DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SCHOOL OF TECHNOLOGY

PANDIT DEENDAYAL ENERGY UNIVERSITY

SESSION 2024-25

SUBMITTED BY

NAME	**:**	**ALAUKI MANISH PATEL**
ROLL NO.	**:**	**24MDS002**
DIVISION	**:**	**1**
COURSE NAME	**:**	**APPLIED MACHINE LEARNING**
COURSE CODE	**:**	**24DS503T**

SUBMITTED TO

Dr. Rajeev Kumar Gupta

Assistant Professor

Department of Computer Science and Engineering

Pandit Deendayal Energy University

****

INDEX

ML: ASSIGNMENT - 1 : Numpy Exercise

---

1) WAP to take 10 inputs from the user as comma separated and stored into a 5*2 matrix.

```
import numpy as np

a = input("Enter 10 values: ")
num = [int(x) for x in a.split(",")]

if len(num) == 10:
  matrix = np.array(num).reshape(5,2)
  print(matrix)
else:
  print("Invalid Input, Please enter 10 values.")
```

```
Enter 10 values: 1,2,3,4,5,6,7,8,9,0
[[1 2]
 [3 4]
 [5 6]
 [7 8]
 [9 0]]
```

2) WAP to create a 4*5 matrix and reverse the elements of 3rd row only.

```
a = np.array([[1,2,3,4,5],[11,12,13,14,15],[21,22,23,24,25],[31,32,33,34,35]])
```

```
a
```

```
array([[ 1,  2,  3,  4,  5],
       [11, 12, 13, 14, 15],
       [21, 22, 23, 24, 25],
       [31, 32, 33, 34, 35]])
```

```
a[2]=a[2][::-1]
a
```

```
array([[ 1,  2,  3,  4,  5],
       [11, 12, 13, 14, 15],
       [25, 24, 23, 22, 21],
       [31, 32, 33, 34, 35]])
```

3) WAP to create a 4*5 matrix and reverse the elements of 2rd column only.

```
b = np.array([[1,2,3,4,5],[11,12,13,14,15],[21,22,23,24,25],[31,32,33,34,35]])
```

```
b[:,1]=b[:,1][::-1]
```

```
b
```

```
array([[ 1, 32,  3,  4,  5],
       [11, 22, 13, 14, 15],
       [21, 12, 23, 24, 25],
       [31,  2, 33, 34, 35]])
```

4) Write a NumPy program to test whether each element of a 1-D array is also present in a second array.

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([1, 2, 3, 4, 5])

arr = np.array([i for i in arr1 for j in arr2 if i == j])

if len(arr) == len(arr1)&len(arr2):
    print("All elements of the first array are present in the second array.")
else:
    print("Not all elements of the first array are present in the second array.")
```

```
All elements of the first array are present in the second array.
```

5) Write a NumPy program to find common values between two arrays using for loop anf if else statement

```
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([2, 4, 6, 8, 10])

common_values = []

for element in arr1:
  if element in arr2:
    common_values.append(element)

print("Common elements:", common_values)
```

```
Common elements: [2, 4]
```

6) Take two 1-D array and make a new 1D array where all elements are greater than 5.

```
arr1 = np.array([1, 2, 3, 4, 5,9])
arr2 = np.array([2, 4, 6, 8, 10,11])

greater_values = []

for element in arr1:
  if element > 5:
    greater_values.append(element)
for element in arr2:
  if element > 5:
    greater_values.append(element)

print("Greater elements:", greater_values)
```

```
Greater elements: [9, 6, 8, 10, 11]
```

7) You are given a space separated list of numbers. Your task is to print a reversed NumPy array with the element type float.

```
c = input("Enter values : ")
num = [float(x) for x in c.split()]
arr = np.array(num)
reverse = arr[::-1]
print(reverse)
```

```
Enter values : 1 2 -7 -5
[-5. -7.  2.  1.]
```

8) Concatenate two size of arrays along axis 0.

