creating a new feature: the total purchase amount for each customer.

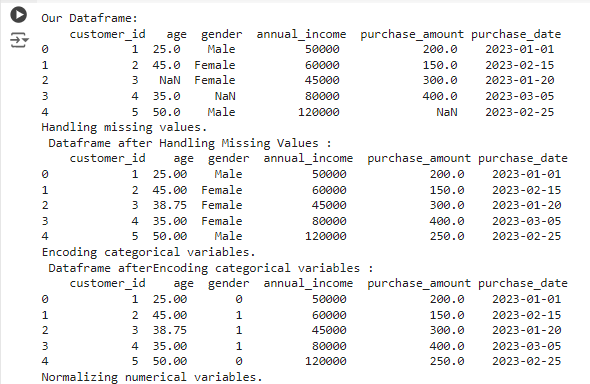
#for demonstration, lets assume 'purchase\_amount' is the total as we dont

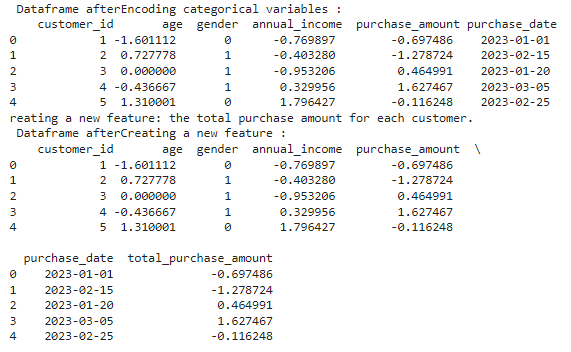
have multiple records per customer

print("reating a new feature: the total purchase amount for each customer.")

df['total\_purchase\_amount']=df['purchase\_amount']

print(f" Dataframe afterCreating a new feature : \n {df}")





"""Aim: You are given a dataset containing information about a machine learning model's

performance over multiple epochs during training. The dataset includes the loss and accuracy of the model for both the training and validation sets. Your task is to visualize this data using Matplotlib to better understand how the model's performance evolves over time and to identify potential issues such as overfitting."""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

data = {

'epoch': np.arange(1, 21),

'train\_loss': np.random.uniform(0.2, 0.6, 20),

'val\_loss': np.random.uniform(0.3, 0.7, 20),

'train\_accuracy': np.random.uniform(0.7, 0.95, 20),

'val\_accuracy': np.random.uniform(0.6, 0.9, 20)

}

df = pd.DataFrame(data)

df=df.sort\_values('epoch')

print(df.head())

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)

plt.plot(df['epoch'],df['train\_loss'],label="Training Loss",color="blue", marker='o')

plt.plot(df['epoch'],df['val\_loss'], label='Validation Loss', color="orange", marker="o")

plt.xlabel('Epoch')

plt.ylabel('Training Loss')

plt.title("Training Loss Visualization Rohitkumar")

plt.legend()

plt.subplot(1,2,2)

plt.plot(df['epoch'],df['train\_accuracy'],label="Training Accuarcy",color="blue", marker='o')

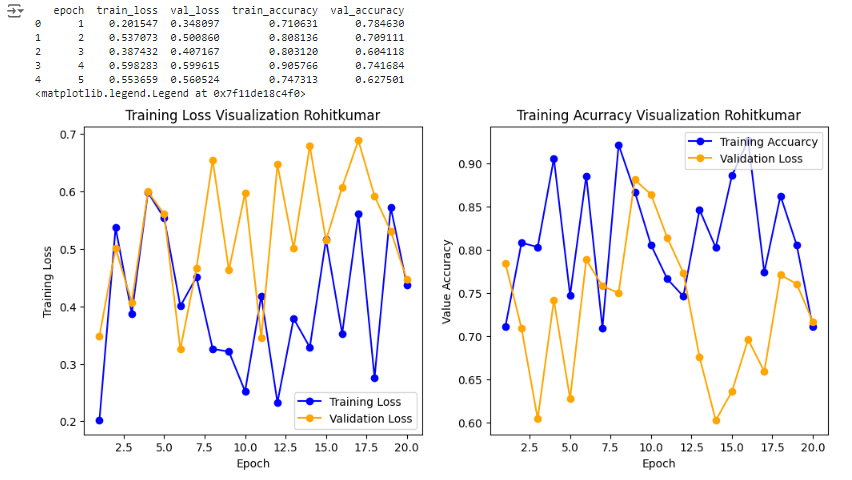
plt.plot(df['epoch'],df['val\_accuracy'], label='Validation Loss', color="orange", marker="o")

plt.xlabel('Epoch')

plt.ylabel('Value Accuracy')

plt.title("Training Acurracy Visualization Rohitkumar")

plt.legend()



