|  |
| --- |
| Experiment No. 1 |
| Analyze the Boston Housing dataset and Apply appropriate Regression Technique |
| Date of Performance: |
| 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 & OUTPUT:**

import pandas as pd  
import numpy as np  
  
file\_path = 'BostonHousing.csv' # Update this if your file path is different  
data = pd.read\_csv(file\_path)

print(data.head())  
  
# Display information about the dataset  
print(data.info())  
  
# Check for missing values  
print(data.isnull().sum())

crim zn indus chas nox rm age dis rad tax ptratio \  
0 0.00632 18.0 2.31 0 0.538 6.575 65.2 4.0900 1 296 15.3   
1 0.02731 0.0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8   
2 0.02729 0.0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8   
3 0.03237 0.0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7   
4 0.06905 0.0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7   
  
 b lstat medv   
0 396.90 4.98 24.0   
1 396.90 9.14 21.6   
2 392.83 4.03 34.7   
3 394.63 2.94 33.4   
4 396.90 5.33 36.2   
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 506 entries, 0 to 505  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 crim 506 non-null float64  
 1 zn 506 non-null float64  
 2 indus 506 non-null float64  
 3 chas 506 non-null int64   
 4 nox 506 non-null float64  
 5 rm 501 non-null float64  
 6 age 506 non-null float64  
 7 dis 506 non-null float64  
 8 rad 506 non-null int64   
 9 tax 506 non-null int64   
 10 ptratio 506 non-null float64  
 11 b 506 non-null float64  
 12 lstat 506 non-null float64  
 13 medv 506 non-null float64  
dtypes: float64(11), int64(3)  
memory usage: 55.5 KB  
None  
crim 0  
zn 0  
indus 0  
chas 0  
nox 0  
rm 5  
age 0  
dis 0  
rad 0  
tax 0  
ptratio 0  
b 0  
lstat 0  
medv 0  
dtype: int64

data = data.dropna()

# Initial linear regression with all parameters  
X\_all = data.drop(columns=['medv'])  
y\_all = data['medv']

import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
X\_train\_all, X\_test\_all, y\_train\_all, y\_test\_all = train\_test\_split(X\_all, y\_all, test\_size=0.2, random\_state=42)  
  
# Initialize the Linear Regression model  
model\_all = LinearRegression()  
  
# Train the model  
model\_all.fit(X\_train\_all, y\_train\_all)

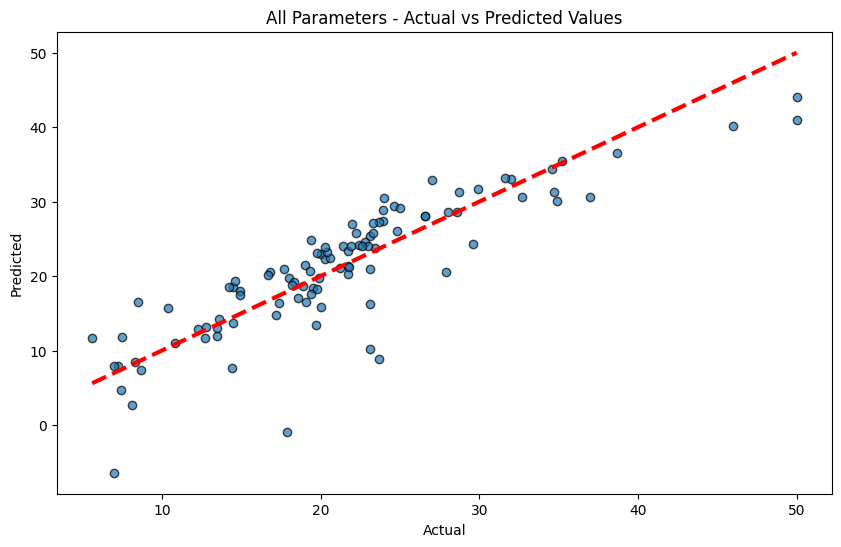
LinearRegression()

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

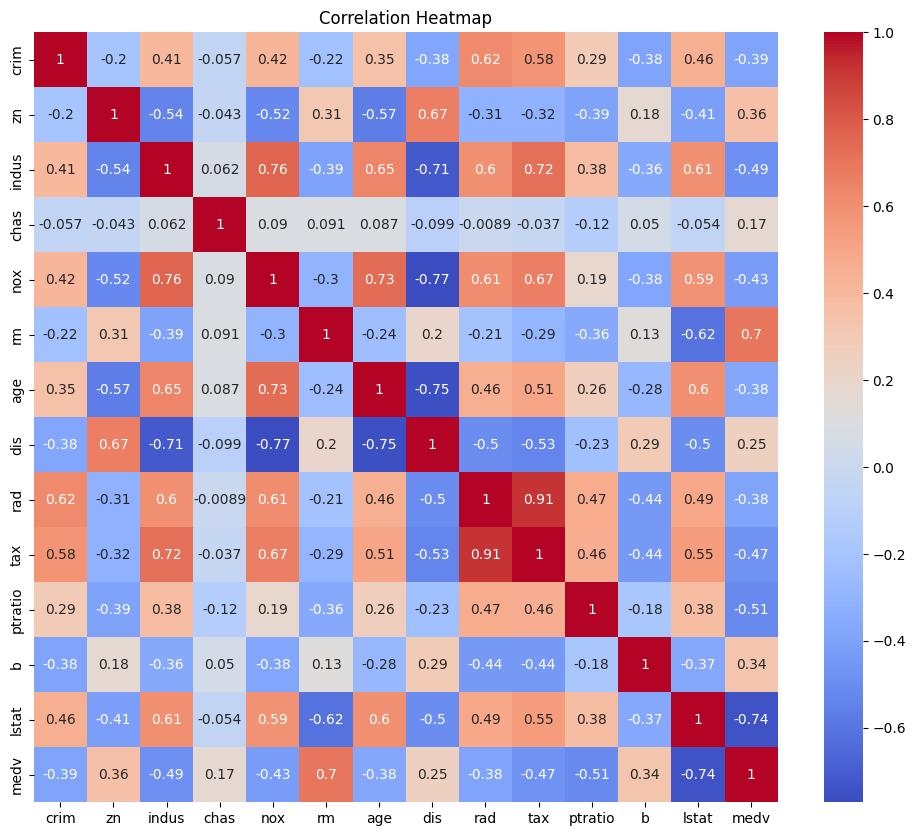
mse\_all = mean\_squared\_error(y\_test\_all, y\_pred\_all)  
rmse\_all = np.sqrt(mse\_all)  
r2\_all = r2\_score(y\_test\_all, y\_pred\_all)  
  
