

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

B22CI0501 – MACHINE LEARNING

PREDICTION OF DIABETES PAITENTS

USING DIABETES DATASET

SUBMITTED BY

NAME: TARUN KUMAR S

SRN: R22EA063

SEMESTER & BRANCH: 5th SEM AIML B

DATE OF SUBMISSION:

SIGNATURE:

VERIFIED BY:

MARKS:

SIGNATURE OF FACULTY:

QUESTIONS TO BE ANSWERED:

1. What is the nature of the dataset (structured, unstructured, or semi-structured)?

* The dataset is **structured**, with clear columns representing variables (features) and the "Survived" as the target label. It is a **labelled** dataset.

1. What is the source of the dataset (public, proprietary, or in-house)?

* The Titanic dataset primarily comes from passenger records of the RMS Titanic, a British passenger liner that sank on its maiden voyage in 1912.

1. What is the size of the dataset in terms of samples or instances before applying PCA?

* The Titanic dataset from Kaggle typically has:
* 891 samples (instances) in the training set.
* 418 samples in the test set.

1. Are the labels accurate and meaningful for the intended task?

* The labels in the "Survived" column are binary (0 for Not Survived, 1 for Survived). They appear meaningful for the task of predicting.

1. How consistent is the labeling across the dataset?

* Since the dataset is structured and fully populated, the labeling seems consistent across all instances.

1. Are there missing values or corrupted data points in the dataset?

* No missing values are detected in this dataset. However, some values (like zero values for Age and Cabin) may need to be addressed as potential data issues or outliers.

1. Are there any legal or ethical issues related to using this dataset?

* No, there is no legal or ethical issues related to using this dataset.

1. What is the class distribution, and is it balanced or imbalanced?

* The target class distribution (Survived) will need to be checked to see if it's balanced or imbalanced.
* Based on analysis, The dataset is slightly imbalanced:

Class 0 (Not Survived): 60%

Class 1 (Survived): 40%

CODE OF THE ML MODEL:

#Step 1: Data Pre-processing step

#importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay, classification\_report

from sklearn.preprocessing import StandardScaler

#importing dataset

#Load the data

train\_data = pd.read\_csv('../dataset/raw/train.csv')

test\_data = pd.read\_csv('../dataset/raw/test.csv')

test\_data\_survived = pd.read\_csv('../dataset/raw/gender\_submission.csv')

# Handle missing values

train\_data.fillna(method='ffill', inplace=True)

test\_data.fillna(method='ffill', inplace=True)

# Convert categorical variables to numerical

train\_data = pd.get\_dummies(train\_data, columns=['Sex', 'Embarked'], drop\_first=True)

test\_data = pd.get\_dummies(test\_data, columns=['Sex', 'Embarked'], drop\_first=True)

#Check the data

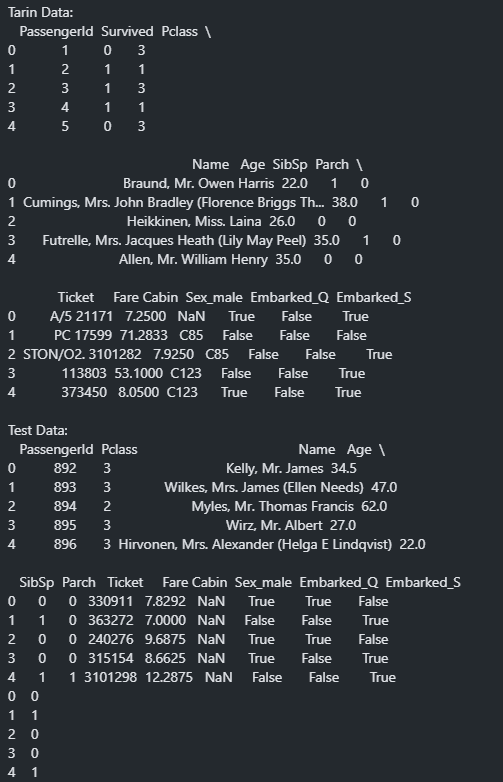
print("Tarin Data:")

print(train\_data.head())

print("\nTest Data:")

print(test\_data.head())

print(test\_data\_survived["Survived"].head())



# Select features and target

X\_train = train\_data.drop(['Survived', 'Name', 'Ticket', 'Cabin'], axis=1)

y\_train = train\_data['Survived']

X\_test = test\_data.drop(['Name', 'Ticket', 'Cabin'], axis=1)

#Fit the train and test data

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test).