# prompt: Aim: Write a program to generate telecomm dataset which contains age,

gender, usage minutes, churn status and draw pair plot of it.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

# Set a seed for reproducibility

np.random.seed(0)

# Number of samples to generate

num\_samples = 1000

# Generate age data (normally distributed)

age = np.random.normal(loc=35, scale=10, size=num\_samples).astype(int)

age = np.clip(age, 18, 80) # Ensure ages are within a reasonable range

# Generate gender data (0 for male, 1 for female)

gender = np.random.choice([0, 1], size=num\_samples)

# Generate usage minutes (correlated with age and gender)

usage\_minutes = 100 + 5 \* age + 20 \* gender + np.random.normal(scale=50,

size=num\_samples)

usage\_minutes = np.clip(usage\_minutes, 0, 500) # Ensure usage minutes are non-negative

# Generate churn status (higher usage less likely to churn)

churn\_prob = np.exp(-usage\_minutes / 500)

churn = np.random.binomial(1, churn\_prob)

# Create a DataFrame

data = {

'age': age,

'gender': gender,

'usage\_minutes': usage\_minutes,

'churn': churn

}

df = pd.DataFrame(data)

# Map gender values to labels

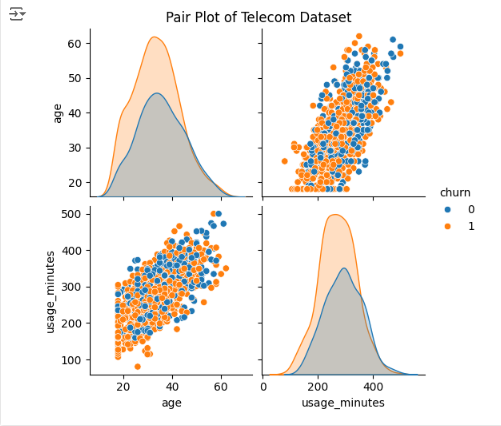
df['gender'] = df['gender'].map({0: 'Male', 1: 'Female'})

# Pair plot

sns.pairplot(df, hue='churn', vars=['age', 'usage\_minutes'], diag\_kind='kde')

plt.suptitle('Pair Plot of Telecom Dataset', y=1.02)

plt.show()



Exp-6

import tkinter as tk

from tkinter import messagebox

import pandas as pd

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

import seaborn as sn

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# Function to train the model

def train\_model():

try:

# Load the digits dataset

digits = load\_digits()

df = pd.DataFrame(digits.data)

df['target'] = digits.target

# Split the dataset

X = df.drop('target', axis='columns')

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest model

model = RandomForestClassifier(n\_estimators=int(estimator\_entry.get()))

model.fit(X\_train, y\_train)

# Test accuracy

accuracy = model.score(X\_test, y\_test)

accuracy\_label.config(text=f"Model Accuracy: {accuracy:.2f}")

# Prediction and Confusion Matrix

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

# Plot confusion matrix

fig = plt.figure(figsize=(6, 4))

sn.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

# Clear previous canvas and display the new one

for widget in canvas\_frame.winfo\_children():

widget.destroy()

canvas = FigureCanvasTkAgg(fig, master=canvas\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack()

except Exception as e:

messagebox.showerror("Error", str(e))

# Set up the GUI window

window = tk.Tk()

window.title("Random Forest Classifier GUI by Rohit pandey 211P002")

# Label for number of estimators

estimator\_label = tk.Label(window, text="Number of Estimators:")

estimator\_label.pack(pady=10)

# Entry for number of estimators

estimator\_entry = tk.Entry(window)

estimator\_entry.insert(0, "20") # Default value

estimator\_entry.pack(pady=10)

# Train button

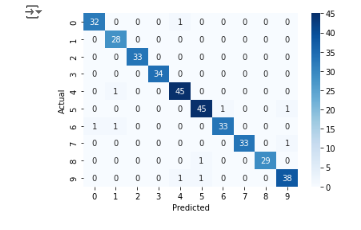
train\_button = tk.Button(window, text="Train Model", command=train\_model)

train\_button.pack(pady=20)

