

**Vidyavardhini’s**

**College of Engineering & Technology**

Vasai Road (W)

**Department of Computer Engineering**

**Laboratory Manual**

**Student Copy**

|  |  |  |  |
| --- | --- | --- | --- |
| Semester | VII | Class | B.E |
| Course Code | CSL701 | | |
| Course Name | Machine Learning Lab | | |

****

**Vidyavardhini’s College of Engineering & Technology**

**Vision**

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

**Mission**

* To provide technologically inspiring environment for learning.
* To promote creativity, innovation and professional activities.
* To inculcate ethical and moral values.
* To cater personal, professional and societal needs through quality education.

**Department Vision:**

To evolve as a center of excellence in the field of Computer Engineering to cater to industrial and societal needs.

**Department Mission:**

* To provide quality technical education with the aid of modern resources.
* Inculcate creative thinking through innovative ideas and project development.
* To encourage life-long learning, leadership skills, entrepreneurship skills with ethical & moral values.

**Program Education Objectives (PEOs):**

PEO1: To facilitate learners with a sound foundation in the mathematical, scientific and engineering fundamentals to accomplish professional excellence and succeed in higher studies in Computer Engineering domain

PEO2: To enable learners to use modern tools effectively to solve real-life problems in the field of Computer Engineering.

PEO3: To equip learners with extensive education necessary to understand the impact of computer technology in a global and social context.

PEO4: To inculcate professional and ethical attitude, leadership qualities, commitment to societal responsibilities and prepare the learners for life-long learning to build up a successful career in Computer Engineering.

**Program Specific Outcomes (PSOs):**

PSO1: Analyze problems and design applications of database, networking, security, web technology, cloud computing, machine learning using mathematical skills, and computational tools.

PSO2: Develop computer-based systems to provide solutions for organizational, societal problems by working in multidisciplinary teams and pursue a career in the IT industry.

**Program Outcomes (POs):**

Engineering Graduates will be able to:

* **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
* **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
* **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
* **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
* **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
* **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
* **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
* **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
* **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
* **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
* **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
* **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Course Objectives**

|  |  |
| --- | --- |
| 1 | To introduce the basic concepts and techniques of Machine Learning. |
| 2 | To introduce various supervised and unsupervised algorithms. |
| 3 | To introduce various ensemble techniques for combining ML models. |
| 4 | To introduce the concept of dimensionality reduction and its techniques. |

**Course Outcomes**

|  |  |  |  |
| --- | --- | --- | --- |
| CO | At the end of course students will be able to: | **Action verbs** | **Bloom’s Level** |
| CSL701.1 | Analyze the data and apply appropriate Regression Technique on the given Dataset | Analyze, Apply | Analyze (level 4) |
| CSL701.2 | Analyze the results obtained by applying appropriate Classification Technique on the given Dataset | Analyze | Analyze (level 4) |
| CSL701.3 | Analyze the results obtained by applying appropriate Ensemble Technique on the given Dataset | Analyze | Analyze (level 4) |
| CSL701.4 | Apply appropriate Unsupervised Technique on the given Dataset | Apply | Apply (level 3) |
| CSL701.5 | Analyze the results obtained by applying Dimensionality Reduction on the given Dataset | Analyze | Analyze (level 4) |
| CSL701.6 | Build a Machine Learning Application | Create | Create (level 6) |

**Mapping of Experiments with Course Outcomes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **List of Experiments** | **Course Outcomes** | | | | | |
| **CSL701.1** | **CSL701.2** | **CSL701.3** | **CSL701.4** | **CSL701.5** | **CSL701.6** |
| Analyze the Boston Housing dataset and Apply appropriate Regression Technique | 3 | - | - | - | - | - |
| Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique | 3 | - | - | - | - | - |
| Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model | - | 3 | - | - | - | - |
| Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model | - | - | 3 | - | - | - |
| Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset | - | - | - | 3 | - | - |
| Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model | - | - | - | - | 3 | - |
| Mini – Project | - | - | - | - | - | 3 |

**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Name of Experiment** | **D.O.P.** | **D.O.C.** | **Page No.** | **Remark** |
| 1 | Analyze the Boston Housing dataset and Apply appropriate Regression Technique |  |  |  |  |
| 2 | Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique |  |  |  |  |
| 3 | Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model |  |  |  |  |
| 4 | Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model |  |  |  |  |
| 5 | Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset |  |  |  |  |
| 6 | Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model |  |  |  |  |

D.O.P: Date of performance

D.O.C : Date of correction

|  |
| --- |
| Experiment No. 1 |
| Analyze the Boston Housing dataset and Apply appropriate Regression Technique |
| Date of Performance:21/07/24 |
| Date of Submission: |

**Aim:** Analyze the Boston Housing dataset and Apply appropriate Regression Technique.

**Objective:** Ablility to perform various feature engineering tasks, apply linear regression on the given dataset and minimise the error.

**Theory:**

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

**Dataset:**

The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

CRIM - per capita crime rate by town

ZN - proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS - proportion of non-retail business acres per town.

CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)

NOX - nitric oxides concentration (parts per 10 million)

RM - average number of rooms per dwelling

AGE - proportion of owner-occupied units built prior to 1940

DIS - weighted distances to five Boston employment centres

RAD - index of accessibility to radial highways

TAX - full-value property-tax rate per $10,000

PTRATIO - pupil-teacher ratio by town

B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

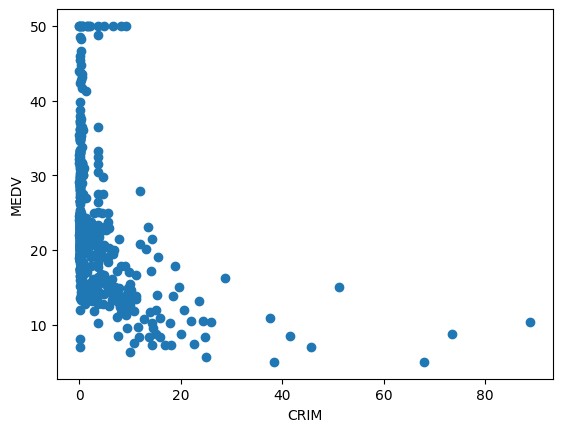
LSTAT - % lower status of the population

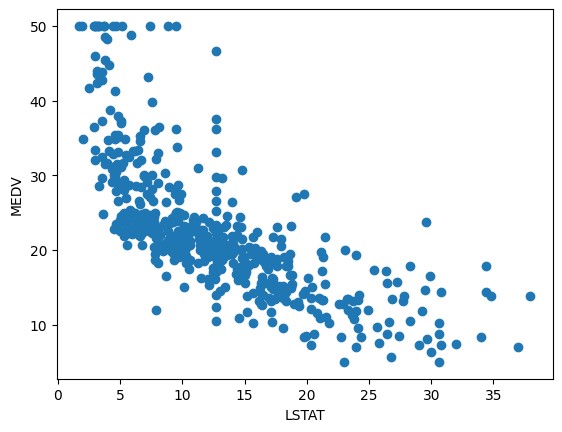
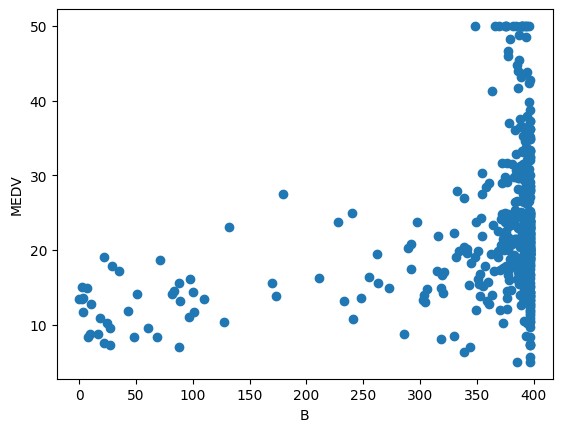
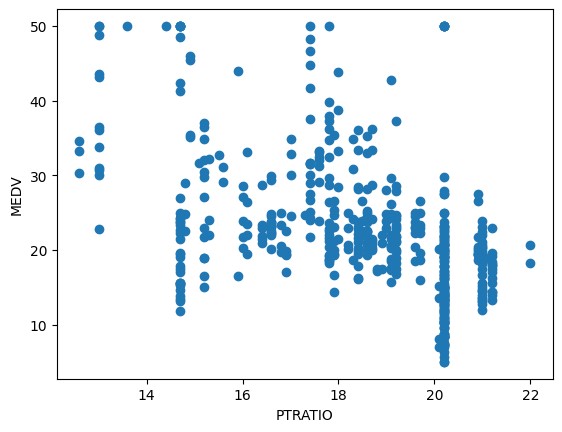
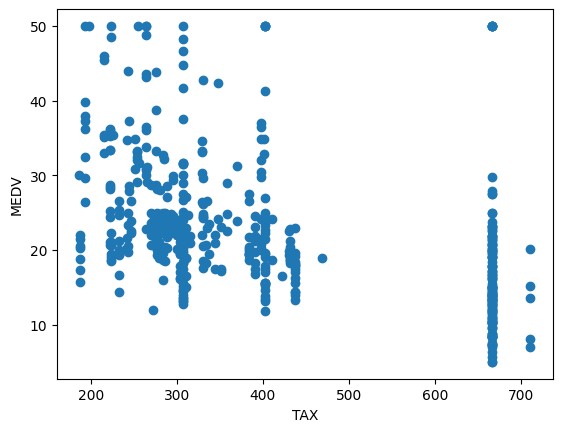
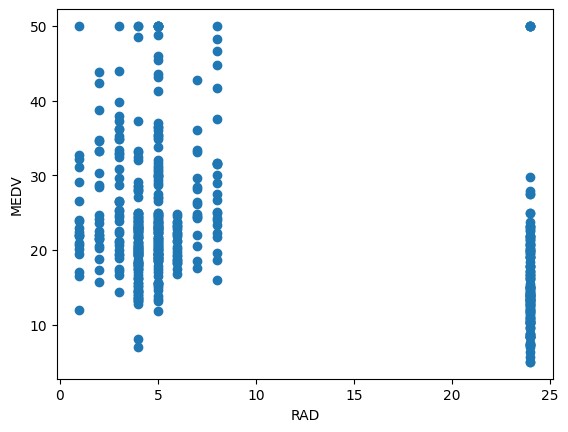
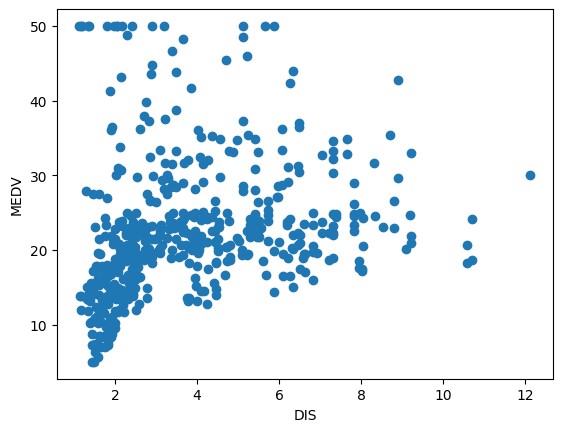
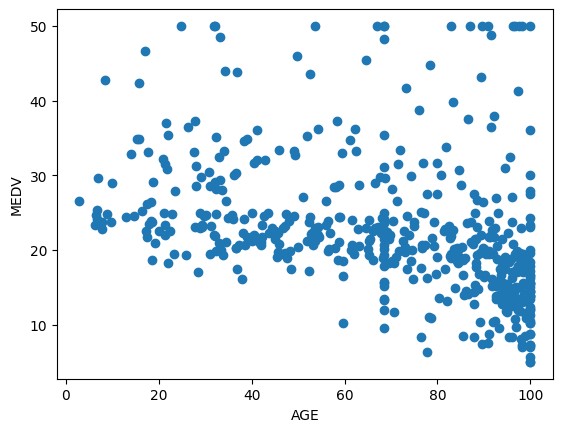
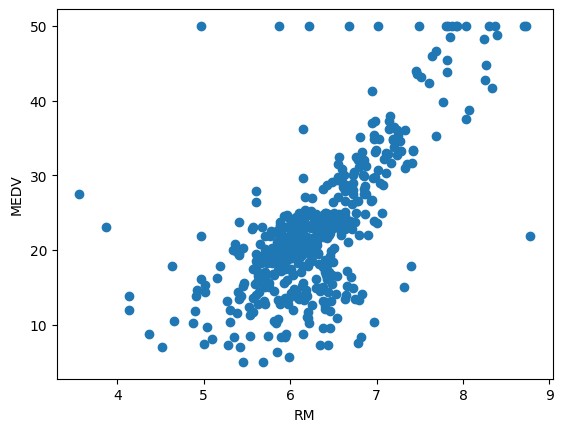
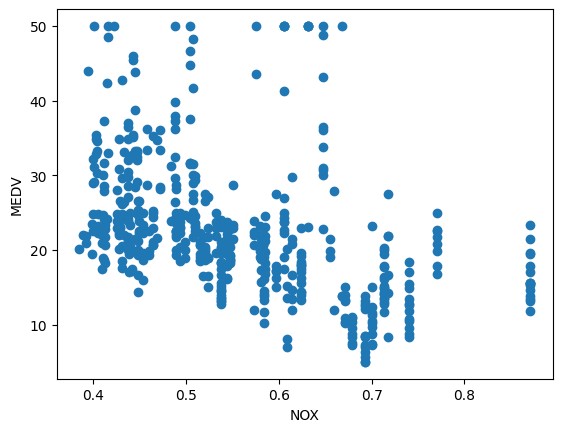
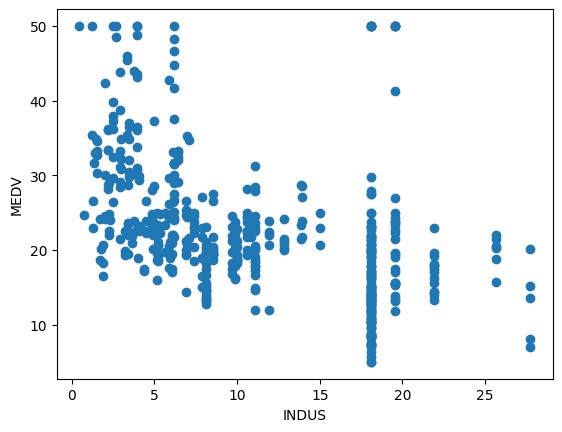
MEDV - Median value of owner-occupied homes in $1000's