```
# m= 4 n=3 p=2
a = input("Enter 8 values : ")
b = input("Enter 6 values: ")

num1 = [int(x) for x in a.split()]
num2 = [int(x) for x in b.split()]

arr1 = np.array(num1).reshape(4,2)
arr2 = np.array(num2).reshape(3,2)

arr3 = np.concatenate((arr1,arr2),axis=0)
print(arr3)
```

```
Enter 8 values : 1 2 3 4 5 6 7 8
Enter 6 values: 1 2 3 4 5 6
[[1 2]
 [3 4]
 [5 6]
 [7 8]
 [1 2]
 [3 4]
```

```
[5 6]]
```

ML: Assignment-2 : Pandas Exercise

1) Consider the following seriesser = pd.Series(np.random.randint(1, 10, 7)). find the positions of numbers that are multiples of 3 from a series?

```
import pandas as pd
import numpy as np
```

```
a = pd.Series(np.random.randint(1,10,7))
```

```
a
```

	0
0	5
1	5
2	6
3	4
4	8
5	2
6	9

```
b = a[a%3==0].index
b
```

```
Index([2, 6], dtype='int64')
```

2) Create the series of 100 random numbers and extract the element at the even position.

```
t = pd.Series(np.random.randint(1,100,100))
```

```
tt = t[t.index % 2 == 0]
tt
```

	0
0	53
2	26
4	37
6	37
8	15
10	20
12	63
14	97
16	71
18	52
20	11
22	56
24	38
26	81
28	74
30	54
32	78
34	29
36	91
38	26
40	6
42	44
44	27
46	44
48	80
50	84
52	99
54	55
56	14
58	82
60	91
62	26
64	19
66	76
68	45
70	75
72	1
74	70
76	83
78	36
80	71
82	24
84	78
86	74
88	45
90	74
92	82
94	79
96	11

98	84

dtype: int64

3) Convert the first character of each element in a series to uppercase

```
u = pd.Series(['alauki','manish','patel'])
u
```

	0
0	alauki
1	manish
2	patel

dtype: object

```
u.str.capitalize()
```

	0
0	Alauki
1	Manish
2	Patel

dtype: object

4) Complete the Euclidean distance between two series

p = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

q = pd.Series([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

```
p = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
q = pd.Series([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
```

```
np.sqrt(np.sum((p-q)**2))
```

```
18.16590212458495
```

5) Apply the label encoding to replace the string the given dataframe.

Marks1	Marks2	Grade	Result
10	50	B	PASS
20	60	C	FAIL
30	40	B	PASS
40	85	A	PASS
50	83	A	FAIL

```
dic = {
    'Marks1':['10','20','30','40','50'],
    'Marks2':['50','60','40','85','83'],
    'Grade':['B','C','B','A','A'],
    'Result':['PASS','FAIL','PASS','PASS','FAIL']
}
```

```
df = pd.DataFrame(dic)
df
```

	Marks1	Marks2	Grade	Result
0	10	50	B	PASS
1	20	60	C	FAIL
2	30	40	B	PASS
3	40	85	A	PASS
4	50	83	A	FAIL

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

cols = ['Grade','Result']
for col in cols:
  df[col] = le.fit_transform(df[col])
df
```

	Marks1	Marks2	Grade	Result
0	10	50	1	1
1	20	60	2	0
2	30	40	1	1
3	40	85	0	1
4	50	83	0	0

6) Create the 3 DataFrames based on the following raw

dataraw_data_1 = {'subject_id': ['1', '2', '3', '4', '5'],'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}

raw_data_2 = { 'subject_id': ['4', '5', '6', '7', '8'],'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}

raw_data_3 = {'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}

Step 1. Assign each to a variable called data1, data2, data3

```
dataraw_data_1 = {
    'subject_id': ['1', '2', '3', '4', '5'],
    'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']
}
data1 = pd.DataFrame(dataraw_data_1)
```

```
raw_data_2 = {
    'subject_id': ['4', '5', '6', '7', '8'],
    'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']
}
data2 = pd.DataFrame(raw_data_2)
```

```
raw_data_3 = {
    'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
    'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]
}
data3 = pd.DataFrame(raw_data_3)
```