print(f"All Parameters - Mean Squared Error: {mse\_all}")  
print(f"All Parameters - Root Mean Squared Error: {rmse\_all}")  
print(f"All Parameters - R-squared: {r2\_all}")

All Parameters - Mean Squared Error: 20.687720473048476  
All Parameters - Root Mean Squared Error: 4.548375586189917  
All Parameters - R-squared: 0.7200277678580317

# Plotting actual vs predicted values for all parameters  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_test\_all, y\_pred\_all, edgecolor='k', alpha=0.7)  
plt.plot([y\_test\_all.min(), y\_test\_all.max()], [y\_test\_all.min(), y\_test\_all.max()], 'r--', lw=3)  
plt.xlabel('Actual')  
plt.ylabel('Predicted')  
plt.title('All Parameters - Actual vs Predicted Values')  
plt.show()



# Draw a heatmap of correlations  
plt.figure(figsize=(12, 10))  
corr\_matrix = data.corr()  
sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()



relevant\_features = corr\_matrix.index[abs(corr\_matrix["medv"]) > 0.5].tolist()  
relevant\_features.remove('medv')  
print("Selected relevant features:", relevant\_features)

Selected relevant features: ['rm', 'ptratio', 'lstat']

# Linear regression with selected parameters  
X\_relevant = data[relevant\_features]  
y\_relevant = data['medv']

# Split the data into training and testing sets  
X\_train\_rel, X\_test\_rel, y\_train\_rel, y\_test\_rel = train\_test\_split(X\_relevant, y\_relevant, test\_size=0.2, random\_state=42)  
  
# Initialize the Linear Regression model  
model\_rel = LinearRegression()  
  
# Train the model  
model\_rel.fit(X\_train\_rel, y\_train\_rel)

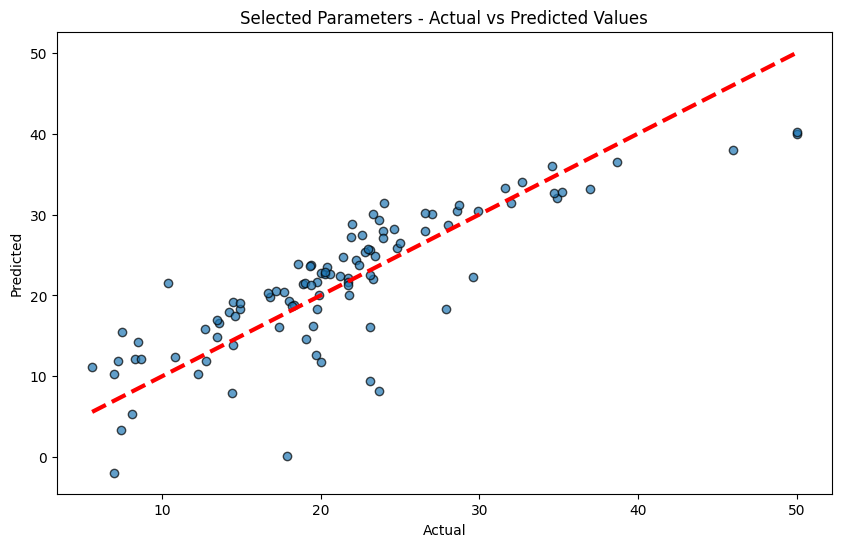
LinearRegression()

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

# Evaluate the model  
mse\_rel = mean\_squared\_error(y\_test\_rel, y\_pred\_rel)  
rmse\_rel = np.sqrt(mse\_rel)  
r2\_rel = r2\_score(y\_test\_rel, y\_pred\_rel)  
  
print(f"Selected Parameters - Mean Squared Error: {mse\_rel}")  
print(f"Selected Parameters - Root Mean Squared Error: {rmse\_rel}")  
print(f"Selected Parameters - R-squared: {r2\_rel}")

Selected Parameters - Mean Squared Error: 24.601921143326106  
Selected Parameters - Root Mean Squared Error: 4.960032373213516  
Selected Parameters - R-squared: 0.6670558853281565

# Plotting actual vs predicted values for selected parameters  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_test\_rel, y\_pred\_rel, edgecolor='k', alpha=0.7)  
plt.plot([y\_test\_rel.min(), y\_test\_rel.max()], [y\_test\_rel.min(), y\_test\_rel.max()], 'r--', lw=3)  
plt.xlabel('Actual')  
plt.ylabel('Predicted')  
plt.title('Selected Parameters - Actual vs Predicted Values')  
plt.show()



# Compare results  
print("\nComparison:")  
print(f"All Parameters - RMSE: {rmse\_all}, R-squared: {r2\_all}")  
print(f"Selected Parameters - RMSE: {rmse\_rel}, R-squared: {r2\_rel}")