#Displaying the data

print("Training set of Independent after features scaling:")

print(X\_train[:5])

print("\nTraining set of Dependent:")

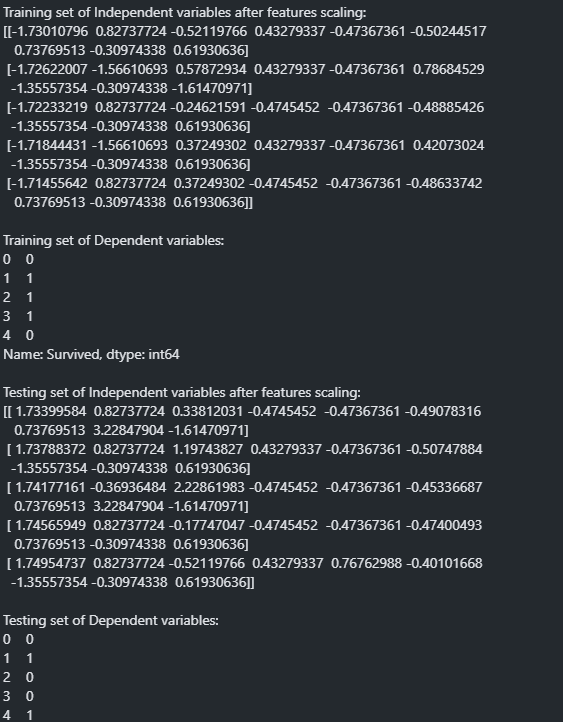
print(y\_train[:5])

print("\nTesting set of Independent after features scaling:")

print(X\_test[:5])

print("\nesting set of Dependent:")

print(y\_test[:5])



#Step 2: Training the model

#Fitting Logistic Regression to the training set

model = LogisticRegression(max\_iter=10, random\_state=7)

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

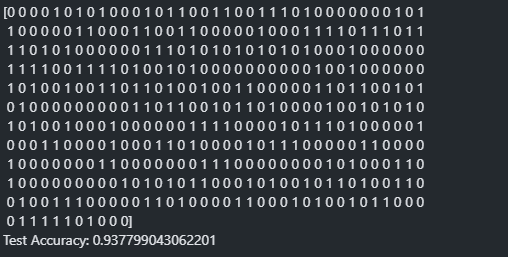


#Step 3: Predicting the test set result

# Predict on the test data

test\_predictions = model.predict(X\_test)

print(test\_predictions)



#Step 4: Creating the confusion matrix and checking other performance metrics

# Calculate the accuracy

accuracy = accuracy\_score(y\_test, test\_predictions)

print(f'Test Accuracy: {accuracy}')

# Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, test\_predictions)

print("\nConfusion Matrix:")

print(conf\_matrix)

# Display the classification report

class\_report = classification\_report(y\_test, test\_predictions, target\_names=['Not Survived', 'Survived'], output\_dict=True)

class\_report\_df = pd.DataFrame(class\_report).transpose()

class\_report\_df = class\_report\_df.round(2)

print("\nFormatted Classification Report:")

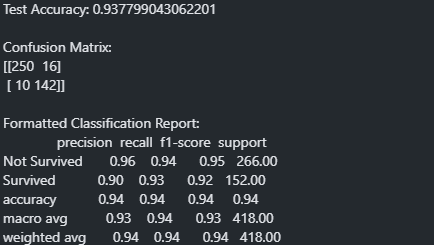
print(class\_report\_df)

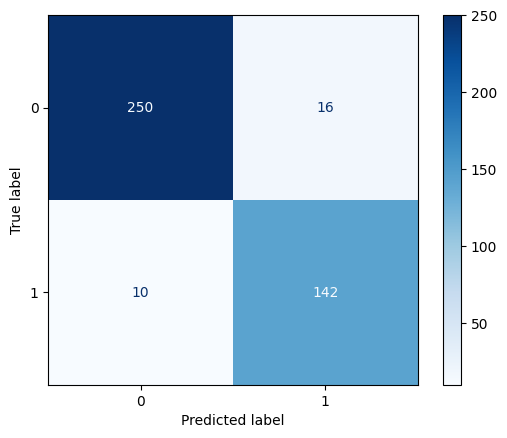
# Display the confusion matrix using matplotlib

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix)

disp.plot(cmap=plt.cm.Blues)