Step 2. Join the two dataframes along rows and assign all_data

```
all_data = pd.concat([data1,data2],axis=0)
all_data
```

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Step 3. Join the two dataframes along columns and assing to all_data_col

```
all_data_1 = pd.concat([data1,data2],axis=1)
all_data_1
```

	subject_id	first_name	last_name	subject_id	first_name	last_name
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner
3	4	Alice	Aoni	7	Bryce	Brice
4	5	Ayoung	Atiches	8	Betty	Btisan

Step 4. Print data3

```
data3
```

	subject_id	test_id
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

Step 5. Merge all_data and data3 along the subject_id value

```
merged_data = pd.merge(all_data, data3, on='subject_id')
merged_data
```

ML:Assignment-3

Implement simple and multi-linear regression to predict profits for a food truck. Compare the performance of the model on linear and multi-linear regression.

https://www.kaggle.com/datasets/gsainathreddy/food-truck-data

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

```
data = pd.read_csv("/content/food_truck_data.txt")
data
```

	Population	Profit
0	6.1101	17.59200
1	5.5277	9.13020
2	8.5186	13.66200
3	7.0032	11.85400
4	5.8598	6.82330
...	...	...
92	5.8707	7.20290
93	5.3054	1.98690
94	8.2934	0.14454
95	13.3940	9.05510
96	5.4369	0.61705

97 rows × 2 columns

```
data.shape
```

```
(97, 2)
```

```
data.columns
```

```
Index(['Population', 'Profit'], dtype='object')
```

```
data.isna().any()
```

	0
Population	False
Profit	False

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Population  97 non-null     float64
 1   Profit      97 non-null     float64
dtypes: float64(2)
memory usage: 1.6 KB
```

```
x = data[['Population']]
y = data['Profit']
```

```
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 =42)
```

LinearRegression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

model = LinearRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: ",mse)
print(f"R^2 Score: ",r2)
```

```
MSE:  15.709362447765187
R^2 Score:  0.500344113338578
```

For multi-linear Regression

Let's create a new col

```
data['Population^2'] = data['Population']**2
data
```

	Population	Profit	Population^2
0	6.1101	17.59200	37.333322
1	5.5277	9.13020	30.555467
2	8.5186	13.66200	72.566546
3	7.0032	11.85400	49.044810
4	5.8598	6.82330	34.337256
...	...	...	...
92	5.8707	7.20290	34.465118
93	5.3054	1.98690	28.147269
94	8.2934	0.14454	68.780484
95	13.3940	9.05510	179.399236
96	5.4369	0.61705	29.559882