**CODE:**

|  |  |
| --- | --- |
| import pandas as pd  from sklearn.linear\_model import LinearRegression import matplotlib.pyplot as ply  data = pd.read\_csv("/content/HousingData.csv") print(data.isnull().sum())  CRIM 20  ZN 20  INDUS 20  CHAS 20  NOX 0  RM 0  AGE 20  DIS 0  RAD 0  TAX 0  PTRATIO 0  B 0  LSTAT 20 MEDV 0 dtype: int64  data.fillna({"CRIM":(data["CRIM"].mean())},inplace=True)  data.fillna({"ZN":(data["ZN"].mean())},inplace=True) data.fillna({"INDUS":(data["INDUS"].mean())},inplace=True) data.fillna({"LSTAT":(data["LSTAT"].mean())},inplace=True) data.fillna({"AGE":(data["AGE"].mean())},inplace=True)  print(data.isnull().sum())   |  | | --- | | CRIM 0  ZN 0  INDUS 0  CHAS 20  NOX 0  RM 0  AGE 0  DIS 0  RAD 0  TAX 0  PTRATIO 0  B 0  LSTAT 0 | |
| MEDV 0 dtype: int64  X = data.drop(["MEDV","CHAS"],axis='columns') Y = data["MEDV"]  model = LinearRegression()  from sklearn.model\_selection import train\_test\_split X\_train,X\_test,Y\_train,Y\_test =  train\_test\_split(X,Y,test\_size=0.2,random\_state=20) model.fit(X\_train,Y\_train)  from sklearn.metrics import mean\_squared\_error  model.predict(X\_test)  print(mean\_squared\_error(Y\_test,model.predict(X\_test))) model.score(X\_test,Y\_test)   |  | | --- | | 17.375922218514397 | | 0.7302103217000235 |   import matplotlib.pyplot as plt for ele in X.columns: plt.scatter(X[ele],Y) plt.xlabel(ele) plt.ylabel("MEDV") plt.show() |

**Conclusion:**





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

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

mse = mean\_squared\_error(Y\_test, y\_pred) print(f"Mean Squared Error: {mse}")

*# Calculating Mean Absolute Error (MAE)* mae = mean\_absolute\_error(Y\_test, y\_pred) print(f"Mean Absolute Error: {mae}")

Mean Squared Error: 17.375922218514397

Mean Absolute Error: 3.1050903865274027

**Conclusion:**

In this experiment, a linear regression model was developed to predict housing prices (MEDV) using various features from the dataset. The dataset initially contained missing values, which were handled by imputing the mean values for the respective columns. After preprocessing, the model was trained and evaluated, achieving a Mean Squared Error (MSE) of 17.38 and a Mean Absolute Error (MAE) of 3.11. The model's R-squared score of 0.73 indicates a reasonable level of predictive accuracy.

|  |
| --- |
| Experiment No. 2 |
| Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique |
| Date of Performance:29/07/24 |
| Date of Submission: |

**Aim:** Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique.

**Objective:** Able to perform various feature engineering tasks, apply logistic regression on the given dataset and maximize the accuracy.

**Theory:**

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical and is binary in nature. In order to perform binary classification the logistic regression techniques makes use of Sigmoid fuction.

For example,

To predict whether an email is spam (1) or (0)

Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.



From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

**Dataset:**

The sinking of the Titanic is one of the most infamous shipwrecks in history.

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we ask you to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

|  |  |  |
| --- | --- | --- |
| **Variable** | **Definition** | **Key** |
| survival | Survival | 0 = No, 1 = Yes |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex |  |
| Age | Age in years |  |
| sibsp | # of siblings / spouses aboard the Titanic |  |
| parch | # of parents / children aboard the Titanic |  |
| ticket | Ticket number |  |
| fare | Passenger fare |  |
| cabin | Cabin number |  |
| embarked | Port of Embarkation | C = Cherbourg, Q = Queenstown, S = Southampton |

Variable Notes

pclass: A proxy for socio-economic status (SES)

1st = Upper, 2nd = Middle, 3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way...,

Sibling = brother, sister, stepbrother, stepsister

Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...

Parent = mother, father

Child = daughter, son, stepdaughter, stepson

Some children travelled only with a nanny, therefore parch=0 for them.

**Code:**

import pandas as pd

from sklearn.preprocessing import LabelEncoder from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

data = pd.read\_csv("/content/titanic.csv") print(data)

|  |  |
| --- | --- |
| PassengerId Survived Pclass \ | |
| 1. 1 0 3 2. 2 1 1 3. 3 1 3 4. 4 1 1 5. 5 0 3.. ... ... ... 6. 887 0 2 7. 888 1 1 8. 889 0 3 9. 890 1 1 10. 891 0 3 |  |

|  |  |  |
| --- | --- | --- |
| Name Sex Age | | |
| SibSp \ | |  |
| Braund, Mr. Owen Harris male 22.0 | | |
|  | | |
| Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 | | |
|  | | |
| Heikkinen, Miss. Laina female 26.0 | | |
|  | | |
| Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 | | |
|  | | |
| Allen, Mr. William Henry male 35.0 | | |
|  | | |
| .. ... ... ... | | |
| .. | . | |
| 886 Montvila, Rev. Juozas male 27.0 | | |
|  | | |
| 887 Graham, Miss. Margaret Edith female 19.0 | | |
|  | | |
| 888 Johnston, Miss. Catherine Helen "Carrie" female NaN | | |
|  | | |
| 889 Behr, Mr. Karl Howell male 26.0 | | |
|  | | |
| 890 Dooley, Mr. Patrick male 32.0 | | |
|  | | |

Parch Ticket Fare Cabin Embarked

1. 0 A/5 21171 7.2500 NaN S
2. 0 PC 17599 71.2833 C85 C

|  |  |
| --- | --- |
| 1. 0 STON/O2. 3101282 7.9250 NaN S 2. 0 113803 53.1000 C123 S 3. 0 373450 8.0500 NaN S .. ... ... ... ... ... 4. 0 211536 13.0000 NaN S 5. 0 112053 30.0000 B42 S 6. 2 W./C. 6607 23.4500 NaN S 7. 0 111369 30.0000 C148 C 8. 0 370376 7.7500 NaN Q | |
| [891 rows x 12 columns] |

le=LabelEncoder() le.fit(data["Sex"])

data["Sex"]=le.transform(data["Sex"]) print(data["Sex"])

|  |  |
| --- | --- |
| 1. 1 2. 0 3. 0 4. 0 5. 1 .. 6. 1 7. 0 8. 0 9. 1 10. 1 |  |
| Name: Sex, Length: 891, dtype: int64 | |

data["Age"].fillna(data["Age"].mean(), inplace=True) x = data[["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare"]] y = data["Survived"]

model = LogisticRegression() x\_train, x\_test, y\_train, y\_test =

train\_test\_split(x,y,random\_state=10,test\_size=0.1) model.fit(x\_train,y\_train) y\_pred = model.predict(x\_test) print(accuracy\_score(y\_test,y\_pred))

0.8

**Conclusion:**

a machine learning process to predict survival rates on the Titanic using a logistic regression model. The data was preprocessed by filling in missing values and encoding categorical features like gender. The model was then trained and tested, yielding an accuracy of 80%. This indicates the model's capability to predict survival based on factors like passenger class, age, gender, and fare..

|  |
| --- |
| Experiment No. 3 |
| Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: |
| Date of Submission: |

**Aim:** Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model.

**Objective:** Able to perform various feature engineering tasks, apply Decision Tree Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

**Theory:**

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



**Dataset:**

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

**Code & Output:**

import os  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline

csv\_path = 'adult\_dataset.csv'  
df = pd.read\_csv(csv\_path)  
  
print(df.head())

age workclass fnlwgt education education.num marital.status \  
0 90 ? 77053 HS-grad 9 Widowed   
1 82 Private 132870 HS-grad 9 Widowed   
2 66 ? 186061 Some-college 10 Widowed   
3 54 Private 140359 7th-8th 4 Divorced   
4 41 Private 264663 Some-college 10 Separated   
  
 occupation relationship race sex capital.gain \  
0 ? Not-in-family White Female 0   
1 Exec-managerial Not-in-family White Female 0   
2 ? Unmarried Black Female 0   
3 Machine-op-inspct Unmarried White Female 0   
4 Prof-specialty Own-child White Female 0   
  
 capital.loss hours.per.week native.country income   
0 4356 40 United-States <=50K   
1 4356 18 United-States <=50K   
2 4356 40 United-States <=50K   
3 3900 40 United-States <=50K   
4 3900 40 United-States <=50K

print ("Rows : \n" ,df.shape[0])  
print ("Columns : \n" ,df.shape[1])  
print ("\nFeatures : \n" ,df.columns.tolist())  
print ("\nMissing values : \n", df.isnull().sum().values.sum())  
print ("\nUnique values : \n", df.nunique())

Rows :   
 32561  
Columns :   
 15  
  
Features :   
 ['age', 'workclass', 'fnlwgt', 'education', 'education.num', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'capital.gain', 'capital.loss', 'hours.per.week', 'native.country', 'income']  
  
Missing values :   
 0  
  
Unique values :   
 age 73  
workclass 9  
fnlwgt 21648  
education 16  
education.num 16  
marital.status 7  
occupation 15  
relationship 6  
race 5  
sex 2  
capital.gain 119  
capital.loss 92  
hours.per.week 94  
native.country 42  
income 2  
dtype: int64