# Accuracy label

accuracy\_label = tk.Label(window, text="Model Accuracy: N/A")

accuracy\_label.pack(pady=10)

# Frame to hold the confusion matrix

canvas\_frame = tk.Frame(window)

canvas\_frame.pack(pady=20)

# Run the GUI event loop

window.mainloop()

import tkinter as tk

from tkinter import messagebox, ttk

import pandas as pd

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

import seaborn as sn

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# Function to train the model

def train\_model():

try:

# Load the digits dataset

digits = load\_digits()

df = pd.DataFrame(digits.data)

df['target'] = digits.target

X = df.drop('target', axis='columns')

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Select the base estimator based on user selection

base\_estimator\_name = estimator\_combo.get()

if base\_estimator\_name == "Decision Tree":

base\_estimator = DecisionTreeClassifier()

elif base\_estimator\_name == "Logistic Regression":

base\_estimator = LogisticRegression(max\_iter=1000)

n\_estimators = int(estimator\_entry.get())

model = BaggingClassifier(base\_estimator=base\_estimator, n\_estimators=n\_estimators)

model.fit(X\_train, y\_train)

accuracy = model.score(X\_test, y\_test)

accuracy\_label.config(text=f"Model Accuracy: {accuracy:.2f}")

# Make predictions and create a confusion matrix

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

# Plot the confusion matrix using seaborn

fig, ax = plt.subplots(figsize=(6, 4))

sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=ax)

ax.set\_xlabel("Predicted")

ax.set\_ylabel("Actual")

# Clear any previous plots and display the new plot in the canvas

for widget in canvas\_frame.winfo\_children():

widget.destroy()

canvas = FigureCanvasTkAgg(fig, master=canvas\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack()

except Exception as e:

messagebox.showerror("Error", str(e))

# Set up the GUI window

window = tk.Tk()

window.title("Bagging Classifier GUI by ROhitkumar 211P002")

# Label for selecting the base estimator

estimator\_label = tk.Label(window, text="Select Base Estimator:")

estimator\_label.pack(pady=10)

# Combobox for base estimator selection

estimator\_combo = ttk.Combobox(window, values=["Decision Tree", "Logistic Regression"])

estimator\_combo.current(0) # Default to Decision Tree

estimator\_combo.pack(pady=10)

# Label for the number of estimators

estimator\_entry\_label = tk.Label(window, text="Number of Estimators:")

estimator\_entry\_label.pack(pady=10)

# Entry box for number of estimators input

estimator\_entry = tk.Entry(window)

estimator\_entry.insert(0, "10") # Default value is 10

estimator\_entry.pack(pady=10)

# Button to start training the model

train\_button = tk.Button(window, text="Train Model", command=train\_model)

train\_button.pack(pady=20)

# Label to display accuracy

accuracy\_label = tk.Label(window, text="Model Accuracy: N/A")

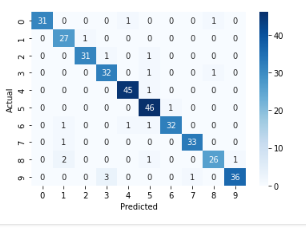
accuracy\_label.pack(pady=10)

canvas\_frame = tk.Frame(window)

canvas\_frame.pack(pady=20)

# Run the Tkinter event loop

window.mainloop()



# Post lab 1

import tkinter as tk

from tkinter import ttk, messagebox

from sklearn.datasets import make\_regression

from sklearn.ensemble import BaggingRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import mean\_squared\_error, r2\_score

# Create the GUI application

def create\_gui():

# Function to train the model and perform Grid Search

def train\_model():

try:

# Generate a synthetic regression dataset

X, y = make\_regression(n\_samples=1000, n\_features=10, noise=0.1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Select base estimator from the GUI

base\_estimator\_name = base\_estimator\_combo.get()

if base\_estimator\_name == "Decision Tree Regressor":

base\_estimator = DecisionTreeRegressor(random\_state=42)

elif base\_estimator\_name == "Linear Regressor":

base\_estimator = LinearRegression()

# Get hyperparameters from the GUI

n\_estimators = int(n\_estimators\_entry.get())

max\_samples = float(max\_samples\_entry.get())

bootstrap = bool(bootstrap\_var.get())