Comparison:  
All Parameters - RMSE: 4.548375586189917, R-squared: 0.7200277678580317  
Selected Parameters - RMSE: 4.960032373213516, R-squared: 0.6670558853281565

**Conclusion:**

The Mean Squared Error (MSE) calculated for the model using all features is lower, suggesting that this model better captures the variability in the housing prices. This could be due to the inclusion of additional information, even if some features contribute noise. Conversely, the higher MSE for the model with selected features indicates that important information might have been lost by excluding certain features, leading to less accurate predictions. This highlights the complexity of feature selection, where excluding less correlated features doesn't always result in improved model performance.

|  |
| --- |
| Experiment No. 2 |
| Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique |
| Date of Performance: |
| 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 & OUTPUT:**

import pandas as pd  
  
df = pd.read\_csv('Titanic-Dataset.csv')  
print(df.head())

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S

print(df.info())

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 PassengerId 891 non-null int64   
 1 Survived 891 non-null int64   
 2 Pclass 891 non-null int64   
 3 Name 891 non-null object   
 4 Sex 891 non-null object   
 5 Age 714 non-null float64  
 6 SibSp 891 non-null int64   
 7 Parch 891 non-null int64   
 8 Ticket 891 non-null object   
 9 Fare 891 non-null float64  
 10 Cabin 204 non-null object   
 11 Embarked 889 non-null object   
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB  
None

df = df[['Survived', 'Age', 'Sex', 'Pclass']]  
df = pd.get\_dummies(df, columns=['Sex', 'Pclass'])  
df.dropna(inplace=True)  
print(df.head())

Survived Age Sex\_female Sex\_male Pclass\_1 Pclass\_2 Pclass\_3  
0 0 22.0 False True False False True  
1 1 38.0 True False True False False  
2 1 26.0 True False False False True  
3 1 35.0 True False True False False  
4 0 35.0 False True False False True

print(df)

Survived Age Sex\_female Sex\_male Pclass\_1 Pclass\_2 Pclass\_3  
0 0 22.0 False True False False True  
1 1 38.0 True False True False False  
2 1 26.0 True False False False True  
3 1 35.0 True False True False False  
4 0 35.0 False True False False True  
.. ... ... ... ... ... ... ...  
885 0 39.0 True False False False True  
886 0 27.0 False True False True False  
887 1 19.0 True False True False False  
889 1 26.0 False True True False False  
890 0 32.0 False True False False True  
  
[714 rows x 7 columns]

from sklearn.model\_selection import train\_test\_split  
  
x = df.drop('Survived', axis=1)  
y = df['Survived']  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, stratify=y, random\_state=0)

from sklearn.linear\_model import LogisticRegression  
  
model = LogisticRegression(random\_state=0)  
model.fit(x\_train, y\_train)

LogisticRegression(random\_state=0)

model.score(x\_test, y\_test)

0.8321678321678322

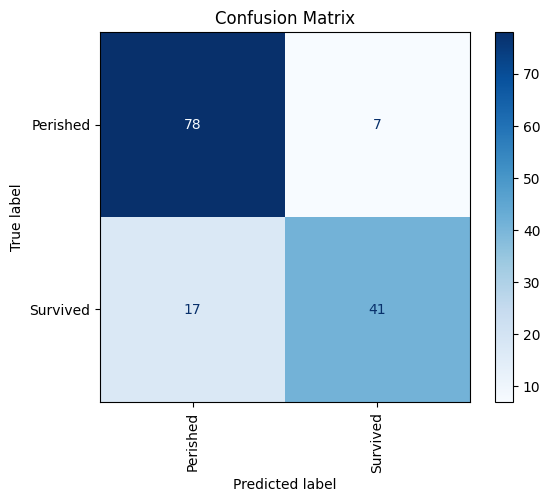
from sklearn.model\_selection import cross\_val\_score  
  
cross\_val\_score(model, x, y, cv=5).mean()

0.7857480547621394

from sklearn.metrics import confusion\_matrix  
  
y\_predicted = model.predict(x\_test)  
confusion\_matrix(y\_test, y\_predicted)

array([[78, 7],  
 [17, 41]])

from sklearn.metrics import ConfusionMatrixDisplay, confusion\_matrix  
import matplotlib.pyplot as plt  
  
y\_pred = model.predict(x\_test)  
  
# Compute the confusion matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Display the confusion matrix  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['Perished', 'Survived'])  
disp.plot(cmap='Blues')  
  
# Optional: customize the plot further  
plt.xticks(rotation='vertical')  
plt.title('Confusion Matrix')  
plt.show()



from sklearn.metrics import classification\_report  
  
print(classification\_report(y\_test, y\_predicted))

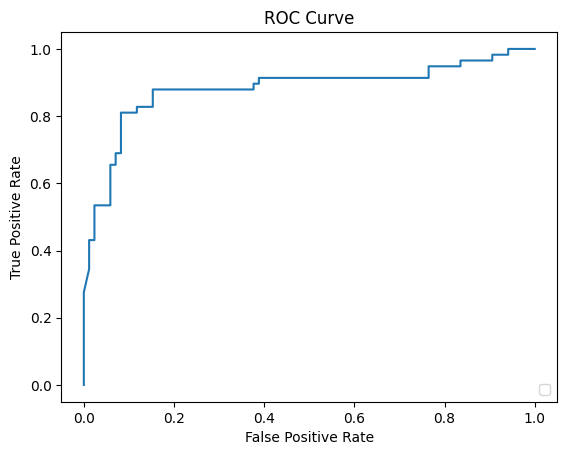
precision recall f1-score support  
  
 0 0.82 0.92 0.87 85  
 1 0.85 0.71 0.77 58  
  
 accuracy 0.83 143  
 macro avg 0.84 0.81 0.82 143  
weighted avg 0.83 0.83 0.83 143

accuracy = model.score(x\_test, y\_test)  
print(f'Accuracy: {accuracy:.2f}')

Accuracy: 0.83

from sklearn.metrics import roc\_curve, RocCurveDisplay  
y\_prob = model.predict\_proba(x\_test)[:,1]  
fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)  
  
# Create the ROC curve display  
disp = RocCurveDisplay(fpr=fpr, tpr=tpr)  
disp.plot()  
  
# Add labels and title if desired  
plt.title('ROC Curve')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
  
plt.show()

WARNING:matplotlib.legend:No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.