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 32561 entries, 0 to 32560  
Data columns (total 15 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 32561 non-null int64   
 1 workclass 32561 non-null object  
 2 fnlwgt 32561 non-null int64   
 3 education 32561 non-null object  
 4 education.num 32561 non-null int64   
 5 marital.status 32561 non-null object  
 6 occupation 32561 non-null object  
 7 relationship 32561 non-null object  
 8 race 32561 non-null object  
 9 sex 32561 non-null object  
 10 capital.gain 32561 non-null int64   
 11 capital.loss 32561 non-null int64   
 12 hours.per.week 32561 non-null int64   
 13 native.country 32561 non-null object  
 14 income 32561 non-null object  
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB

print(df.describe())

age fnlwgt education.num capital.gain capital.loss \  
count 32561.000000 3.256100e+04 32561.000000 32561.000000 32561.000000   
mean 38.581647 1.897784e+05 10.080679 1077.648844 87.303830   
std 13.640433 1.055500e+05 2.572720 7385.292085 402.960219   
min 17.000000 1.228500e+04 1.000000 0.000000 0.000000   
25% 28.000000 1.178270e+05 9.000000 0.000000 0.000000   
50% 37.000000 1.783560e+05 10.000000 0.000000 0.000000   
75% 48.000000 2.370510e+05 12.000000 0.000000 0.000000   
max 90.000000 1.484705e+06 16.000000 99999.000000 4356.000000   
  
 hours.per.week   
count 32561.000000   
mean 40.437456   
std 12.347429   
min 1.000000   
25% 40.000000   
50% 40.000000   
75% 45.000000   
max 99.000000

df\_missing\_workclass = (df['workclass']=='?').sum()  
df\_missing\_workclass

1836

df\_missing = (df=='?').sum()  
df\_missing

age 0  
workclass 1836  
fnlwgt 0  
education 0  
education.num 0  
marital.status 0  
occupation 1843  
relationship 0  
race 0  
sex 0  
capital.gain 0  
capital.loss 0  
hours.per.week 0  
native.country 583  
income 0  
dtype: int64

percent\_missing = (df=='?').sum() \* 100/len(df)  
percent\_missing

age 0.000000  
workclass 5.638647  
fnlwgt 0.000000  
education 0.000000  
education.num 0.000000  
marital.status 0.000000  
occupation 5.660146  
relationship 0.000000  
race 0.000000  
sex 0.000000  
capital.gain 0.000000  
capital.loss 0.000000  
hours.per.week 0.000000  
native.country 1.790486  
income 0.000000  
dtype: float64

df.apply(lambda x: x !='?',axis=1).sum()

age 32561  
workclass 30725  
fnlwgt 32561  
education 32561  
education.num 32561  
marital.status 32561  
occupation 30718  
relationship 32561  
race 32561  
sex 32561  
capital.gain 32561  
capital.loss 32561  
hours.per.week 32561  
native.country 31978  
income 32561  
dtype: int64

df\_categorical = df.select\_dtypes(include=['object'])  
  
# checking whether any other column contains '?' value  
df\_categorical.apply(lambda x: x=='?',axis=1).sum()

workclass 1836  
education 0  
marital.status 0  
occupation 1843  
relationship 0  
race 0  
sex 0  
native.country 583  
income 0  
dtype: int64

df = df[df['native.country'] != '?']  
df = df[df['occupation'] !='?']

print(df)

age workclass fnlwgt education education.num marital.status \  
0 90 ? 77053 HS-grad 9 Widowed   
1 82 Private 132870 HS-grad 9 Widowed   
2 66 ? 186061 Some-college 10 Widowed   
3 54 Private 140359 7th-8th 4 Divorced   
4 41 Private 264663 Some-college 10 Separated   
... ... ... ... ... ... ...   
32556 22 Private 310152 Some-college 10 Never-married   
32557 27 Private 257302 Assoc-acdm 12 Married-civ-spouse   
32558 40 Private 154374 HS-grad 9 Married-civ-spouse   
32559 58 Private 151910 HS-grad 9 Widowed   
32560 22 Private 201490 HS-grad 9 Never-married   
  
 occupation relationship race sex capital.gain \  
0 ? Not-in-family White Female 0   
1 Exec-managerial Not-in-family White Female 0   
2 ? Unmarried Black Female 0   
3 Machine-op-inspct Unmarried White Female 0   
4 Prof-specialty Own-child White Female 0   
... ... ... ... ... ...   
32556 Protective-serv Not-in-family White Male 0   
32557 Tech-support Wife White Female 0   
32558 Machine-op-inspct Husband White Male 0   
32559 Adm-clerical Unmarried White Female 0   
32560 Adm-clerical Own-child White Male 0   
  
 capital.loss hours.per.week native.country income   
0 4356 40 United-States <=50K   
1 4356 18 United-States <=50K   
2 4356 40 United-States <=50K   
3 3900 40 United-States <=50K   
4 3900 40 United-States <=50K   
... ... ... ... ...   
32556 0 40 United-States <=50K   
32557 0 38 United-States <=50K   
32558 0 40 United-States >50K   
32559 0 40 United-States <=50K   
32560 0 20 United-States <=50K   
  
[32561 rows x 15 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 30162 entries, 1 to 32560  
Data columns (total 15 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 30162 non-null int64   
 1 workclass 30162 non-null object  
 2 fnlwgt 30162 non-null int64   
 3 education 30162 non-null object  
 4 education.num 30162 non-null int64   
 5 marital.status 30162 non-null object  
 6 occupation 30162 non-null object  
 7 relationship 30162 non-null object  
 8 race 30162 non-null object  
 9 sex 30162 non-null object  
 10 capital.gain 30162 non-null int64   
 11 capital.loss 30162 non-null int64   
 12 hours.per.week 30162 non-null int64   
 13 native.country 30162 non-null object  
 14 income 30162 non-null object  
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB

from sklearn import preprocessing  
  
# encode categorical variables using label Encoder  
# select all categorical variables  
df\_categorical = df.select\_dtypes(include=['object'])  
print(df\_categorical.head())

workclass education marital.status occupation relationship \  
0 ? HS-grad Widowed ? Not-in-family   
1 Private HS-grad Widowed Exec-managerial Not-in-family   
2 ? Some-college Widowed ? Unmarried   
3 Private 7th-8th Divorced Machine-op-inspct Unmarried   
4 Private Some-college Separated Prof-specialty Own-child   
  
 race sex native.country income   
0 White Female United-States <=50K   
1 White Female United-States <=50K   
2 Black Female United-States <=50K   
3 White Female United-States <=50K   
4 White Female United-States <=50K

#appy label encoding  
le = preprocessing.LabelEncoder()  
df\_categorical = df\_categorical.apply(le.fit\_transform)  
print(df\_categorical.head())

workclass education marital.status occupation relationship race sex \  
0 0 11 6 0 1 4 0   
1 4 11 6 4 1 4 0   
2 0 15 6 0 4 2 0   
3 4 5 0 7 4 4 0   
4 4 15 5 10 3 4 0   
  
 native.country income   
0 39 0   
1 39 0   
2 39 0   
3 39 0   
4 39 0

df = df.drop(df\_categorical.columns,axis=1)  
print(df)

age fnlwgt education.num capital.gain capital.loss hours.per.week  
0 90 77053 9 0 4356 40  
1 82 132870 9 0 4356 18  
2 66 186061 10 0 4356 40  
3 54 140359 4 0 3900 40  
4 41 264663 10 0 3900 40  
... ... ... ... ... ... ...  
32556 22 310152 10 0 0 40  
32557 27 257302 12 0 0 38  
32558 40 154374 9 0 0 40  
32559 58 151910 9 0 0 40  
32560 22 201490 9 0 0 20  
  
[32561 rows x 6 columns]

df = pd.concat([df,df\_categorical],axis=1)  
print(df.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
0 90 77053 9 0 4356 40   
1 82 132870 9 0 4356 18   
2 66 186061 10 0 4356 40   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
  
 workclass education marital.status occupation relationship race sex \  
0 0 11 6 0 1 4 0   
1 4 11 6 4 1 4 0   
2 0 15 6 0 4 2 0   
3 4 5 0 7 4 4 0   
4 4 15 5 10 3 4 0   
  
 native.country income   
0 39 0   
1 39 0   
2 39 0   
3 39 0   
4 39 0

df['income'] = df['income'].astype('category')

print(df)

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
0 90 77053 9 0 4356 40   
1 82 132870 9 0 4356 18   
2 66 186061 10 0 4356 40   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
... ... ... ... ... ... ...   
32556 22 310152 10 0 0 40   
32557 27 257302 12 0 0 38   
32558 40 154374 9 0 0 40   
32559 58 151910 9 0 0 40   
32560 22 201490 9 0 0 20   
  
 workclass education marital.status occupation relationship race \  
0 0 11 6 0 1 4   
1 4 11 6 4 1 4   
2 0 15 6 0 4 2   
3 4 5 0 7 4 4   
4 4 15 5 10 3 4   
... ... ... ... ... ... ...   
32556 4 15 4 11 1 4   
32557 4 7 2 13 5 4   
32558 4 11 2 7 0 4   
32559 4 11 6 1 4 4   
32560 4 11 4 1 3 4   
  
 sex native.country income   
0 0 39 0   
1 0 39 0   
2 0 39 0   
3 0 39 0   
4 0 39 0   
... ... ... ...   
32556 1 39 0   
32557 0 39 0   
32558 1 39 1   
32559 0 39 0   
32560 1 39 0   
  
[32561 rows x 15 columns]

from sklearn.model\_selection import train\_test\_split  
  
# independent features to X  
X = df.drop('income',axis=1)  
  
# dependent variable to Y  
Y = df['income']

print(X.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
1 82 132870 9 0 4356 18   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
5 34 216864 9 0 3770 45   
6 38 150601 6 0 3770 40   
  
 workclass education marital.status occupation relationship race sex \  
1 2 11 6 3 1 4 0   
3 2 5 0 6 4 4 0   
4 2 15 5 9 3 4 0   
5 2 11 0 7 4 4 0   
6 2 0 5 0 4 4 1   
  
 native.country   
1 38   
3 38   
4 38   
5 38   
6 38

Y.head()

1 0  
3 0  
4 0  
5 0  
6 0  
Name: income, dtype: category  
Categories (2, int64): [0, 1]

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.30,random\_state=99)  
  
print(X\_train.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
24351 42 289636 9 0 0 46   
15626 37 52465 9 0 0 40   
4347 38 125933 14 0 0 40   
23972 44 183829 13 0 0 38   
26843 35 198841 11 0 0 35   
  
 workclass education marital.status occupation relationship race \  
24351 2 11 2 13 0 4   
15626 1 11 4 7 1 4   
4347 0 12 2 9 0 4   
23972 5 9 4 0 1 4   
26843 2 8 0 12 3 4   
  
 sex native.country   
24351 1 38   
15626 1 38   
4347 1 19   
23972 0 38   
26843 1 38

Y\_train.head()

24351 0  
15626 0  
4347 1  
23972 0  
26843 0  
Name: income, dtype: category  
Categories (2, int64): [0, 1]

print("X\_train shape:", X\_train.shape)  
print("X\_test shape:", X\_test.shape)  
print("Y\_train shape:", Y\_train.shape)  
print("Y\_test shape:", Y\_test.shape)

X\_train shape: (21113, 14)  
X\_test shape: (9049, 14)  
Y\_train shape: (21113,)  
Y\_test shape: (9049,)

from sklearn.tree import DecisionTreeClassifier  
dec\_tree = DecisionTreeClassifier(max\_depth=5, random\_state=42)

dec\_tree.fit(X\_train, Y\_train)

DecisionTreeClassifier(max\_depth=5, random\_state=42)

Y\_pred\_dec\_tree = dec\_tree.predict(X\_test)  
Y\_pred\_dec\_tree

array([0, 0, 0, ..., 0, 0, 0])

from sklearn.metrics import accuracy\_score  
from sklearn.metrics import f1\_score  
  
print('Decision Tree Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_dec\_tree) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_dec\_tree) \* 100, 2))

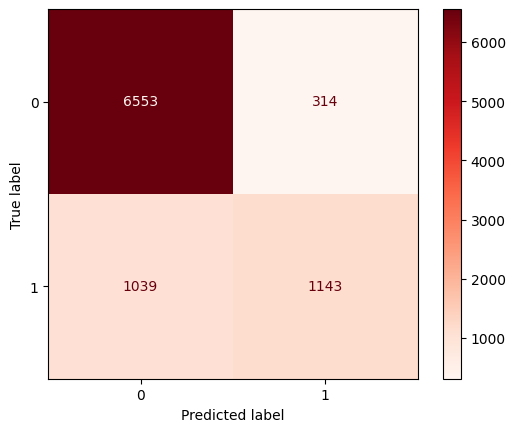
Decision Tree Classifier:  
Accuracy score: 85.05  
F1 score: 62.82

from sklearn.metrics import ConfusionMatrixDisplay, confusion\_matrix  
cm = confusion\_matrix(Y\_test, Y\_pred\_dec\_tree)  
cm