# Define Bagging Regressor with the base estimator

bagging\_regressor = BaggingRegressor(

n\_estimators=n\_estimators,

max\_samples=max\_samples,

bootstrap=bootstrap,

random\_state=42

)

# Define Grid Search parameters

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_samples': [0.5, 0.7, 1.0],

'bootstrap': [True, False]

}

# Perform Grid Search

bagging\_regressor\_grid = GridSearchCV(bagging\_regressor, param\_grid, cv=5)

bagging\_regressor\_grid.fit(X\_train, y\_train)

# Best model from Grid Search

best\_model = bagging\_regressor\_grid.best\_estimator\_

# Predict using the best model

y\_pred = best\_model.predict(X\_test)

# Calculate Mean Squared Error and R^2 Scores

mse = mean\_squared\_error(y\_test, y\_pred)

train\_r2 = best\_model.score(X\_train, y\_train)

test\_r2 = best\_model.score(X\_test, y\_test)

best\_r2 = bagging\_regressor\_grid.best\_score\_

best\_params = bagging\_regressor\_grid.best\_params\_

# Update result labels with MSE and R^2 Scores

result\_label.config(text=f"MSE: {mse:.4f}")

r2\_train\_label.config(text=f"Train R² Score: {train\_r2:.3f}")

r2\_test\_label.config(text=f"Test R² Score: {test\_r2:.3f}")

best\_r2\_label.config(text=f"Best R² Score (Grid Search): {best\_r2:.3f}")

best\_params\_label.config(text=f"Best Parameters: {best\_params}")

except Exception as e:

messagebox.showerror("Error", str(e))

# Create the main window

window = tk.Tk()

window.title("Bagging Regressor GUI with Grid Search byRohitpandey 211P002")

# Base Estimator selection

base\_estimator\_label = tk.Label(window, text="Select Base Estimator:")

base\_estimator\_label.pack(pady=5)

base\_estimator\_combo = ttk.Combobox(window, values=["Decision Tree Regressor", "Linear Regressor"])

base\_estimator\_combo.current(0)

base\_estimator\_combo.pack(pady=5)

# Number of Estimators

n\_estimators\_label = tk.Label(window, text="Number of Estimators:")

n\_estimators\_label.pack(pady=5)

n\_estimators\_entry = tk.Entry(window)

n\_estimators\_entry.insert(0, "100")

n\_estimators\_entry.pack(pady=5)

# Max Samples

max\_samples\_label = tk.Label(window, text="Max Samples (0.0 to 1.0):")

max\_samples\_label.pack(pady=5)

max\_samples\_entry = tk.Entry(window)

max\_samples\_entry.insert(0, "0.5")

max\_samples\_entry.pack(pady=5)

# Bootstrap option

bootstrap\_var = tk.IntVar()

bootstrap\_check = tk.Checkbutton(window, text="Bootstrap", variable=bootstrap\_var)

bootstrap\_check.pack(pady=5)

# Train Model Button

train\_button = tk.Button(window, text="Train Model", command=train\_model)

train\_button.pack(pady=10)

# Label to display results

result\_label = tk.Label(window, text="MSE: N/A")

result\_label.pack(pady=5)

r2\_train\_label = tk.Label(window, text="Train R² Score: N/A")

r2\_train\_label.pack(pady=5)

r2\_test\_label = tk.Label(window, text="Test R² Score: N/A")

r2\_test\_label.pack(pady=5)

best\_r2\_label = tk.Label(window, text="Best R² Score (Grid Search): N/A")

best\_r2\_label.pack(pady=5)

best\_params\_label = tk.Label(window, text="Best Parameters: N/A")

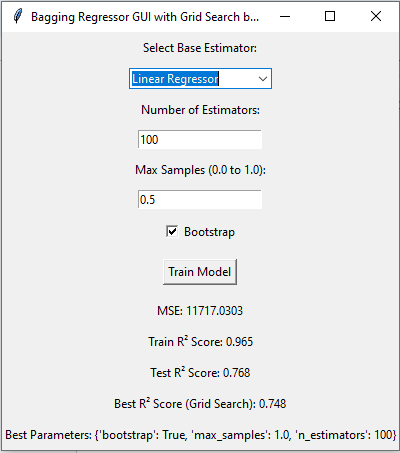
best\_params\_label.pack(pady=5)

# Start the main loop

window.mainloop()