**Conclusion:**

The accuracy obtained from the Logistic Regression model on the Titanic dataset provides an overall measure of the model's performance, indicating the proportion of correct predictions out of the total instances. However, accuracy alone can be misleading, especially in the presence of imbalanced classes. For instance, if there are significantly more non-survivors than survivors, a high accuracy might still mean the model predominantly predicts the majority class. Therefore, while a high accuracy suggests good performance, it is essential to also consider other metrics like precision, recall, F1-score, and the ROC curve to comprehensively evaluate the model's ability to correctly predict both survivors and non-survivors. The provided script includes these additional metrics to ensure a more thorough assessment of the model.

|  |
| --- |
| 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))

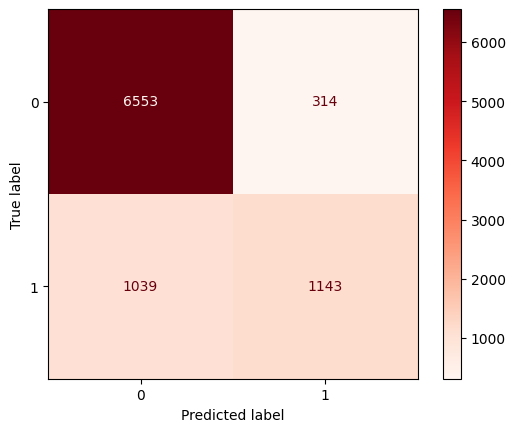
Decision Tree Classifier:  
Accuracy score: 85.05  
F1 score: 62.82

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

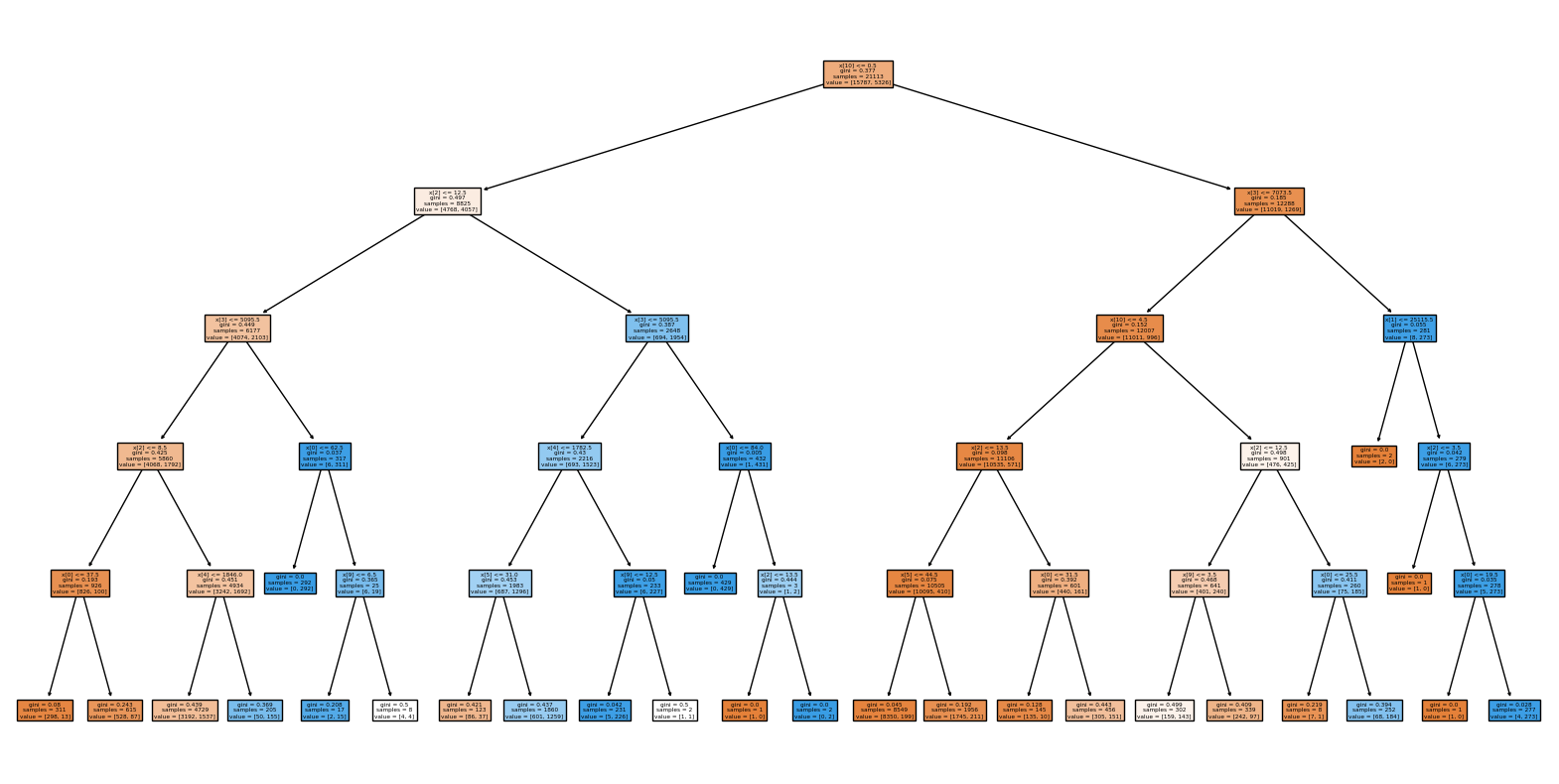
array([[6553, 314],  
 [1039, 1143]])