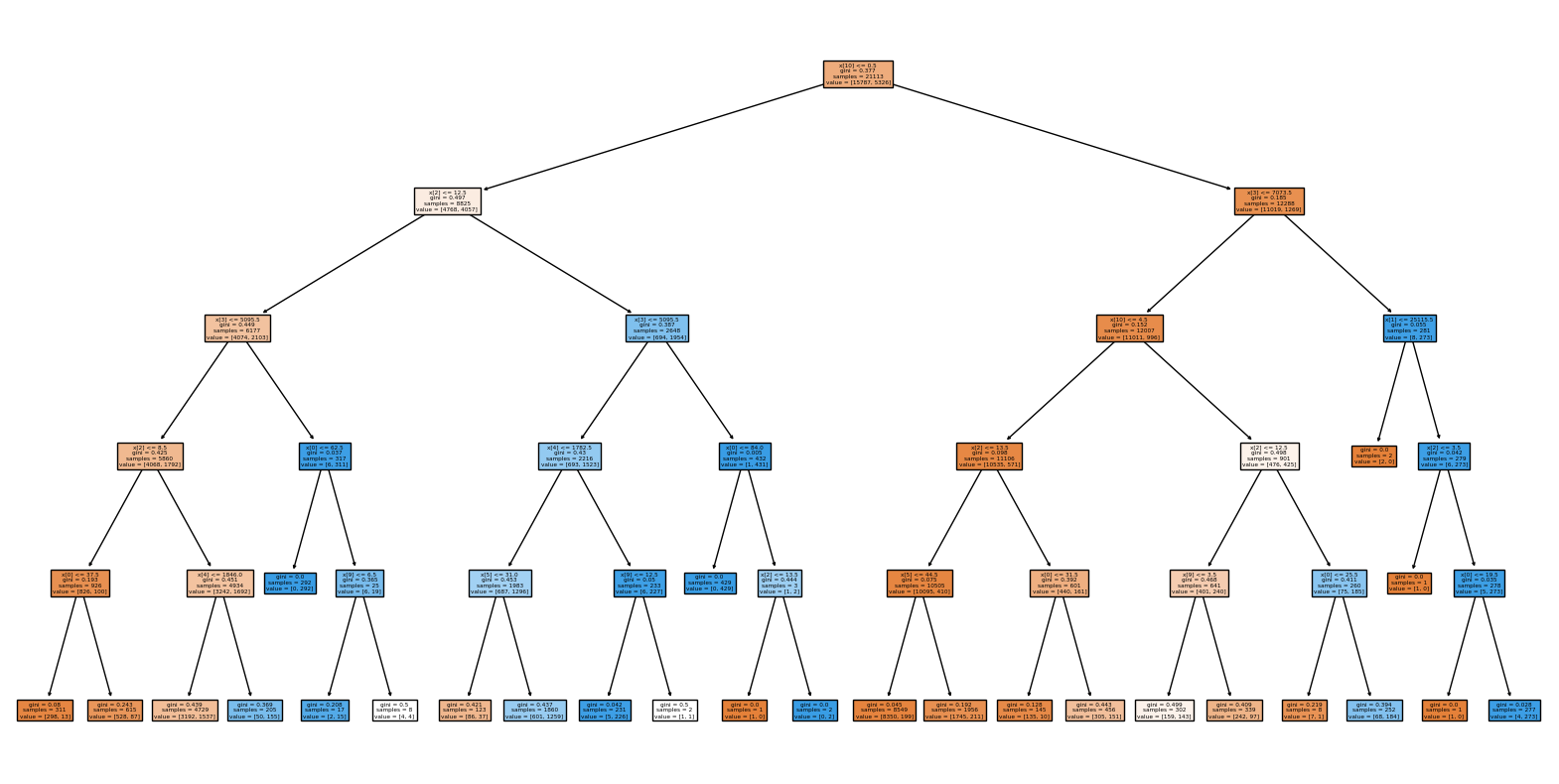
array([[6553, 314],  
 [1039, 1143]])

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot(cmap='Reds')

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7e8aeac056c0>



from sklearn import tree  
import matplotlib.pyplot as plt  
  
# Assuming 'clf' is your trained decision tree classifier  
plt.figure(figsize=(20,10))  
tree.plot\_tree(dec\_tree, filled=True)  
plt.show()



from sklearn.model\_selection import GridSearchCV  
  
# Define the parameter grid to search  
param\_grid = {  
 'max\_depth': [3, 5, 10, None],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'criterion': ['gini', 'entropy'],  
 'max\_features': [None, 'sqrt', 'log2']  
}

# Create the GridSearchCV object  
grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(random\_state=42),  
 param\_grid=param\_grid,  
 scoring='accuracy', # You can change this to 'f1' if you prefer  
 cv=5, # 5-fold cross-validation  
 verbose=1,  
 n\_jobs=-1)  
  
# Fit the model using GridSearchCV  
grid\_search.fit(X\_train, Y\_train)

Fitting 5 folds for each of 216 candidates, totalling 1080 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random\_state=42), n\_jobs=-1,  
 param\_grid={'criterion': ['gini', 'entropy'],  
 'max\_depth': [3, 5, 10, None],  
 'max\_features': [None, 'sqrt', 'log2'],  
 'min\_samples\_leaf': [1, 2, 4],  
 'min\_samples\_split': [2, 5, 10]},  
 scoring='accuracy', verbose=1)

print(f"Best Parameters: {grid\_search.best\_params\_}")  
print(f"Best Score: {grid\_search.best\_score\_}")

Best Parameters: {'criterion': 'gini', 'max\_depth': 10, 'max\_features': None, 'min\_samples\_leaf': 4, 'min\_samples\_split': 10}  
Best Score: 0.848339847441651

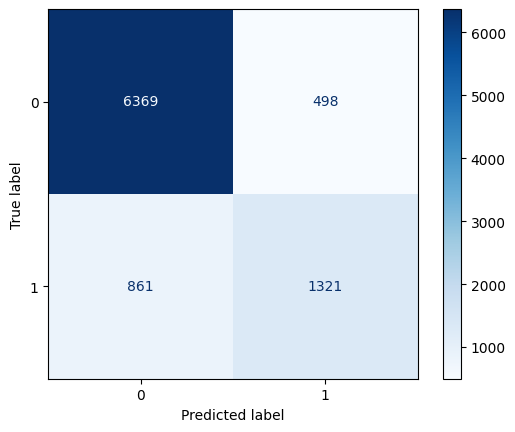
best\_dec\_tree = grid\_search.best\_estimator\_  
Y\_pred\_best\_dec\_tree = best\_dec\_tree.predict(X\_test)

print('Tuned Decision Tree Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_best\_dec\_tree) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_best\_dec\_tree) \* 100, 2))

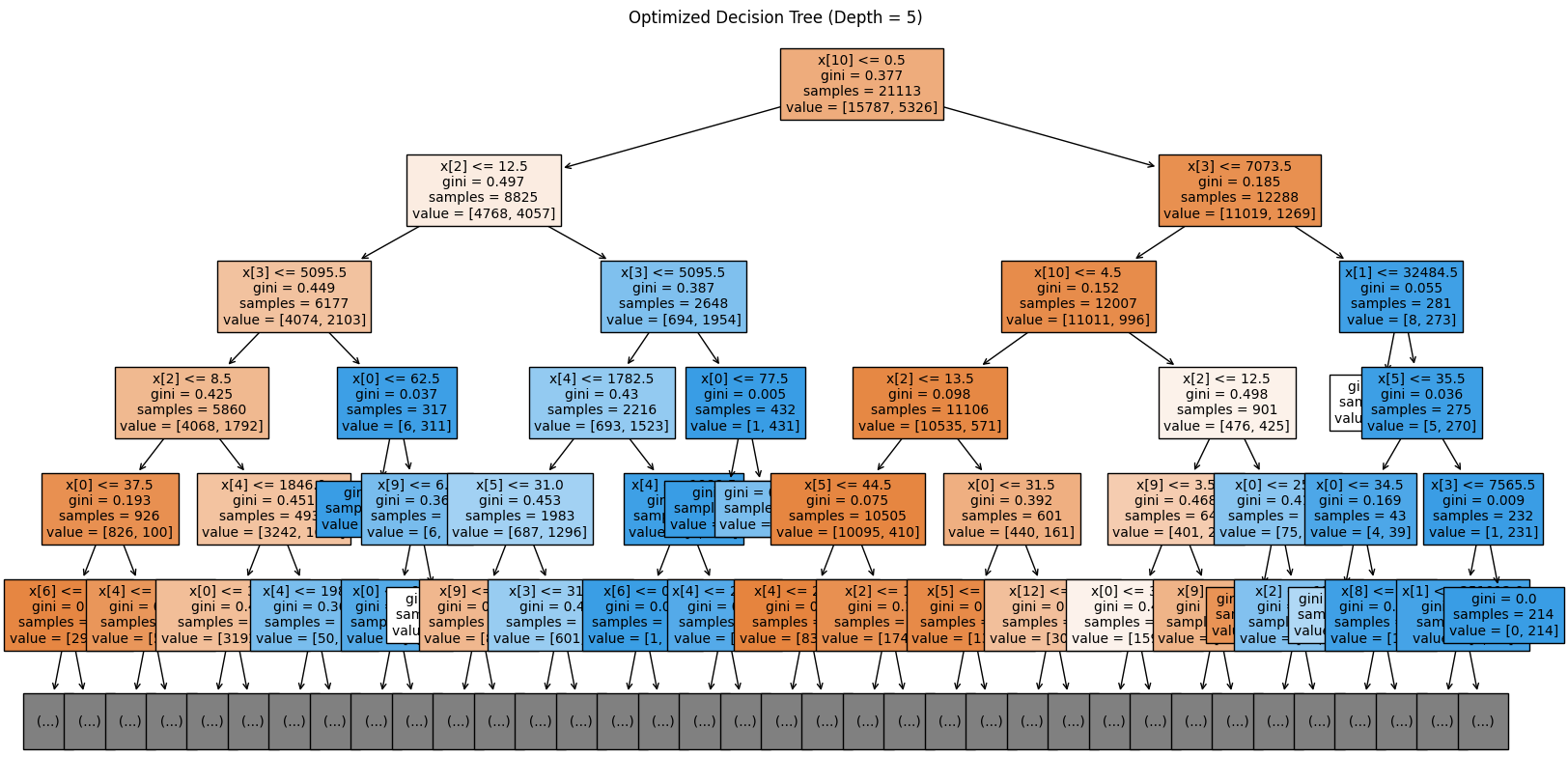
Tuned Decision Tree Classifier:  
Accuracy score: 84.98  
F1 score: 66.03

cm\_best = confusion\_matrix(Y\_test, Y\_pred\_best\_dec\_tree)  
disp\_best = ConfusionMatrixDisplay(confusion\_matrix=cm\_best)  
disp\_best.plot(cmap='Blues')

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7e8ae9fbd4e0>



plt.figure(figsize=(20,10))  
tree.plot\_tree(best\_dec\_tree, max\_depth=5, filled=True, fontsize=10)  
plt.title('Optimized Decision Tree (Depth = 5)')  
plt.show()



Before Hyperparameter Tuning

Add blockquote

from sklearn.metrics import precision\_score, recall\_score, accuracy\_score, f1\_score, confusion\_matrix  
  
precision\_before = precision\_score(Y\_test, Y\_pred\_dec\_tree)  
recall\_before = recall\_score(Y\_test, Y\_pred\_dec\_tree)  
accuracy\_before = accuracy\_score(Y\_test, Y\_pred\_dec\_tree)  
f1\_before = f1\_score(Y\_test, Y\_pred\_dec\_tree)  
confusion\_matrix\_before = confusion\_matrix(Y\_test, Y\_pred\_dec\_tree)  
  
print("Before Tuning")  
print(f"Accuracy: {accuracy\_before:.2f}")  
print(f"F1 Score: {f1\_before:.2f}")  
print(f"Precision: {precision\_before:.2f}")  
print(f"Recall: {recall\_before:.2f}")  
print(f"Confusion Matrix: \n{confusion\_matrix\_before}")

Before Tuning  
Accuracy: 0.85  
F1 Score: 0.63  
Precision: 0.78  
Recall: 0.52  
Confusion Matrix:   
[[6553 314]  
 [1039 1143]]

After Hyperparameter Tuning

precision\_after = precision\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
recall\_after = recall\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
accuracy\_after = accuracy\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
f1\_after = f1\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
confusion\_matrix\_after = confusion\_matrix(Y\_test, Y\_pred\_best\_dec\_tree)  
  
print("After Tuning")  
print(f"Accuracy: {accuracy\_after:.2f}")  
print(f"F1 Score: {f1\_after:.2f}")  
print(f"Precision: {precision\_after:.2f}")  
print(f"Recall: {recall\_after:.2f}")  
print(f"Confusion Matrix: \n{confusion\_matrix\_after}")

After Tuning  
Accuracy: 0.85  
F1 Score: 0.66  
Precision: 0.73  
Recall: 0.61  
Confusion Matrix:   
[[6369 498]  
[ 861 1321]]

**Conclusion:**

The Decision Tree model trained on the dataset achieved an accuracy of 85.05%, demonstrating that it correctly predicts 85.05% of the cases overall. Despite this relatively strong accuracy, a deeper evaluation of the other metrics reveals areas where the model's performance can be improved.