# Run the GUI

create\_gui()



Exp-7

import pandas as pd

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

# Load Iris dataset and create DataFrame

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['target'] = iris.target

df['flower\_name'] = df['target'].apply(lambda x: iris.target\_names[x])

# Split dataset by class

df0, df1, df2 = df[:50], df[50:100], df[100:]

# Plot Sepal and Petal dimensions for Setosa and Versicolor

def plot\_dimensions(x, y, xlabel, ylabel, title):

plt.title(f"By Rohitkumar Pandey 211P002\n{title}")

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.scatter(df0[x], df0[y], color="red", marker='+', label="Setosa")

plt.scatter(df1[x], df1[y], color="blue", marker='.', label="Versicolor")

plt.legend()

plt.show()

plot\_dimensions('sepal length (cm)', 'sepal width (cm)', 'Sepal Length', 'Sepal Width', "Plot Sepal Length vs Sepal Width")

plot\_dimensions('petal length (cm)', 'petal width (cm)', 'Petal Length', 'Petal Width', "Plot Petal Length vs Petal Width")

# Prepare training and testing data

X = df.drop(['target', 'flower\_name'], axis='columns')

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Create and train SVM models with different parameters

models = {

'default': SVC(),

'C=1': SVC(C=1),

'C=10': SVC(C=10),

'gamma=10': SVC(gamma=10),

'linear': SVC(kernel='linear')

}

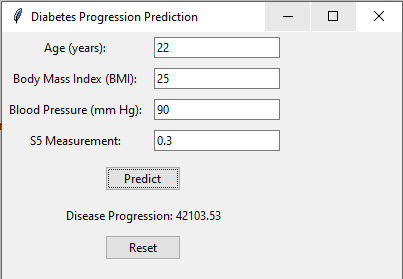
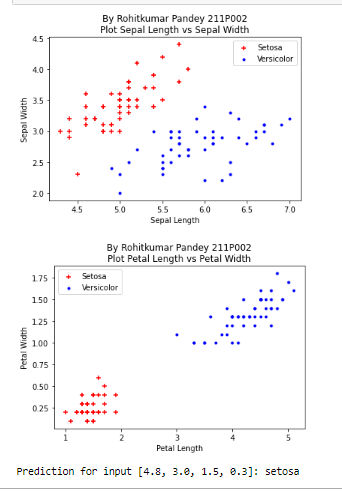
for name, model in models.items():

model.fit(X\_train, y\_train)

# Example prediction

prediction = models['default'].predict([[4.8, 3.0, 1.5, 0.3]])

print(f"Prediction for input [4.8, 3.0, 1.5, 0.3]: {iris.target\_names[prediction[0]]}")



# postlab 1 rohitkumar

import tkinter as tk

from tkinter import ttk, messagebox

import pandas as pd

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

# Load the Diabetes dataset

diabetes = load\_diabetes()

df = pd.DataFrame(diabetes.data, columns=diabetes.feature\_names)

df['target'] = diabetes.target

# Select fewer features correctly

X = df[['age', 'bmi', 'bp', 's5']] # 'bp' is blood pressure and 's5' is a selected feature

y = df['target']

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model once

model = LinearRegression()

model.fit(X\_train, y\_train)

# Create the GUI window

window = tk.Tk()

window.title("Diabetes Progression Prediction")

window.geometry("400x400")

# Create input fields for selected features

labels = {

'age': "Age (years)",

'bmi': "Body Mass Index (BMI)",

'bp': "Blood Pressure (mm Hg)",

's5': "S5 Measurement"

}

entries = []

for i, (key, label\_text) in enumerate(labels.items()):

label = ttk.Label(window, text=label\_text + ":")

label.grid(row=i, column=0, padx=5, pady=5)

entry = ttk.Entry(window)

entry.grid(row=i, column=1, padx=5, pady=5)

entries.append(entry)

# Create a button to predict

def predict():

try:

# Get input values

input\_values = [float(entry.get()) for entry in entries]

# Make prediction

prediction = model.predict([input\_values])[0]

# Display prediction

result\_label.config(text="Disease Progression: {:.2f}".format(prediction))

except ValueError:

messagebox.showerror("Input Error", "Invalid input. Please enter valid numbers for all features.")

except Exception as e:

messagebox.showerror("Prediction Error", f"An error occurred: {str(e)}")