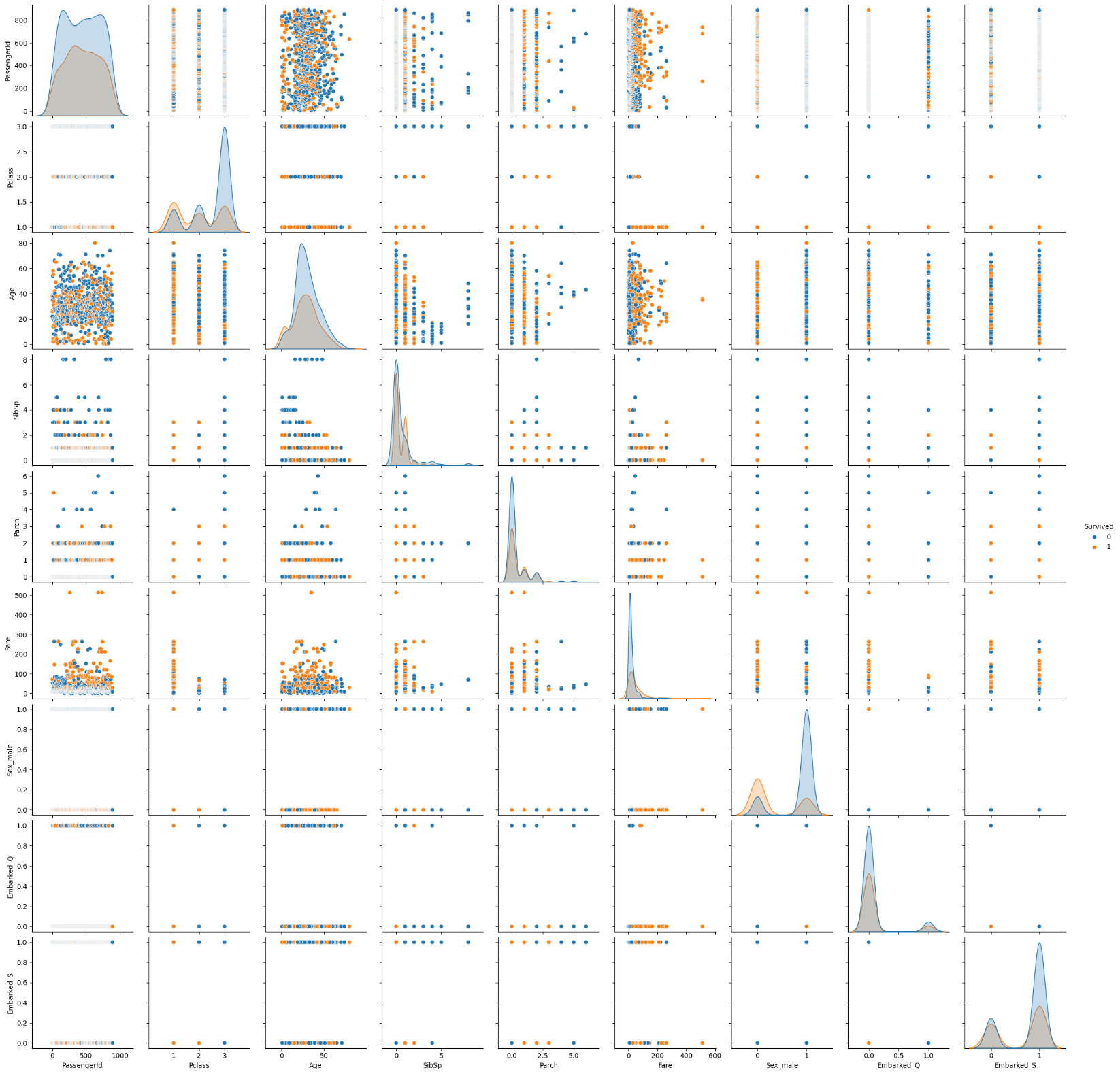
plt.show()





# Pair plot to visualize pairwise relationships between features

sns.pairplot(train\_data, hue='Survived', diag\_kind='kde')

plt.show()

**Predicting the output based on user input**

#Predicting the output based on user input

import joblib

import numpy as np

import os

from sklearn.preprocessing import StandardScaler

# Function to load all models from a specified folder

def load\_models\_from\_folder(folder\_path):

models = {}

for filename in os.listdir(folder\_path):

if filename.endswith('.pkl'):

model\_name = filename.split('.')[0]

model\_path = os.path.join(folder\_path, filename)

models[model\_name] = joblib.load(model\_path)

return models

# Function to take input for the Titanic dataset

def get\_titanic\_input():