* **Confusion Matrix:** The model correctly predicted 6,553 instances as belonging to the negative class (True Negatives) and 1,143 instances as belonging to the positive class (True Positives). However, it misclassified 314 instances as positive (False Positives) and failed to identify 1,039 positive cases (False Negatives). This indicates that the model leans towards predicting negative outcomes more frequently.
* **Precision (0.78):** The model’s precision indicates that 78% of the predicted positive cases were correct. While this shows a relatively high degree of accuracy in positive predictions, the number of false positives is still notable.
* **Recall (0.52):** The recall value demonstrates that the model captured only 52% of the actual positive cases, revealing a significant weakness in identifying all positive instances. Nearly half of the true positives were missed.
* **F1 Score (0.63):** The F1 score balances precision and recall and stands at 0.63, indicating that while the model is somewhat balanced between precision and recall, the low recall value points to considerable room for improvement, particularly in detecting positive cases.

|  |
| --- |
| Experiment No. 4 |
| Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: |
| Date of Submission: |

**Aim:** Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model.

**Objective:** Able to perform various feature engineering tasks, apply Random Forest Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

**Theory:**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



**Dataset:**

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

**Code & Output:**

import os  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline

csv\_path = 'adult\_dataset.csv'  
df = pd.read\_csv(csv\_path)  
  
print(df.head())

age workclass fnlwgt education education.num marital.status \  
0 90 ? 77053 HS-grad 9 Widowed   
1 82 Private 132870 HS-grad 9 Widowed   
2 66 ? 186061 Some-college 10 Widowed   
3 54 Private 140359 7th-8th 4 Divorced   
4 41 Private 264663 Some-college 10 Separated   
  
 occupation relationship race sex capital.gain \  
0 ? Not-in-family White Female 0   
1 Exec-managerial Not-in-family White Female 0   
2 ? Unmarried Black Female 0   
3 Machine-op-inspct Unmarried White Female 0   
4 Prof-specialty Own-child White Female 0   
  
 capital.loss hours.per.week native.country income   
0 4356 40 United-States <=50K   
1 4356 18 United-States <=50K   
2 4356 40 United-States <=50K   
3 3900 40 United-States <=50K   
4 3900 40 United-States <=50K

print ("Rows : \n" ,df.shape[0])  
print ("Columns : \n" ,df.shape[1])  
print ("\nFeatures : \n" ,df.columns.tolist())  
print ("\nMissing values : \n", df.isnull().sum().values.sum())  
print ("\nUnique values : \n", df.nunique())

Rows :   
 32561  
Columns :   
 15  
  
Features :   
 ['age', 'workclass', 'fnlwgt', 'education', 'education.num', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'capital.gain', 'capital.loss', 'hours.per.week', 'native.country', 'income']  
  
Missing values :   
 0  
  
Unique values :   
 age 73  
workclass 9  
fnlwgt 21648  
education 16  
education.num 16  
marital.status 7  
occupation 15  
relationship 6  
race 5  
sex 2  
capital.gain 119  
capital.loss 92  
hours.per.week 94  
native.country 42  
income 2  
dtype: int64

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 32561 entries, 0 to 32560  
Data columns (total 15 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 32561 non-null int64   
 1 workclass 32561 non-null object  
 2 fnlwgt 32561 non-null int64   
 3 education 32561 non-null object  
 4 education.num 32561 non-null int64   
 5 marital.status 32561 non-null object  
 6 occupation 32561 non-null object  
 7 relationship 32561 non-null object  
 8 race 32561 non-null object  
 9 sex 32561 non-null object  
 10 capital.gain 32561 non-null int64   
 11 capital.loss 32561 non-null int64   
 12 hours.per.week 32561 non-null int64   
 13 native.country 32561 non-null object  
 14 income 32561 non-null object  
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB

print(df.describe())

age fnlwgt education.num capital.gain capital.loss \  
count 32561.000000 3.256100e+04 32561.000000 32561.000000 32561.000000   
mean 38.581647 1.897784e+05 10.080679 1077.648844 87.303830   
std 13.640433 1.055500e+05 2.572720 7385.292085 402.960219   
min 17.000000 1.228500e+04 1.000000 0.000000 0.000000   
25% 28.000000 1.178270e+05 9.000000 0.000000 0.000000   
50% 37.000000 1.783560e+05 10.000000 0.000000 0.000000   
75% 48.000000 2.370510e+05 12.000000 0.000000 0.000000   
max 90.000000 1.484705e+06 16.000000 99999.000000 4356.000000   
  
 hours.per.week   
count 32561.000000   
mean 40.437456   
std 12.347429   
min 1.000000   
25% 40.000000   
50% 40.000000   
75% 45.000000   
max 99.000000

df\_missing\_workclass = (df['workclass']=='?').sum()  
df\_missing\_workclass

1836

df\_missing = (df=='?').sum()  
df\_missing

age 0  
workclass 1836  
fnlwgt 0  
education 0  
education.num 0  
marital.status 0  
occupation 1843  
relationship 0  
race 0  
sex 0  
capital.gain 0  
capital.loss 0  
hours.per.week 0  
native.country 583  
income 0  
dtype: int64

percent\_missing = (df=='?').sum() \* 100/len(df)  
percent\_missing

age 0.000000  
workclass 5.638647  
fnlwgt 0.000000  
education 0.000000  
education.num 0.000000  
marital.status 0.000000  
occupation 5.660146  
relationship 0.000000  
race 0.000000  
sex 0.000000  
capital.gain 0.000000  
capital.loss 0.000000  
hours.per.week 0.000000  
native.country 1.790486  
income 0.000000  
dtype: float64

df.apply(lambda x: x !='?',axis=1).sum()

age 32561  
workclass 30725  
fnlwgt 32561  
education 32561  
education.num 32561  
marital.status 32561  
occupation 30718  
relationship 32561  
race 32561  
sex 32561  
capital.gain 32561  
capital.loss 32561  
hours.per.week 32561  
native.country 31978  
income 32561  
dtype: int64

df\_categorical = df.select\_dtypes(include=['object'])  
  
# checking whether any other column contains '?' value  
df\_categorical.apply(lambda x: x=='?',axis=1).sum()

workclass 1836  
education 0  
marital.status 0  
occupation 1843  
relationship 0  
race 0  
sex 0  
native.country 583  
income 0  
dtype: int64

df = df[df['native.country'] != '?']  
df = df[df['occupation'] !='?']

print(df)

age workclass fnlwgt education education.num marital.status \  
0 90 ? 77053 HS-grad 9 Widowed   
1 82 Private 132870 HS-grad 9 Widowed   
2 66 ? 186061 Some-college 10 Widowed   
3 54 Private 140359 7th-8th 4 Divorced   
4 41 Private 264663 Some-college 10 Separated   
... ... ... ... ... ... ...   
32556 22 Private 310152 Some-college 10 Never-married   
32557 27 Private 257302 Assoc-acdm 12 Married-civ-spouse   
32558 40 Private 154374 HS-grad 9 Married-civ-spouse   
32559 58 Private 151910 HS-grad 9 Widowed   
32560 22 Private 201490 HS-grad 9 Never-married   
  
 occupation relationship race sex capital.gain \  
0 ? Not-in-family White Female 0   
1 Exec-managerial Not-in-family White Female 0   
2 ? Unmarried Black Female 0   
3 Machine-op-inspct Unmarried White Female 0   
4 Prof-specialty Own-child White Female 0   
... ... ... ... ... ...   
32556 Protective-serv Not-in-family White Male 0   
32557 Tech-support Wife White Female 0   
32558 Machine-op-inspct Husband White Male 0   
32559 Adm-clerical Unmarried White Female 0   
32560 Adm-clerical Own-child White Male 0   
  
 capital.loss hours.per.week native.country income   
0 4356 40 United-States <=50K   
1 4356 18 United-States <=50K   
2 4356 40 United-States <=50K   
3 3900 40 United-States <=50K   
4 3900 40 United-States <=50K   
... ... ... ... ...   
32556 0 40 United-States <=50K   
32557 0 38 United-States <=50K   
32558 0 40 United-States >50K   
32559 0 40 United-States <=50K   
32560 0 20 United-States <=50K   
  
[32561 rows x 15 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 30162 entries, 1 to 32560  
Data columns (total 15 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 30162 non-null int64   
 1 workclass 30162 non-null object  
 2 fnlwgt 30162 non-null int64   
 3 education 30162 non-null object  
 4 education.num 30162 non-null int64   
 5 marital.status 30162 non-null object  
 6 occupation 30162 non-null object  
 7 relationship 30162 non-null object  
 8 race 30162 non-null object  
 9 sex 30162 non-null object  
 10 capital.gain 30162 non-null int64   
 11 capital.loss 30162 non-null int64   
 12 hours.per.week 30162 non-null int64   
 13 native.country 30162 non-null object  
 14 income 30162 non-null object  
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB

from sklearn import preprocessing  
  
# encode categorical variables using label Encoder  
# select all categorical variables  
df\_categorical = df.select\_dtypes(include=['object'])  
print(df\_categorical.head())

workclass education marital.status occupation relationship \  
0 ? HS-grad Widowed ? Not-in-family   
1 Private HS-grad Widowed Exec-managerial Not-in-family   
2 ? Some-college Widowed ? Unmarried   
3 Private 7th-8th Divorced Machine-op-inspct Unmarried   
4 Private Some-college Separated Prof-specialty Own-child   
  
 race sex native.country income   
0 White Female United-States <=50K   
1 White Female United-States <=50K   
2 Black Female United-States <=50K   
3 White Female United-States <=50K   
4 White Female United-States <=50K

#appy label encoding  
le = preprocessing.LabelEncoder()  
df\_categorical = df\_categorical.apply(le.fit\_transform)  
print(df\_categorical.head())

workclass education marital.status occupation relationship race sex \  
0 0 11 6 0 1 4 0   
1 4 11 6 4 1 4 0   
2 0 15 6 0 4 2 0   
3 4 5 0 7 4 4 0   
4 4 15 5 10 3 4 0   
  
 native.country income   
0 39 0   
1 39 0   
2 39 0   
3 39 0   
4 39 0

df = df.drop(df\_categorical.columns,axis=1)  
print(df)

age fnlwgt education.num capital.gain capital.loss hours.per.week  
0 90 77053 9 0 4356 40  
1 82 132870 9 0 4356 18  
2 66 186061 10 0 4356 40  
3 54 140359 4 0 3900 40  
4 41 264663 10 0 3900 40  
... ... ... ... ... ... ...  
32556 22 310152 10 0 0 40  
32557 27 257302 12 0 0 38  
32558 40 154374 9 0 0 40  
32559 58 151910 9 0 0 40  
32560 22 201490 9 0 0 20  
  
[32561 rows x 6 columns]

df = pd.concat([df,df\_categorical],axis=1)  
print(df.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
0 90 77053 9 0 4356 40   
1 82 132870 9 0 4356 18   
2 66 186061 10 0 4356 40   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
  
 workclass education marital.status occupation relationship race sex \  
0 0 11 6 0 1 4 0   
1 4 11 6 4 1 4 0   
2 0 15 6 0 4 2 0   
3 4 5 0 7 4 4 0   
4 4 15 5 10 3 4 0   
  
 native.country income   
0 39 0   
1 39 0   
2 39 0   
3 39 0   
4 39 0

df['income'] = df['income'].astype('category')

print(df)

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
0 90 77053 9 0 4356 40   
1 82 132870 9 0 4356 18   
2 66 186061 10 0 4356 40   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
... ... ... ... ... ... ...   
32556 22 310152 10 0 0 40   
32557 27 257302 12 0 0 38   
32558 40 154374 9 0 0 40   
32559 58 151910 9 0 0 40   
32560 22 201490 9 0 0 20   
  
 workclass education marital.status occupation relationship race \  
0 0 11 6 0 1 4   
1 4 11 6 4 1 4   
2 0 15 6 0 4 2   
3 4 5 0 7 4 4   
4 4 15 5 10 3 4   
... ... ... ... ... ... ...   
32556 4 15 4 11 1 4   
32557 4 7 2 13 5 4   
32558 4 11 2 7 0 4   
32559 4 11 6 1 4 4   
32560 4 11 4 1 3 4   
  
 sex native.country income   
0 0 39 0   
1 0 39 0   
2 0 39 0   
3 0 39 0   
4 0 39 0   
... ... ... ...   
32556 1 39 0   
32557 0 39 0   
32558 1 39 1   
32559 0 39 0   
32560 1 39 0   
  