# Create a button to predict

predict\_button = ttk.Button(window, text="Predict", command=predict)

predict\_button.grid(row=len(labels), column=0, columnspan=2, padx=5, pady=10)

# Create a label to display the prediction

result\_label = ttk.Label(window, text="")

result\_label.grid(row=len(labels) + 1, column=0, columnspan=2, padx=5, pady=5)

# Create a reset button to clear inputs

def reset():

for entry in entries:

entry.delete(0, tk.END)

result\_label.config(text="")

reset\_button = ttk.Button(window, text="Reset", command=reset)

reset\_button.grid(row=len(labels) + 2, column=0, columnspan=2, padx=5, pady=5)

# Start the GUI event loop

window.mainloop()

Exp8

import pandas as pd

import joblib

from tkinter import \*

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from sklearn.cluster import KMeans

# Load the dataset

df = pd.read\_csv("Mall\_Customers.csv")

# Prepare the data for clustering

x = df[['Annual Income (k$)', 'Spending Score (1-100)']]

# Initialize KMeans with the desired number of clusters

k\_mean = KMeans(n\_clusters=5, random\_state=42)

y\_mean = k\_mean.fit\_predict(x)

# Function to show entry fields and predict cluster

def show\_entry\_fields():

p1 = int(e1.get())

p2 = int(e2.get())

# Predict the cluster for the new input

result = k\_mean.predict([[p1, p2]])

# Result output

print("This customer belongs to cluster no:", result[0])

# Display cluster information

cluster\_info = {

0: "customer with medium annual income & medium annual spending score",

1: "customer with high annual income & low annual spending score",

2: "customer with low annual income & low annual spending score",

3: "customer with low annual income & high annual spending score",

4: "customer with high annual income & high annual spending score"

}

# Clear previous labels if any

for widget in master.grid\_slaves():

if int(widget.grid\_info()["row"]) >= 4: # Assuming info labels start from row 4

widget.destroy()

Label(master, text=cluster\_info[result[0]]).grid(row=4)

# Create the main window

master = Tk()

master.title("Customer Segmentation using Machine Learning")

# Title label

Label(master, text="Customer Segmentation using Machine Learning", bg="black", fg="white").grid(row=0, columnspan=2)

# Input labels and entries

Label(master, text="Annual Income").grid(row=1)

Label(master, text="Spending Score").grid(row=2)

e1 = Entry(master)

e2 = Entry(master)

e1.grid(row=1, column=1)

e2.grid(row=2, column=1)

# Predict button

Button(master, text='Predict', command=show\_entry\_fields).grid(row=3)

# Plotting

figure3 = plt.Figure(figsize=(5, 4), dpi=100)

ax3 = figure3.add\_subplot(111)

# Sample scatter plot for clusters

# Plot the existing clusters

for i in range(5): # Since n\_clusters=5

ax3.scatter(x.iloc[y\_mean == i, 0], x.iloc[y\_mean == i, 1], s=100, label=f'Cluster {i}')

# Set labels and title

ax3.set\_xlabel('Annual Income (k$)')

ax3.set\_ylabel('Spending Score (1-100)')

ax3.set\_title('Annual Income vs Spending Score')

ax3.legend()

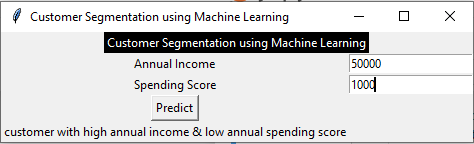
# Displaying figure in Tkinter

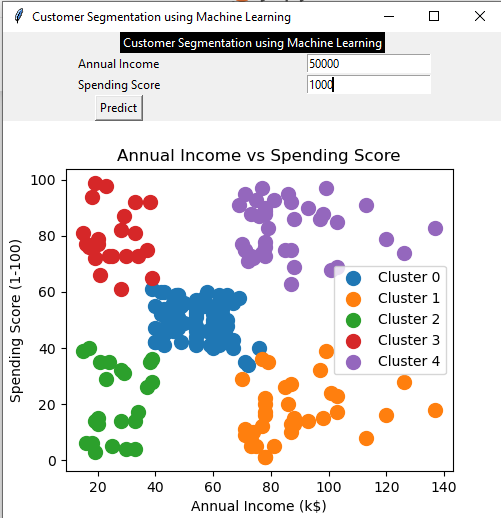
scatter3 = FigureCanvasTkAgg(figure3, master)

scatter3.get\_tk\_widget().grid(row=5, columnspan=2)