97 rows × 3 columns

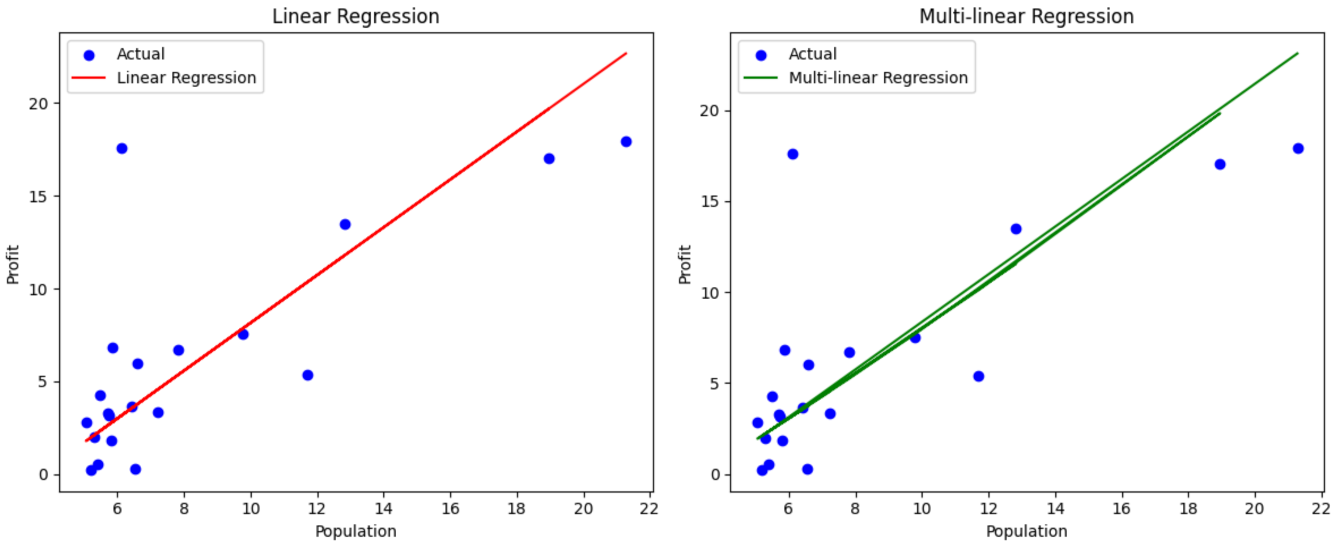
```
X = data[['Population','Population^2']]
y = data['Profit']
```

```
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 =42)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

model = LinearRegression()
model.fit(X_train, y_train)

y_pred2 = model.predict(X_test)

mse2 = mean_squared_error(y_test, y_pred2)
r2_2 = r2_score(y_test, y_pred2)

print(f"MSE: ",mse2)
print(f"R^2 Score: ",r2_2)
```

```
MSE:  15.815792426525025
R^2 Score:  0.4969589749803903
```

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

plt.subplot(1, 2, 1)
plt.scatter(x_test['Population'], y_test, color='blue', label='Actual')
plt.plot(x_test['Population'], y_pred, color='red', label='Linear Regression')
plt.xlabel('Population')
plt.ylabel('Profit')
plt.title('Linear Regression')
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(x_test['Population'], y_test, color='blue', label='Actual')
plt.plot(x_test['Population'], y_pred2, color='green', label='Multi-linear Regression')
plt.xlabel('Population')
plt.ylabel('Profit')
plt.title('Multi-linear Regression')
plt.legend()

plt.tight_layout()
plt.show()
```



ML:Assignment-4

Apply different Machine Learning approaches on the Crop Recommendation Dataset. Compare the performance of different ML approaches in term of accuracy, precision and recall.

https://www.kaggle.com/datasets/atharvaingle/crop-recommendation-dataset?select=Crop_recommendation.csv

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn import tree
```

```
df = pd.read_csv("/content/Crop_recommendation.csv")
df
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	coffee
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	coffee
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	coffee
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	coffee
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	coffee

2200 rows × 8 columns

```
df.size
```

```
17600
```

```
df.shape
```

```
(2200, 8)
```

```
df.columns
```

```
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

```
df['label'].unique() #22
```

```
array(['rice', 'maize', 'chickpea', 'kidneybeans', 'pigeonpeas',
       'mothbeans', 'mungbean', 'blackgram', 'lentil', 'pomegranate',
       'banana', 'mango', 'grapes', 'watermelon', 'muskmelon', 'apple',
       'orange', 'papaya', 'coconut', 'cotton', 'jute', 'coffee'],
      dtype=object)
```

```
df['label'].value_counts()
```

label	count
rice	100
maize	100
jute	100
cotton	100
coconut	100
papaya	100
orange	100
apple	100
muskmelon	100
watermelon	100
grapes	100
mango	100
banana	100
pomegranate	100
lentil	100
blackgram	100
mungbean	100
mothbeans	100
pigeonpeas	100
kidneybeans	100
chickpea	100
coffee	100