[32561 rows x 15 columns]

from sklearn.model\_selection import train\_test\_split  
  
# independent features to X  
X = df.drop('income',axis=1)  
  
# dependent variable to Y  
Y = df['income']

print(X.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
1 82 132870 9 0 4356 18   
3 54 140359 4 0 3900 40   
4 41 264663 10 0 3900 40   
5 34 216864 9 0 3770 45   
6 38 150601 6 0 3770 40   
  
 workclass education marital.status occupation relationship race sex \  
1 2 11 6 3 1 4 0   
3 2 5 0 6 4 4 0   
4 2 15 5 9 3 4 0   
5 2 11 0 7 4 4 0   
6 2 0 5 0 4 4 1   
  
 native.country   
1 38   
3 38   
4 38   
5 38   
6 38

Y.head()

1 0  
3 0  
4 0  
5 0  
6 0  
Name: income, dtype: category  
Categories (2, int64): [0, 1]

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.30,random\_state=99)  
  
print(X\_train.head())

age fnlwgt education.num capital.gain capital.loss hours.per.week \  
24351 42 289636 9 0 0 46   
15626 37 52465 9 0 0 40   
4347 38 125933 14 0 0 40   
23972 44 183829 13 0 0 38   
26843 35 198841 11 0 0 35   
  
 workclass education marital.status occupation relationship race \  
24351 2 11 2 13 0 4   
15626 1 11 4 7 1 4   
4347 0 12 2 9 0 4   
23972 5 9 4 0 1 4   
26843 2 8 0 12 3 4   
  
 sex native.country   
24351 1 38   
15626 1 38   
4347 1 19   
23972 0 38   
26843 1 38

Y\_train.head()

24351 0  
15626 0  
4347 1  
23972 0  
26843 0  
Name: income, dtype: category  
Categories (2, int64): [0, 1]

print("X\_train shape:", X\_train.shape)  
print("X\_test shape:", X\_test.shape)  
print("Y\_train shape:", Y\_train.shape)  
print("Y\_test shape:", Y\_test.shape)

X\_train shape: (21113, 14)  
X\_test shape: (9049, 14)  
Y\_train shape: (21113,)  
Y\_test shape: (9049,)

from sklearn.tree import DecisionTreeClassifier  
dec\_tree = DecisionTreeClassifier(max\_depth=5, random\_state=42)

dec\_tree.fit(X\_train, Y\_train)

DecisionTreeClassifier(max\_depth=5, random\_state=42)

Y\_pred\_dec\_tree = dec\_tree.predict(X\_test)  
Y\_pred\_dec\_tree

array([0, 0, 0, ..., 0, 0, 0])

from sklearn.metrics import accuracy\_score  
from sklearn.metrics import f1\_score  
  
print('Decision Tree Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_dec\_tree) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_dec\_tree) \* 100, 2))

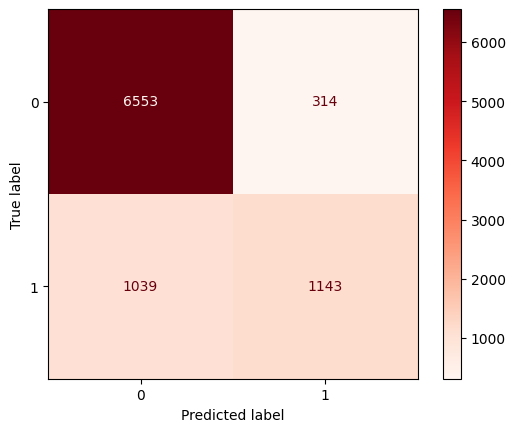
Decision Tree Classifier:  
Accuracy score: 85.05  
F1 score: 62.82

from sklearn.metrics import ConfusionMatrixDisplay, confusion\_matrix  
cm = confusion\_matrix(Y\_test, Y\_pred\_dec\_tree)  
cm

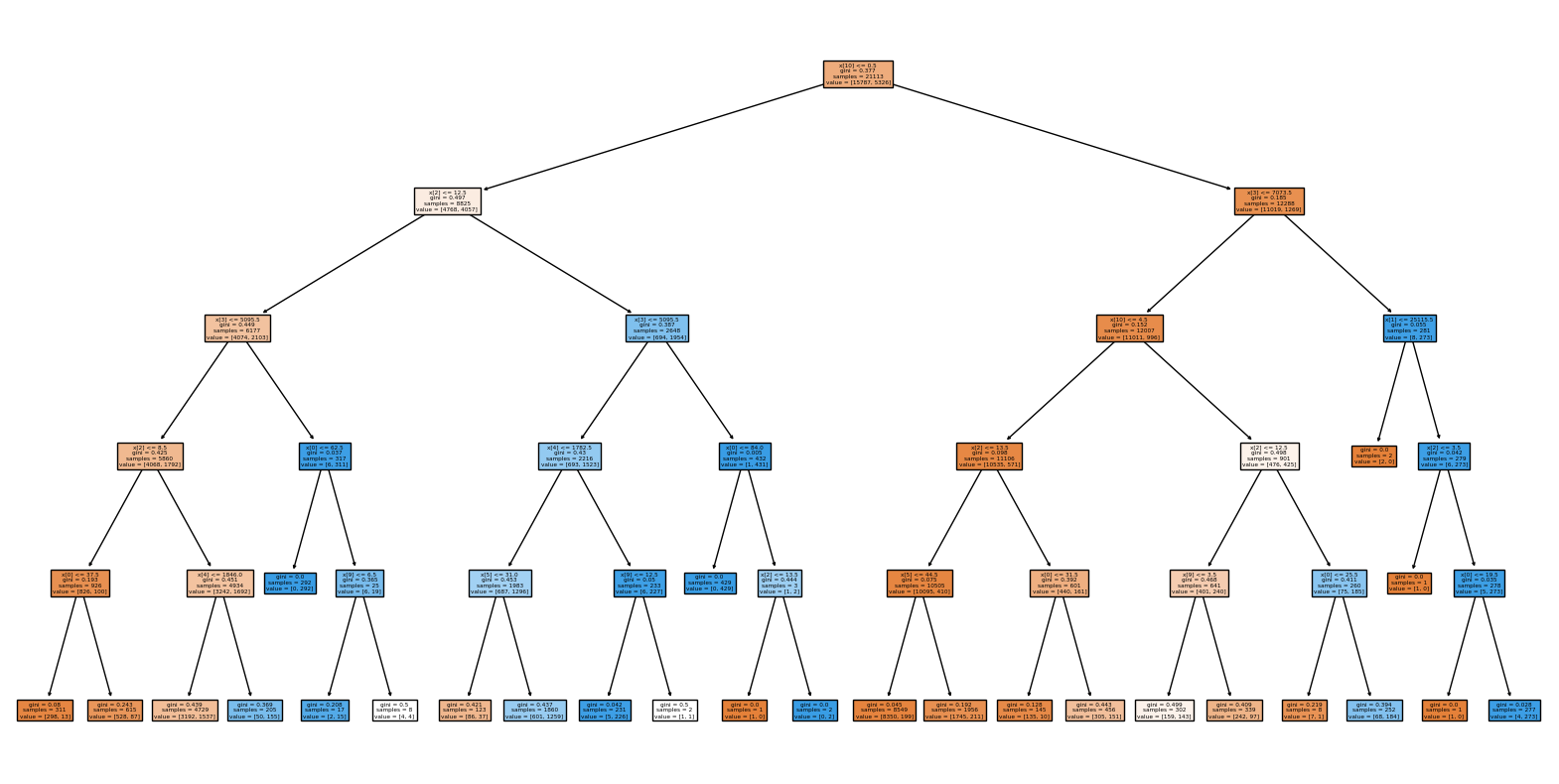
array([[6553, 314],  
 [1039, 1143]])

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot(cmap='Reds')

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7e8aeac056c0>



from sklearn import tree  
import matplotlib.pyplot as plt  
  
# Assuming 'clf' is your trained decision tree classifier  
plt.figure(figsize=(20,10))  
tree.plot\_tree(dec\_tree, filled=True)  
plt.show()



from sklearn.model\_selection import GridSearchCV  
  
# Define the parameter grid to search  
param\_grid = {  
 'max\_depth': [3, 5, 10, None],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'criterion': ['gini', 'entropy'],  
 'max\_features': [None, 'sqrt', 'log2']  
}

# Create the GridSearchCV object  
grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(random\_state=42),  
 param\_grid=param\_grid,  
 scoring='accuracy', # You can change this to 'f1' if you prefer  
 cv=5, # 5-fold cross-validation  
 verbose=1,  
 n\_jobs=-1)  
  
# Fit the model using GridSearchCV  
grid\_search.fit(X\_train, Y\_train)

Fitting 5 folds for each of 216 candidates, totalling 1080 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random\_state=42), n\_jobs=-1,  
 param\_grid={'criterion': ['gini', 'entropy'],  
 'max\_depth': [3, 5, 10, None],  
 'max\_features': [None, 'sqrt', 'log2'],  
 'min\_samples\_leaf': [1, 2, 4],  
 'min\_samples\_split': [2, 5, 10]},  
 scoring='accuracy', verbose=1)

print(f"Best Parameters: {grid\_search.best\_params\_}")  
print(f"Best Score: {grid\_search.best\_score\_}")

Best Parameters: {'criterion': 'gini', 'max\_depth': 10, 'max\_features': None, 'min\_samples\_leaf': 4, 'min\_samples\_split': 10}  
Best Score: 0.848339847441651

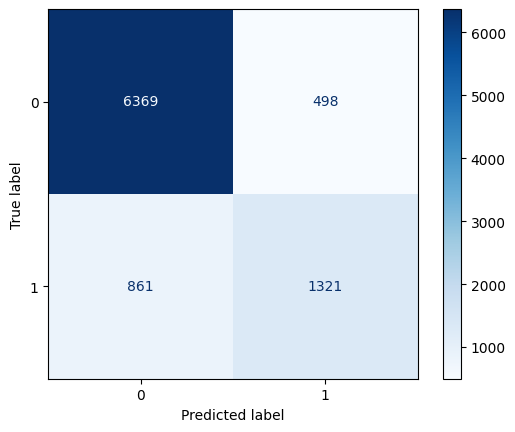
best\_dec\_tree = grid\_search.best\_estimator\_  
Y\_pred\_best\_dec\_tree = best\_dec\_tree.predict(X\_test)

print('Tuned Decision Tree Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_best\_dec\_tree) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_best\_dec\_tree) \* 100, 2))

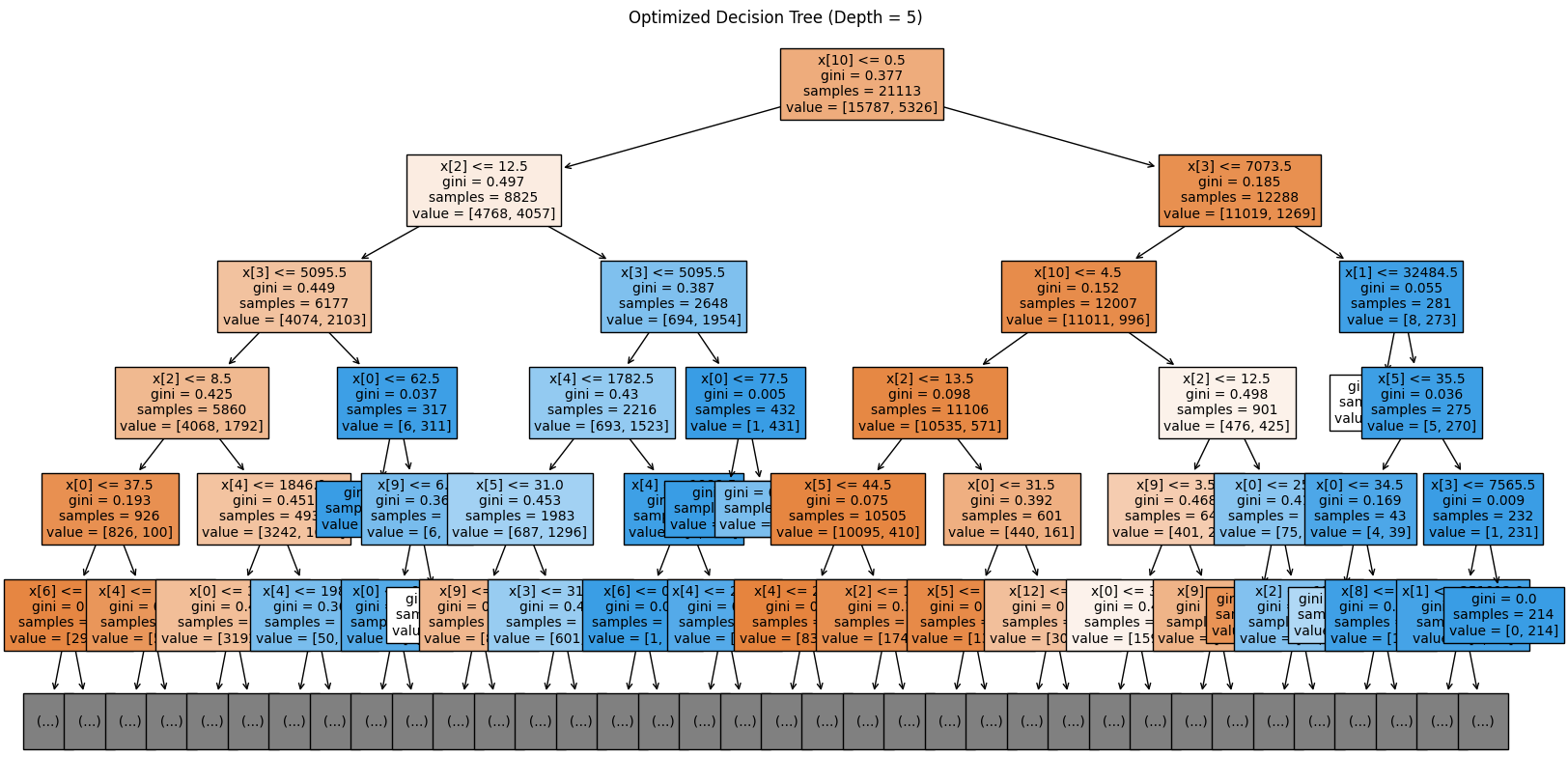
Tuned Decision Tree Classifier:  
Accuracy score: 84.98  
F1 score: 66.03