# Run the application

master.mainloop()





# Postlab 1

import pandas as pd

import joblib

from tkinter import \*

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from sklearn.cluster import KMeans

# Load the dataset

df = pd.read\_csv("Placement.csv")

# Prepare the data for clustering

x = df[['cgpa', 'package']]

# Initialize KMeans with the desired number of clusters

k\_mean = KMeans(n\_clusters=5, random\_state=42)

y\_mean = k\_mean.fit\_predict(x)

# Function to show entry fields and predict cluster

def show\_entry\_fields():

p1 = float(e1.get())

p2 = float(e2.get())

# Predict the cluster for the new input

result = k\_mean.predict([[p1, p2]])

# Result output

print("This student belongs to cluster no:", result[0])

# Display cluster information

cluster\_info = {

0: "students with medium CGPA and medium package",

1: "students with high CGPA and low package",

2: "students with low CGPA and low package",

3: "students with low CGPA and high package",

4: "students with high CGPA and high package"

}

# Clear previous labels if any

for widget in master.grid\_slaves():

if int(widget.grid\_info()["row"]) >= 4: # Assuming info labels start from row 4

widget.destroy()

Label(master, text=cluster\_info[result[0]]).grid(row=4)

# Create the main window

master = Tk()

master.title("Student Placement Segmentation by Vishal Boss")

# Title label

Label(master, text="Student Placement Segmentation using Machine Learning", bg="Yellow", fg="black").grid(row=0, columnspan=2)

# Input labels and entries

Label(master, text="CGPA").grid(row=1)

Label(master, text="Package").grid(row=2)

e1 = Entry(master)

e2 = Entry(master)

e1.grid(row=1, column=1)

e2.grid(row=2, column=1)

# Predict button

Button(master, text='Predict', command=show\_entry\_fields).grid(row=3)

# Plotting

figure3 = plt.Figure(figsize=(5, 4), dpi=100)

ax3 = figure3.add\_subplot(111)

# Sample scatter plot for clusters

# Plot the existing clusters

for i in range(5): # Since n\_clusters=5

ax3.scatter(x.iloc[y\_mean == i, 0], x.iloc[y\_mean == i, 1], s=100, label=f'Cluster {i}')

# Set labels and title

ax3.set\_xlabel('CGPA')

ax3.set\_ylabel('Package')

ax3.set\_title('CGPA vs Package')

ax3.legend()

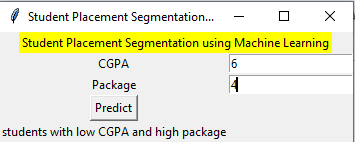
# Displaying figure in Tkinter

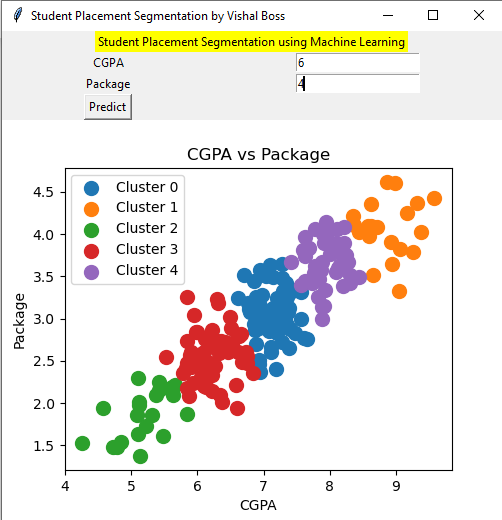
scatter3 = FigureCanvasTkAgg(figure3, master)

scatter3.get\_tk\_widget().grid(row=5, columnspan=2)

# Run the application

master.mainloop()





Exp9

# Step 1: Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

# Step 2: Load the Iris dataset

iris = load\_iris()

iris\_data = iris.data

iris\_target = iris.target

iris\_target\_names = iris.target\_names

# Step 3: Standardize the data

scaler = StandardScaler()

iris\_data\_scaled = scaler.fit\_transform(iris\_data)

# Step 4: Perform PCA

pca = PCA(n\_components=2) # Reduce to 2 principal components for visualization

iris\_pca = pca.fit\_transform(iris\_data\_scaled)