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

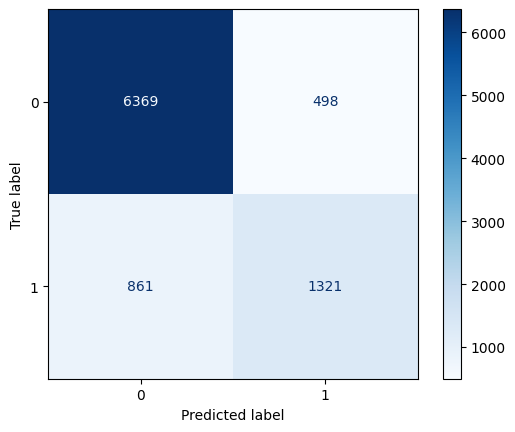
best\_dec\_tree = grid\_search.best\_estimator\_  
Y\_pred\_best\_dec\_tree = best\_dec\_tree.predict(X\_test)

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))

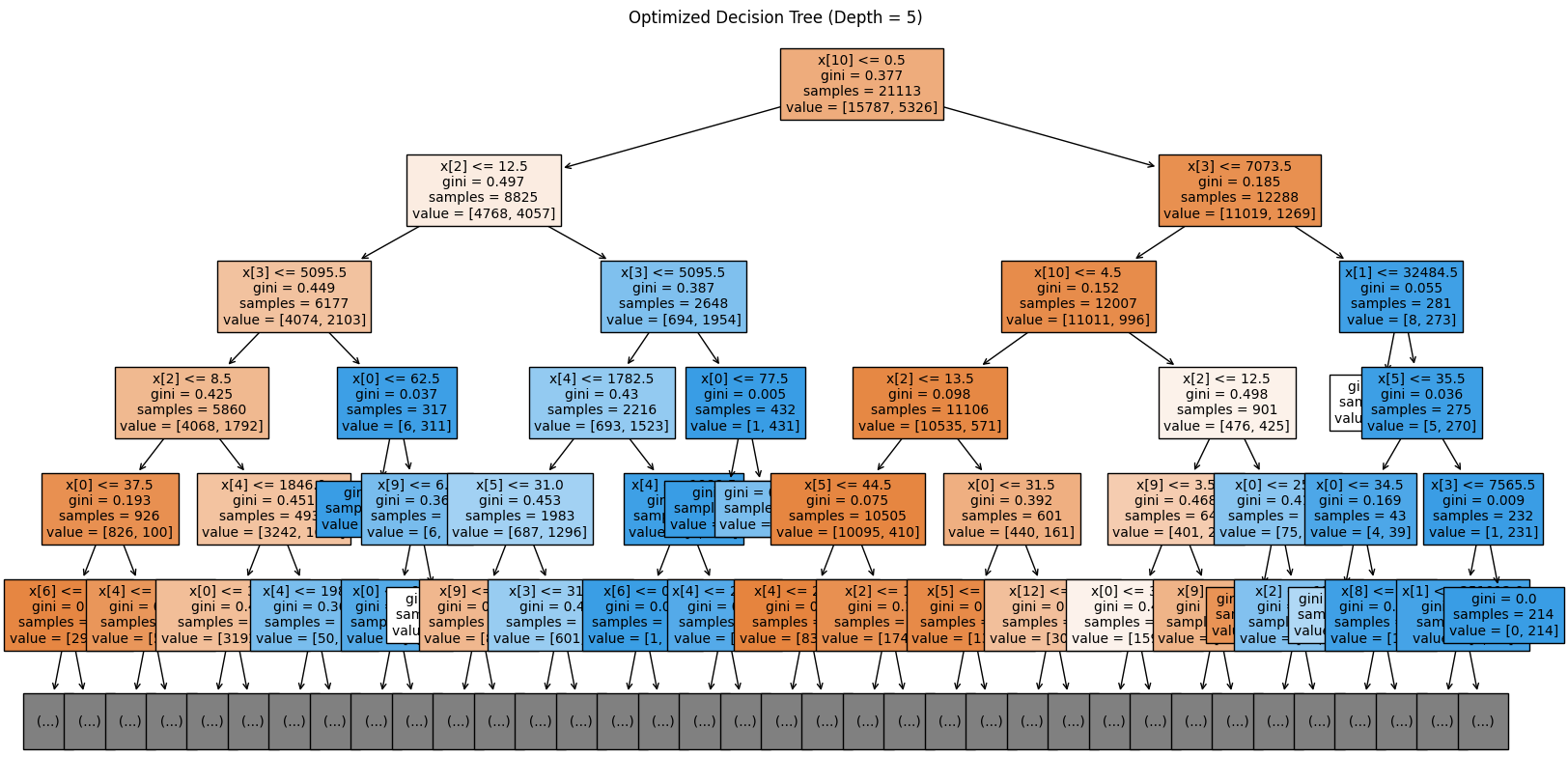
Tuned Decision Tree Classifier:  
Accuracy score: 84.98  
F1 score: 66.03