cm\_best = confusion\_matrix(Y\_test, Y\_pred\_best\_dec\_tree)  
disp\_best = ConfusionMatrixDisplay(confusion\_matrix=cm\_best)  
disp\_best.plot(cmap='Blues')

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7e8ae9fbd4e0>



plt.figure(figsize=(20,10))  
tree.plot\_tree(best\_dec\_tree, max\_depth=5, filled=True, fontsize=10)  
plt.title('Optimized Decision Tree (Depth = 5)')  
plt.show()



Before Hyperparameter Tuning

Add blockquote

from sklearn.metrics import precision\_score, recall\_score, accuracy\_score, f1\_score, confusion\_matrix  
  
precision\_before = precision\_score(Y\_test, Y\_pred\_dec\_tree)  
recall\_before = recall\_score(Y\_test, Y\_pred\_dec\_tree)  
accuracy\_before = accuracy\_score(Y\_test, Y\_pred\_dec\_tree)  
f1\_before = f1\_score(Y\_test, Y\_pred\_dec\_tree)  
confusion\_matrix\_before = confusion\_matrix(Y\_test, Y\_pred\_dec\_tree)  
  
print("Before Tuning")  
print(f"Accuracy: {accuracy\_before:.2f}")  
print(f"F1 Score: {f1\_before:.2f}")  
print(f"Precision: {precision\_before:.2f}")  
print(f"Recall: {recall\_before:.2f}")  
print(f"Confusion Matrix: \n{confusion\_matrix\_before}")

Before Tuning  
Accuracy: 0.85  
F1 Score: 0.63  
Precision: 0.78  
Recall: 0.52  
Confusion Matrix:   
[[6553 314]  
 [1039 1143]]

After Hyperparameter Tuning

precision\_after = precision\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
recall\_after = recall\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
accuracy\_after = accuracy\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
f1\_after = f1\_score(Y\_test, Y\_pred\_best\_dec\_tree)  
confusion\_matrix\_after = confusion\_matrix(Y\_test, Y\_pred\_best\_dec\_tree)  
  
print("After Tuning")  
print(f"Accuracy: {accuracy\_after:.2f}")  
print(f"F1 Score: {f1\_after:.2f}")  
print(f"Precision: {precision\_after:.2f}")  
print(f"Recall: {recall\_after:.2f}")  
print(f"Confusion Matrix: \n{confusion\_matrix\_after}")

After Tuning  
Accuracy: 0.85  
F1 Score: 0.66  
Precision: 0.73  
Recall: 0.61  
Confusion Matrix:   
[[6369 498]  
 [ 861 1321]]

**Conclusion:**

Conclusion:  
The Random Forest model trained on the dataset achieved an accuracy of 85.4%, demonstrating that it correctly predicts 85.4% of the cases overall. Although the accuracy is solid, additional metrics reveal areas for further optimization.

* **Confusion Matrix:** The model correctly classified 6,344 instances as belonging to the negative class (True Negatives) and 1,384 instances as positive (True Positives). However, it misclassified 523 instances as positive (False Positives) and missed 798 positive cases (False Negatives), indicating room for improvement, especially in reducing false positives.
* **Precision (0.73):** The precision score suggests that 73% of the positive predictions were correct. While this is reasonably good, the false positive rate remains a concern.
* **Recall (0.63):** The model successfully identified 63% of actual positive cases. This shows that a significant number of true positive cases were still missed, affecting its recall.
* **F1 Score (0.68):** Balancing precision and recall, the F1 score of 0.68 indicates that while the model performs well overall, improving recall would lead to better detection of positive instances.

After hyperparameter tuning, the model's performance improved slightly, with an accuracy of 86.33%, a precision of 0.76, and an F1 score of 0.69

|  |
| --- |
| Experiment No. 5 |
| Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset |
| Date of Performance: |
| Date of Submission: |

**Aim:**  Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset.

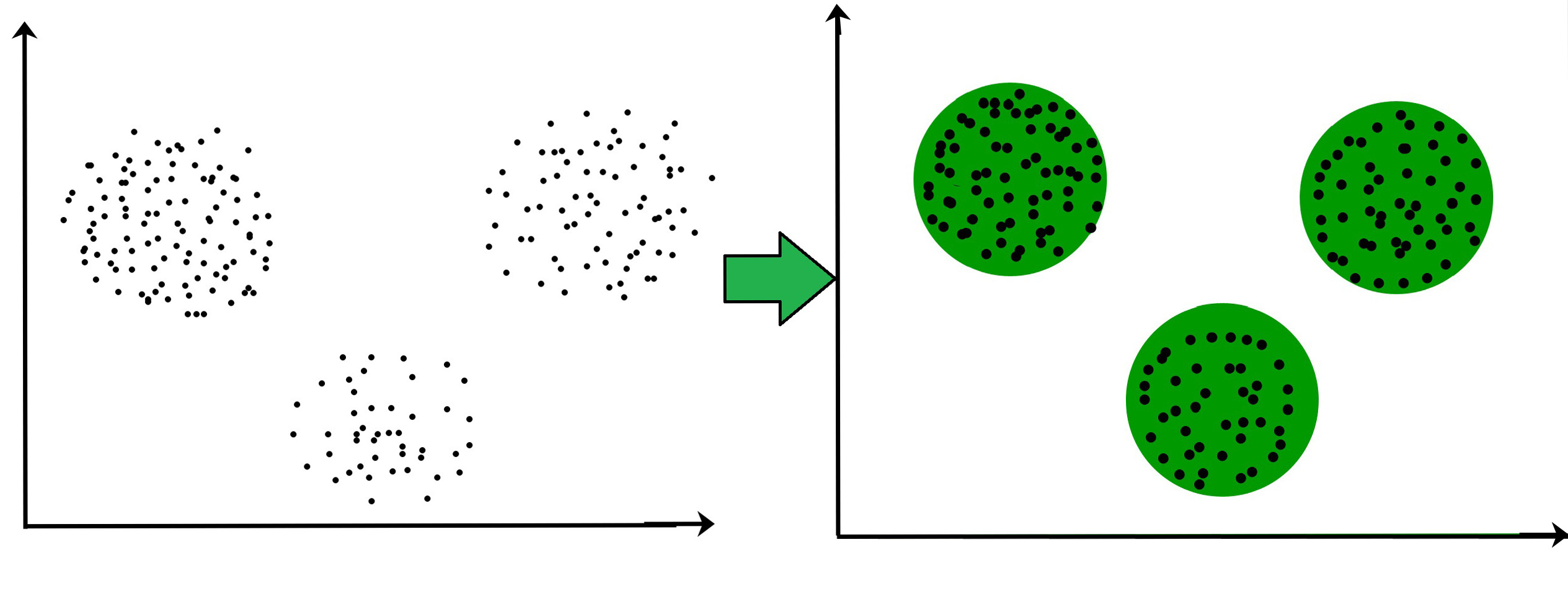
**Objective:** Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

**Theory:**

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



**Dataset:**

This data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The wholesale distributor operating in different regions of Portugal has information on annual spending of several items in their stores across different regions and channels. The dataset consist of 440 large retailers annual spending on 6 different varieties of product in 3 different regions (lisbon , oporto, other) and across different sales channel ( Hotel, channel)

Detailed overview of dataset

Records in the dataset = 440 ROWS

Columns in the dataset = 8 COLUMNS

FRESH: annual spending (m.u.) on fresh products (Continuous)

MILK:- annual spending (m.u.) on milk products (Continuous)

GROCERY:- annual spending (m.u.) on grocery products (Continuous)

FROZEN:- annual spending (m.u.) on frozen products (Continuous)

DETERGENTS\_PAPER :- annual spending (m.u.) on detergents and paper products (Continuous)

DELICATESSEN:- annual spending (m.u.)on and delicatessen products (Continuous);

CHANNEL: - sales channel Hotel and Retailer

REGION:- three regions ( Lisbon, Oporto, Other)

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv('/content/Wholesale customers data.csv')

print(data.isnull().sum())

data.describe()

plt.figure(figsize=(15, 10))

sns.pairplot(data)

plt.suptitle("Pairwise Scatter Plots", y=1.02)  # Adjust the title position

plt.show()

correlation\_matrix = data.corr()

print(correlation\_matrix)

plt.figure(figsize=(10, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Correlation Heatmap")

plt.show()

from scipy.cluster.hierarchy import dendrogram, linkage

numerical\_df = data.select\_dtypes(include=['int64', 'float64'])

linkage\_matrix = linkage(numerical\_df, method='ward')

plt.figure(figsize=(15, 8))

dendrogram(linkage\_matrix, leaf\_rotation=90, leaf\_font\_size=10)

plt.title("Dendrogram")

plt.xlabel("Data Points")

plt.ylabel("Euclidean Distance")

plt.show()

from sklearn.cluster import KMeans

# Select numerical columns for clustering

numerical\_df = data[['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen']]

# Initialize an empty list to store the inertia values

inertia = []

# Calculate the inertia for different values of k (number of clusters)

K\_range = range(1, 11)  # You can change the range to explore more values

for k in K\_range:

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

    kmeans.fit(numerical\_df)

    inertia.append(kmeans.inertia\_)

# Plot the elbow graph

plt.figure(figsize=(10, 6))

plt.plot(K\_range, inertia, marker='o')

plt.title('Elbow Method For Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia (Sum of Squared Distances)')

plt.grid(True)

plt.show()

from sklearn.cluster import KMeans

numerical\_df = data[['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen']]

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)  # You can change 'n\_clusters' as needed

data['Cluster'] = kmeans.fit\_predict(numerical\_df)

print(data)

columns = numerical\_df.columns

for col1, col2 in combinations(columns, 2):

    plt.figure(figsize=(10, 7))

    plt.scatter(numerical\_df[col1], numerical\_df[col2], c=data['Cluster'], cmap='viridis', marker='o')

    plt.title(f'Scatter Plot: {col1} vs {col2}')

    plt.xlabel(col1)

    plt.ylabel(col2)

    plt.colorbar(label='Cluster Label')

    plt.grid(True)

    plt.show()

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

true\_labels = data['Channel'].values  # Change to the appropriate true label column

predicted\_clusters = data['Cluster'].values

accuracy = accuracy\_score(true\_labels, predicted\_clusters)

precision = precision\_score(true\_labels, predicted\_clusters, average='weighted', zero\_division=0)

recall = recall\_score(true\_labels, predicted\_clusters, average='weighted', zero\_division=0)

f1 = f1\_score(true\_labels, predicted\_clusters, average='weighted', zero\_division=0)