Pclass = int(input("Enter Pclass (1, 2, or 3): "))

Sex = input("Enter Sex (male or female): ")

Age = float(input("Enter Age: "))

SibSp = int(input("Enter number of siblings/spouses aboard: "))

Parch = int(input("Enter number of parents/children aboard: "))

Fare = float(input("Enter Fare: "))

Embarked = input("Enter Embarked (C, Q, or S): ")

# Convert categorical variables to numerical

Sex = 1 if Sex == 'male' else 0

Embarked\_C = 1 if Embarked == 'C' else 0

Embarked\_Q = 1 if Embarked == 'Q' else 0

Embarked\_S = 1 if Embarked == 'S' else 0

# Create input array

input\_data = np.array([[Pclass, Sex, Age, SibSp, Parch, Fare, Embarked\_C, Embarked\_Q, Embarked\_S]])

print("Pclass:", Pclass)

print("Sex:", Sex)

print("Age:", Age)

print("Siblings/Spouses:", SibSp)

print("Parents/Children:", Parch)

print("Fare:", Fare)

print("Embarked\_C:", Embarked\_C)

print("Embarked\_Q:", Embarked\_Q)

print("Embarked\_S:", Embarked\_S)

print("\n")

return input\_data

# Load models from the specified folder

folder\_path = '../models' # Replace with the path to your models folder

models = load\_models\_from\_folder(folder\_path)

# Get input data

input\_data = get\_titanic\_input()

survive\_dict = {

0: "Not Survived",

1: "Survived"

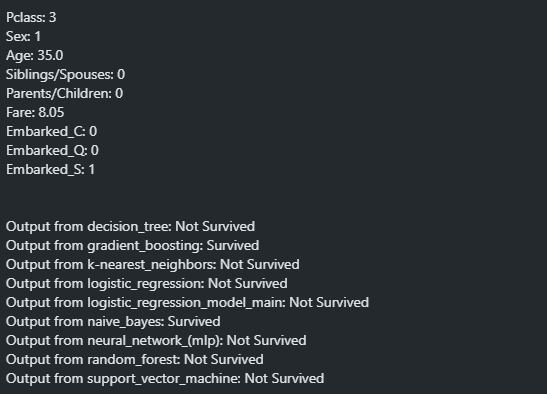
}

# Predict using each model and print the outputs

for model\_name, model in models.items():

output = model.predict(input\_data)

print(f'Output from {model\_name}: {survive\_dict[output[0]]}')



#Using same dataset on different ML Algorithms

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

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

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

#Defining different models to a Dictionary

models = {

"Support Vector Machine": SVC(kernel='linear', random\_state=0),

"K-Nearest Neighbors": KNeighborsClassifier(),

"Decision Tree": DecisionTreeClassifier(random\_state=0),

"Logistic Regression": LogisticRegression(random\_state=0),

"Random Forest": RandomForestClassifier(random\_state=0),

"Gradient Boosting": GradientBoostingClassifier(random\_state=0),

"Neural Network (MLP)": MLPClassifier(random\_state=0),

"Naive Bayes": GaussianNB(),

}

#Storing the accuracies of every model

results = []

for model\_name, model in models.items():

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

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

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"{model\_name} Accuracy: {accuracy \* 100:.2f}%")

results.append([model\_name, accuracy])

file\_name = model\_name.lower().replace(" ", "\_") + ".pkl"

joblib.dump(model, f"../models/{file\_name}")

# Create a DataFrame for the results

results\_df = pd.DataFrame(results, columns=["Model", "Accuracy"])

#Ploting different models accuracy

plt.figure(figsize=(12, 6)) # Adjust figure size as needed

ax = sns.barplot(x="Model", y="Accuracy", data=results\_df)

plt.title("Comparison of Model Accuracies")

plt.xlabel("Model")

plt.ylabel("Accuracy")

plt.ylim(0, 1) # Set y-axis limits to 0-1 for accuracy

plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for better readability

# Add text labels on the bars

for p in ax.patches:

ax.annotate(f'{p.get\_height():.2f}',

(p.get\_x() + p.get\_width() / 2., p.get\_height()),

ha='center', va='center',

xytext=(0, 9),

textcoords='offset points')

plt.tight\_layout()

plt.show()