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:**

After hyperparameter tuning, the Decision Tree model showed improved accuracy and balance in predicting income levels. The adjustments made the model better at correctly identifying both high and low-income classes, resulting in more reliable and precise predictions. This indicates that tuning the model's parameters made it more effective overall.

|  |
| --- |
| 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 \  
1 82 Private 132870 HS-grad 9 Widowed   
3 54 Private 140359 7th-8th 4 Divorced   
4 41 Private 264663 Some-college 10 Separated   
5 34 Private 216864 HS-grad 9 Divorced   
6 38 Private 150601 10th 6 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 \  
1 Exec-managerial Not-in-family White Female 0   
3 Machine-op-inspct Unmarried White Female 0   
4 Prof-specialty Own-child White Female 0   
5 Other-service Unmarried White Female 0   
6 Adm-clerical Unmarried White Male 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   
1 4356 18 United-States <=50K   
3 3900 40 United-States <=50K   
4 3900 40 United-States <=50K   
5 3770 45 United-States <=50K   
6 3770 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   
  
[30162 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 \  
1 Private HS-grad Widowed Exec-managerial Not-in-family   
3 Private 7th-8th Divorced Machine-op-inspct Unmarried   
4 Private Some-college Separated Prof-specialty Own-child   
5 Private HS-grad Divorced Other-service Unmarried   
6 Private 10th Separated Adm-clerical Unmarried   
  
 race sex native.country income   
1 White Female United-States <=50K   
3 White Female United-States <=50K   
4 White Female United-States <=50K   
5 White Female United-States <=50K   
6 White Male 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 \  
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 income   
1 38 0   
3 38 0   
4 38 0   
5 38 0   
6 38 0

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

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  
... ... ... ... ... ... ...  
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  
  
[30162 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 \  
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 income   
1 38 0   
3 38 0   
4 38 0   
5 38 0   
6 38 0

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

print(df)

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   
... ... ... ... ... ... ...   
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 \  
1 2 11 6 3 1 4   
3 2 5 0 6 4 4   
4 2 15 5 9 3 4   
5 2 11 0 7 4 4   
6 2 0 5 0 4 4   
... ... ... ... ... ... ...   
32556 2 15 4 10 1 4   
32557 2 7 2 12 5 4   
32558 2 11 2 6 0 4   
32559 2 11 6 0 4 4   
32560 2 11 4 0 3 4   
  
 sex native.country income   
1 0 38 0   
3 0 38 0   
4 0 38 0   
5 0 38 0   
6 1 38 0   
... ... ... ...   
32556 1 38 0   
32557 0 38 0   
32558 1 38 1   
32559 0 38 0   
32560 1 38 0   
  
[30162 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.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, f1\_score, confusion\_matrix, ConfusionMatrixDisplay  
  
# Initialize the Random Forest model  
rf = RandomForestClassifier(random\_state=42)

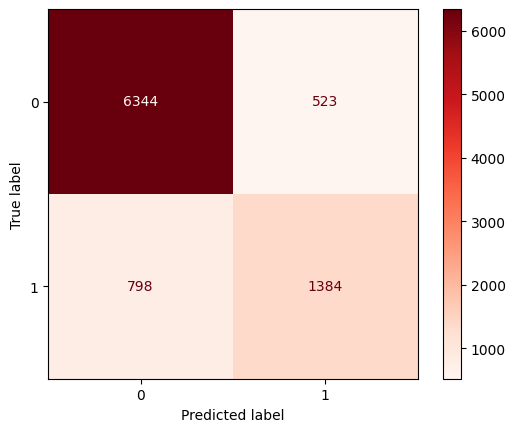
# Fit the model on the training data  
rf.fit(X\_train, Y\_train)  
  
# Predict the labels on the test data  
Y\_pred\_rf = rf.predict(X\_test)

# Evaluate the performance of the model  
print('Random Forest Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_rf) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_rf) \* 100, 2))

Random Forest Classifier:  
Accuracy score: 85.4  
F1 score: 67.69

# Confusion matrix  
cm\_rf = confusion\_matrix(Y\_test, Y\_pred\_rf)  
disp\_rf = ConfusionMatrixDisplay(confusion\_matrix=cm\_rf)  
disp\_rf.plot(cmap='Reds')