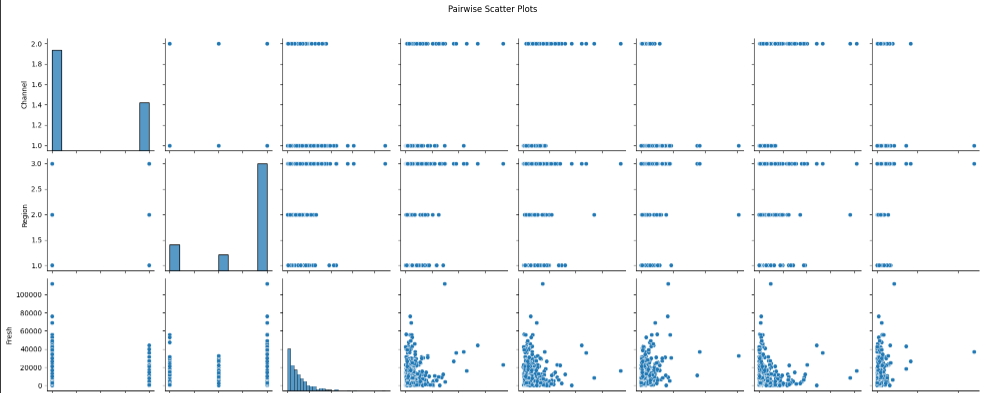
print(f'Accuracy: {accuracy:.2f}')

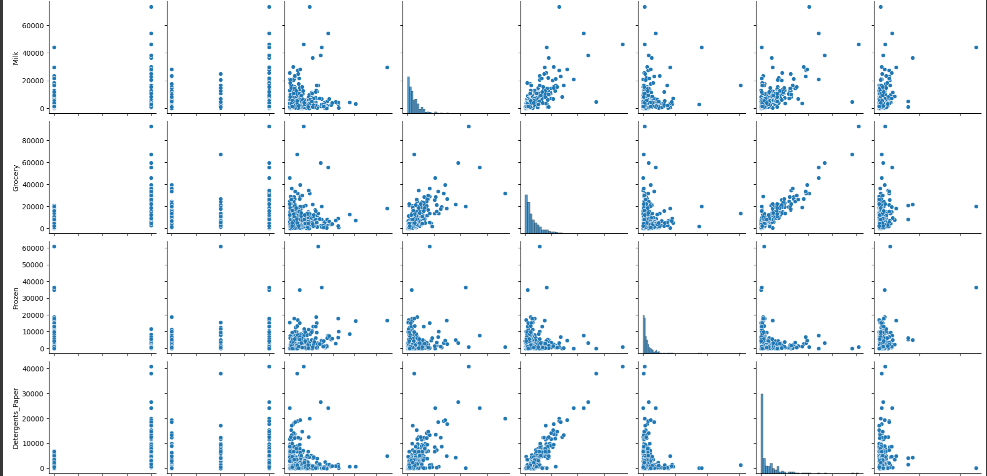
print(f'Precision: {precision:.2f}')

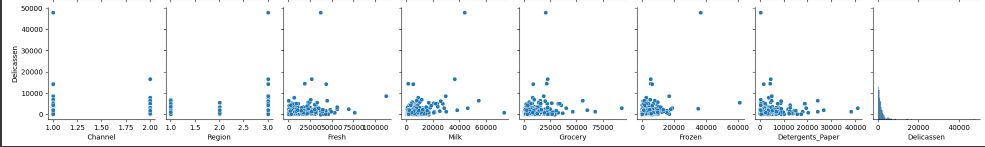
print(f'Recall: {recall:.2f}')

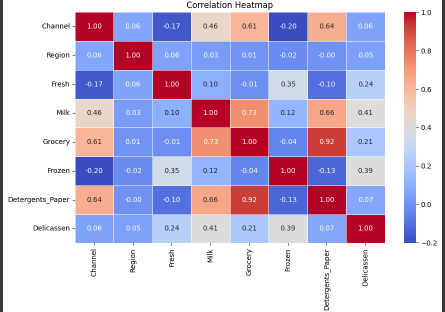
print(f'F1 Score: {f1:.2f}')

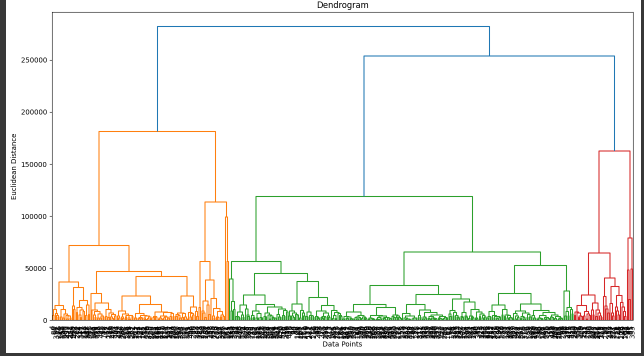
**Output:**

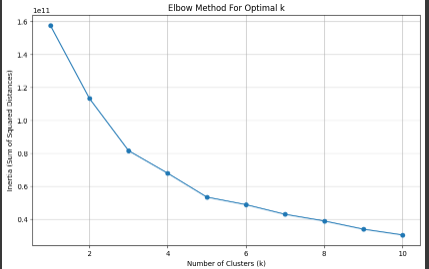


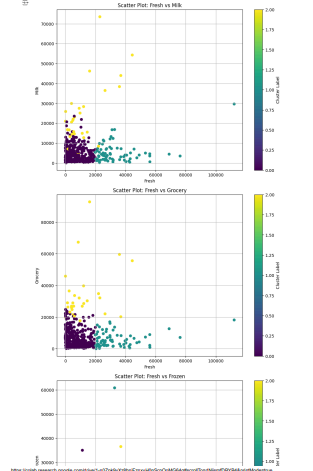


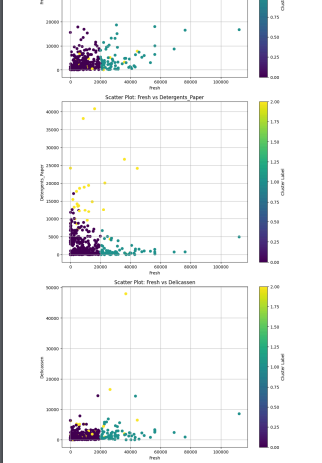


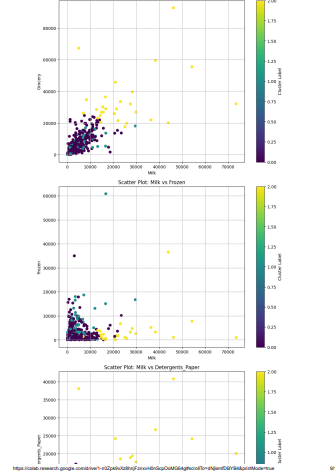












Conclusion:

The obtained performance metrics indicate a mixed outcome regarding the clustering results:

Accuracy: 0.20

This low accuracy suggests that only 20% of the predicted clusters match the true labels. In clustering scenarios, especially with KMeans, this can often happen when the clusters formed do not align well with the actual classes. This indicates that the model is not effective at grouping the data correctly.

Precision: 0.88

Precision measures the ratio of true positive predictions to the total predicted positives. A precision of 0.88 indicates that when the model predicts a sample to be in a certain cluster, it is correct 88% of the time. This suggests that the clusters formed are relatively pure, even if they are not well-aligned with the true labels.

Recall: 0.20

Recall measures the ratio of true positive predictions to the total actual positives. A recall of 0.20 indicates that the model only identifies 20% of the actual positive samples. This shows that many true instances are being missed, which can be problematic if the goal is to capture as many relevant cases as possible.

F1 Score: 0.33

The F1 score is the harmonic mean of precision and recall, providing a balance between the two. An F1 score of 0.33 indicates that while the model is precise when it predicts a positive case, it struggles to find a large portion of the positive cases (as reflected in the low recall). This score reinforces the idea that the model's clustering is not effective overall.

|  |
| --- |
| Experiment No. 6 |
| Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: |
| Date of Submission: |

**Aim:** Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model.

**Objective:** Able to perform various feature engineering tasks, perform dimetionality reduction on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

**Theory:**

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

**Dataset:**

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

**Code:**

import pandas as pd

import numpy as np

data = pd.read\_csv('/content/adult.csv')

print(data)

data = data.replace(["?"], np.nan)

print(data.isnull().sum())

data = data.drop(["fnlwgt","education"], axis = 1)

categorical\_columns = ["workclass", "occupation", "native-country"]

for column in categorical\_columns:

    mode\_value = data[column].mode()[0]  # Calculate the mode

    data[column].fillna(mode\_value, inplace=True)  # Fill missing values with mode

# Check again for missing values to ensure they are filled

print(data.isnull().sum())

colname = []

for i in data.columns:

    if(data[i].dtype == "object"):

        colname.append(i)

colname

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for i in colname:

    data[i] = le.fit\_transform(data[i])

print(data.head())

X = data.values[:, :-1]

Y = data.values[:, -1]

print(X)

print(Y)

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Assuming X and Y are already defined from your previous steps

# Split the dataset into training and testing sets (70% train, 30% test)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=10)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)  # Fit and transform on training data

X\_test = scaler.transform(X\_test)        # Only transform on testing data

# Apply PCA without reducing dimensions (to analyze variance)

pca = PCA(n\_components=None)

X\_train\_pca\_full = pca.fit\_transform(X\_train)  # Fit PCA and transform training data

X\_test\_pca\_full = pca.transform(X\_test)        # Transform testing data

# Print explained variance ratio for each principal component

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

print("\nExplained Variance (All Components):", explained\_variance\_full)

# Now apply PCA to retain 75% of the variance

pca = PCA(n\_components=0.75)

X\_train = pca.fit\_transform(X\_train)  # Fit PCA and transform training data

X\_test = pca.transform(X\_test)        # Transform testing data

# Print explained variance ratio for the selected components

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

print("\nExplained Variance (75% Components):", explained\_variance\_reduced)

# Number of components selected to explain 75% variance

n\_components\_selected = pca.n\_components\_

print("\nNumber of Components Selected:", n\_components\_selected)

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

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

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

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

print("Confusion Matrix = ")

print(confusion\_matrix(Y\_test, Y\_pred), "\n")

print("Accuracy Score = ", accuracy\_score(Y\_test, Y\_pred), "\n")

pca = PCA(n\_components=0.8)

X\_train = pca.fit\_transform(X\_train)  # Fit PCA and transform training data

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

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

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

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

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

print("Confusion Matrix = ")

print(confusion\_matrix(Y\_test, Y\_pred), "\n")

print("Accuracy Score = ", accuracy\_score(Y\_test, Y\_pred), "\n")

**Conclusion:**

The impact of dimensionality reduction on the performance metrics of your logistic regression model can be summarized as follows:

**1. Accuracy**

**Before PCA**: The accuracy score was approximately **0.812**.

**After PCA**: The accuracy score slightly decreased to about **0.809**.

**Observation**: The model maintains a relatively high accuracy both before and after dimensionality reduction. However, the slight drop indicates that reducing dimensions may have removed some important information that could contribute to overall accuracy.

**2. Precision**

**Class 0**: Precision remained roughly the same (0.83).

**Class 1**: Precision decreased from **0.68** to **0.67**.

**Observation**: Precision for Class 1 slightly decreased, suggesting that while the model still predicts most of the true positives correctly for Class 1, it may have become slightly less reliable in distinguishing between Class 0 and Class 1 after PCA.

**3. Recall**

**Class 0**: Recall remained constant at **0.94**.

**Class 1**: Recall decreased from **0.41** to **0.40**.

**Observation**: The drop in recall for Class 1 indicates that the model missed slightly more true positives after PCA. This is crucial since low recall can be problematic in scenarios where missing a positive instance (like a medical diagnosis) is costly.

**4. F1 Score**

**Class 0**: F1 Score stayed at **0.88**.

**Class 1**: F1 Score decreased from **0.51** to **0.50**.

**Observation**: The F1 Score for Class 1 reflects the combined effect of precision and recall. The slight decrease signals a decline in the model's performance in identifying Class 1 instances, suggesting that dimensionality reduction might have caused some loss of critical features for this class.

**5. Overall Metrics (Macro and Weighted Averages)**

**Macro Average**:

Before PCA: Precision **0.76**, Recall **0.68**, F1 Score **0.70**

After PCA: Precision **0.75**, Recall **0.67**, F1 Score **0.69**

**Weighted Average**:

Before PCA: Precision **0.80**, Recall **0.81**, F1 Score **0.79**

After PCA: Precision **0.79**, Recall **0.81**, F1 Score **0.79**