# Step 5: Visualize the results

plt.figure(figsize=(8, 6))

scatter = plt.scatter(iris\_pca[:, 0], iris\_pca[:, 1], c=iris\_target, cmap='viridis', edgecolor='k', s=100)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA of Iris Dataset By211P002 Rohitkumar')

# Modified legend creation

# Instead of using scatter.legend\_elements(), manually create the legend handles

# using the unique target values and corresponding labels.

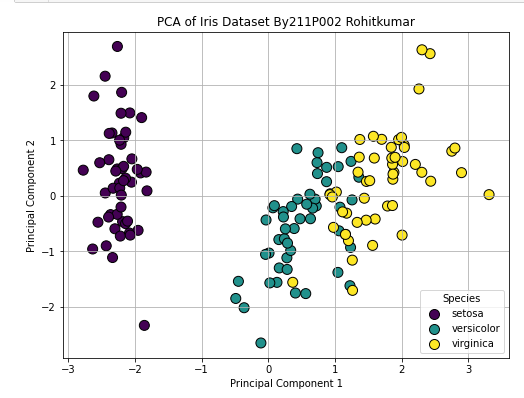
for target, target\_name in zip(np.unique(iris\_target), iris\_target\_names):

plt.scatter([], [], c=[plt.cm.viridis(target / 2)], label=target\_name, edgecolor='k', s=100)

plt.legend(title="Species")

plt.grid()

plt.show()



# Step 1: Import necessary libraries # ROhitkumar 211P002

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# Step 2: Load the Iris dataset

iris = load\_iris()

iris\_data = iris.data

iris\_target = iris.target

iris\_target\_names = iris.target\_names

# Step 3: Standardize the data

scaler = StandardScaler()

iris\_data\_scaled = scaler.fit\_transform(iris\_data)

# Step 4: Perform PCA

pca = PCA(n\_components=3) # Reduce to 3 principal components for more features

iris\_pca = pca.fit\_transform(iris\_data\_scaled)

# Step 5: Visualize the cumulative explained variance

explained\_variance = pca.explained\_variance\_ratio\_

cumulative\_explained\_variance = np.cumsum(explained\_variance)

plt.figure(figsize=(8, 5))

plt.plot(range(1, 4), cumulative\_explained\_variance, marker='o', linestyle='--', color='b')

plt.title('Cumulative Explained Variance by Principal Components\nRohitkumar Pandey 211P002')

plt.xlabel('Number of Principal Components')

plt.ylabel('Cumulative Explained Variance')

plt.grid()

plt.show()

# Step 6: 2D Visualization of the first two principal components

plt.figure(figsize=(10, 7))

scatter = plt.scatter(iris\_pca[:, 0], iris\_pca[:, 1], c=iris\_target, cmap='viridis', edgecolor='k', s=100)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('2D PCA of Iris Dataset By Rohitkumar Pandey 211P002')

# Manually create legend handles for each target class

handles = []

for i in range(len(iris\_target\_names)):

handles.append(plt.scatter([], [], marker='o', s=100, edgecolor='k', c=[plt.cm.viridis(i / (len(iris\_target\_names) - 1))]))

# Use colormap to match scatter plot colors

plt.legend(handles, iris\_target\_names, title="Species") # Use the created handles

plt.grid()

plt.show()

# Step 7: 3D Visualization of the first three principal components

fig = plt.figure(figsize=(10, 7))

ax = fig.add\_subplot(111, projection='3d')

scatter = ax.scatter(iris\_pca[:, 0], iris\_pca[:, 1], iris\_pca[:, 2], c=iris\_target, cmap='viridis', edgecolor='k', s=100)

ax.set\_xlabel('Principal Component 1')

ax.set\_ylabel('Principal Component 2')

ax.set\_zlabel('Principal Component 3')

ax.set\_title('3D PCA of Iris Dataset ByRohitkumar Pandey 211P002')

# Manually create legend handles for each target class

handles\_3d = []

for i in range(len(iris\_target\_names)):

handles\_3d.append(ax.scatter([], [], [], marker='o', s=100, edgecolor='k', c=[plt.cm.viridis(i / (len(iris\_target\_names) - 1))]))

ax.legend(handles\_3d, iris\_target\_names, title="Species") # Use created handles 3D plot

plt.show()

# Step 8: Print explained variance ratio for each principal component

print("Explained variance ratio for each principal component:", explained\_variance)