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7d74dbe936a0>

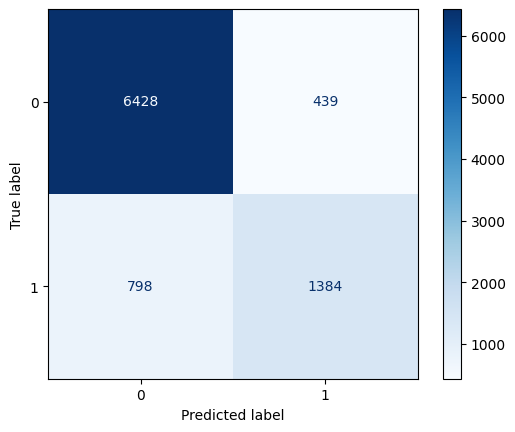


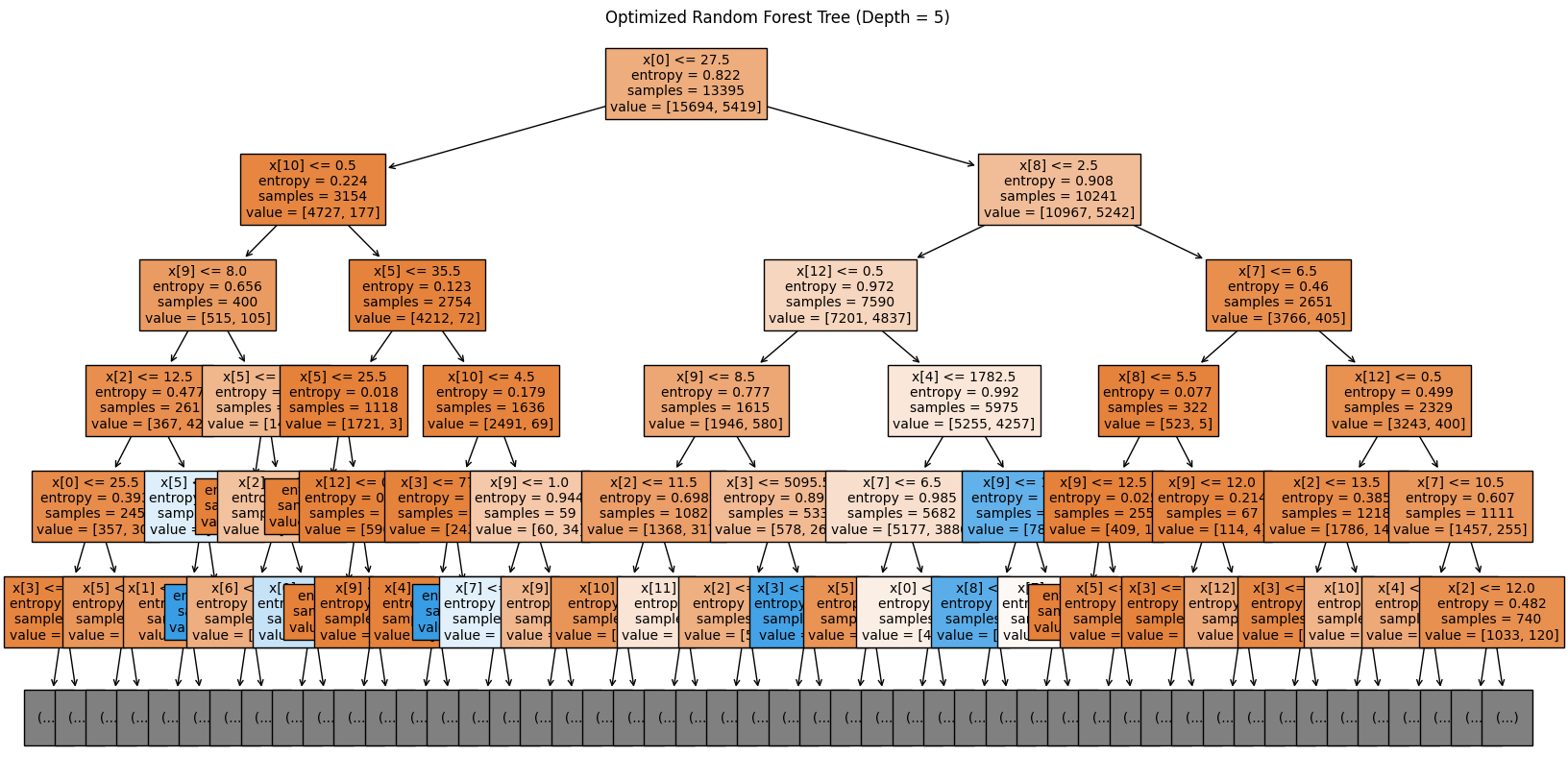
from sklearn.model\_selection import RandomizedSearchCV  
  
# Define the parameter grid to search  
param\_grid\_rf = {  
 'n\_estimators': [50, 100, 200],  
 'max\_depth': [3, 5, 10, None],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'criterion': ['gini', 'entropy'],  
 'max\_features': [None, 'sqrt', 'log2']  
}  
  
# Create the RandomizedSearchCV object  
random\_search\_rf = RandomizedSearchCV(estimator=RandomForestClassifier(random\_state=42),  
 param\_distributions=param\_grid\_rf,  
 n\_iter=20, # Number of parameter settings that are sampled  
 scoring='accuracy',  
 cv=3, # 3-fold cross-validation  
 verbose=1,  
 n\_jobs=-1,  
 random\_state=42)  
  
# Fit the model using RandomizedSearchCV  
random\_search\_rf.fit(X\_train, Y\_train)  
  
# Best parameters and score  
print(f"Best Parameters: {random\_search\_rf.best\_params\_}")  
print(f"Best Score: {random\_search\_rf.best\_score\_}")

Fitting 3 folds for each of 20 candidates, totalling 60 fits  
Best Parameters: {'n\_estimators': 200, 'min\_samples\_split': 2, 'min\_samples\_leaf': 2, 'max\_features': 'sqrt', 'max\_depth': None, 'criterion': 'entropy'}  
Best Score: 0.8583810758783237

# Use the best estimator to predict the test set  
best\_rf = random\_search\_rf.best\_estimator\_  
Y\_pred\_best\_rf = best\_rf.predict(X\_test)  
  
print('Tuned Random Forest Classifier:')  
print('Accuracy score:', round(accuracy\_score(Y\_test, Y\_pred\_best\_rf) \* 100, 2))  
print('F1 score:', round(f1\_score(Y\_test, Y\_pred\_best\_rf) \* 100, 2))  
  
# Confusion matrix for the tuned model  
cm\_best\_rf = confusion\_matrix(Y\_test, Y\_pred\_best\_rf)  
disp\_best\_rf = ConfusionMatrixDisplay(confusion\_matrix=cm\_best\_rf)  
disp\_best\_rf.plot(cmap='Blues')  
  
from sklearn import tree  
  
# Plot one of the trees in the Random Forest (for visualization)  
plt.figure(figsize=(20, 10))  
tree.plot\_tree(best\_rf.estimators\_[0], max\_depth=5, filled=True, fontsize=10)  
plt.title('Optimized Random Forest Tree (Depth = 5)')  
plt.show()

Tuned Random Forest Classifier:  
Accuracy score: 86.33  
F1 score: 69.11





from sklearn.metrics import precision\_score, recall\_score  
  
# Before tuning  
precision\_before = precision\_score(Y\_test, Y\_pred\_rf)  
recall\_before = recall\_score(Y\_test, Y\_pred\_rf)  
accuracy\_before = accuracy\_score(Y\_test, Y\_pred\_rf)  
f1\_before = f1\_score(Y\_test, Y\_pred\_rf)  
confusion\_matrix\_before = confusion\_matrix(Y\_test, Y\_pred\_rf)  
  
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}")  
  
# After tuning  
precision\_after = precision\_score(Y\_test, Y\_pred\_best\_rf)  
recall\_after = recall\_score(Y\_test, Y\_pred\_best\_rf)  
accuracy\_after = accuracy\_score(Y\_test, Y\_pred\_best\_rf)  
f1\_after = f1\_score(Y\_test, Y\_pred\_best\_rf)  
confusion\_matrix\_after = confusion\_matrix(Y\_test, Y\_pred\_best\_rf)  
  
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}")

Before Tuning  
Accuracy: 0.85  
F1 Score: 0.68  
Precision: 0.73  
Recall: 0.63  
Confusion Matrix:   
[[6344 523]  
 [ 798 1384]]  
After Tuning  
Accuracy: 0.86  
F1 Score: 0.69  
Precision: 0.76  
Recall: 0.63  
Confusion Matrix:   
[[6428 439]  
 [ 798 1384]]

**Conclusion:**

After tuning the Random Forest model, accuracy improved slightly from 85% to 86%, and the F1 score increased from 0.68 to 0.69, indicating better overall performance. Precision went up from 0.73 to 0.76, showing the model is now more accurate in identifying positive cases. Recall stayed the same at 0.63, meaning the model's ability to detect all actual positive cases didn’t change. The confusion matrix shows that while the model now makes fewer false positive predictions, the number of missed positive cases remains unchanged. Overall, the tuning led to modest improvements in precision and overall accuracy.

|  |
| --- |
| 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.

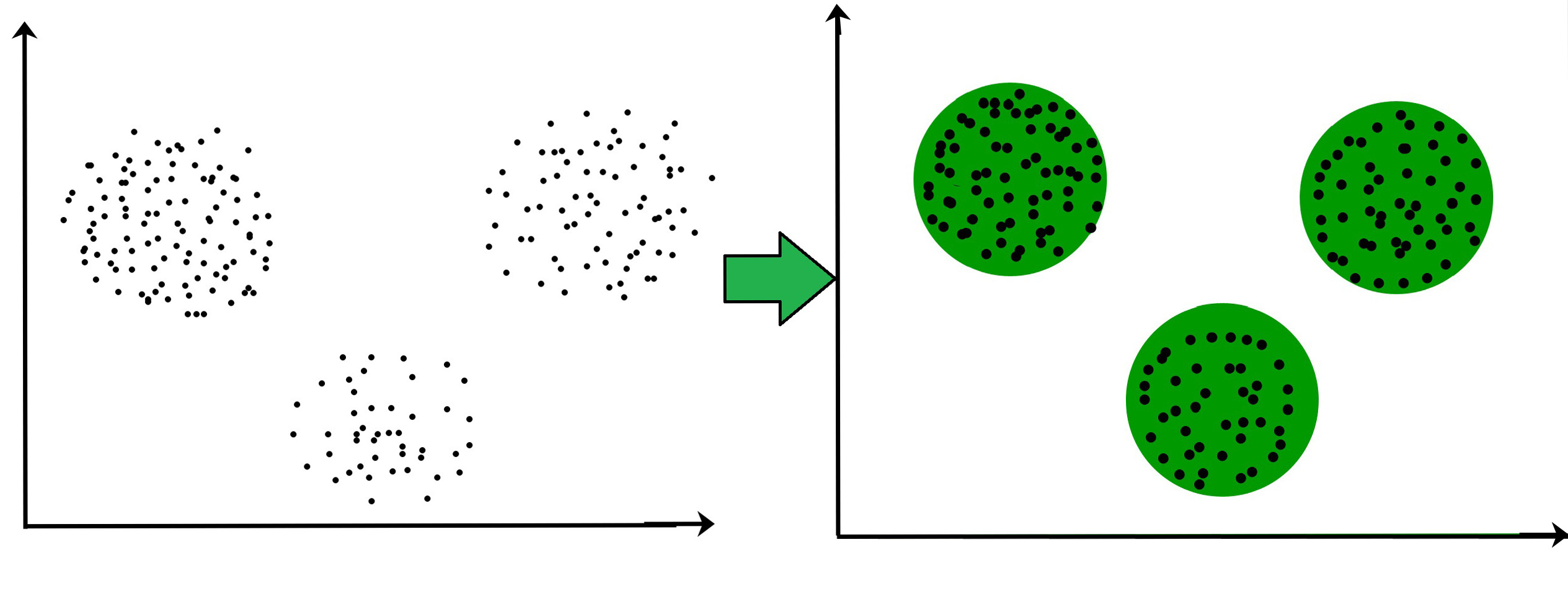
**Objective:** Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

**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)

**Conclusion:**

Comment on results obtained and its visualization.

|  |
| --- |
| 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.

**Conclusion:**

Comment on the impact of dimensionality reduction on the accuracy, precision, recall and F1 score.