dtype: int64

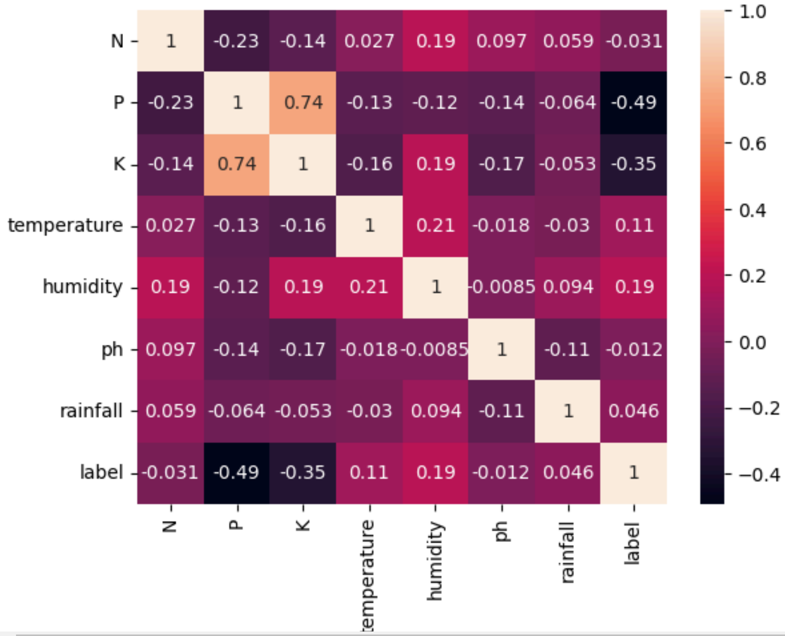
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])
df
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	20
1	85	58	41	21.770462	80.319644	7.038096	226.655537	20
2	60	55	44	23.004459	82.320763	7.840207	263.964248	20
3	74	35	40	26.491096	80.158363	6.980401	242.864034	20
4	78	42	42	20.130175	81.604873	7.628473	262.717340	20
...	...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	5
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	5
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	5
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	5
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	5

2200 rows × 8 columns

```
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```



Seperating features and target label

```
x = df[['N', 'P','K','temperature', 'humidity', 'ph', 'rainfall']]
y = df['label']
```

```
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 =42)
```

```
acc = []
model_name = []
precision = []
recall = []
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

log = LogisticRegression()
log.fit(x_train, y_train)

predicted_log = log.predict(x_test)

x = accuracy_score(y_test, predicted_log)
pre = precision_score(y_test, predicted_log, average='macro')
re = recall_score(y_test, predicted_log, average='macro')

acc.append(x)
model_name.append('Logistic Regression')
precision.append(pre)
recall.append(re)

print("logistic regression's Accuracy is: ", x * 100)
print("Precision: ", pre)
print("Recall: ", re)
```

```
logistic regression's Accuracy is:  94.54545454545455
Precision:  0.9439047973948057
Recall:  0.9463285374667674
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

rf = RandomForestClassifier()
rf.fit(x_train, y_train)

predicted_rf = rf.predict(x_test)

x = accuracy_score(y_test, predicted_rf)
pre = precision_score(y_test, predicted_rf, average='macro')
re = recall_score(y_test, predicted_rf, average='macro')

acc.append(x)
model_name.append('Random Forest')
precision.append(pre)
recall.append(re)

print("logistic regression's Accuracy is: ", x * 100)
print("Precision: ", pre)
print("Recall: ", re)
```

```
logistic regression's Accuracy is:  99.31818181818181
Precision:  0.9925757575757576
Recall:  0.9933213716108454
```

SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score

SVM = SVC()
SVM.fit(x_train, y_train)

predicted_svm = SVM.predict(x_test)

x = accuracy_score(y_test, predicted_svm)
pre = precision_score(y_test, predicted_svm, average='macro')
re = recall_score(y_test, predicted_svm, average='macro')

acc.append(x)
model_name.append('SVM')
precision.append(pre)
recall.append(re)

print("SVM's Accuracy is: ", x * 100)
print("Precision: ", pre)
print("Recall: ", re)
```

```
SVM's Accuracy is:  96.13636363636363
Precision:  0.9632920110192837
Recall:  0.9628916732076647
```

DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)

predicted_dt = dt.predict(x_test)

x = accuracy_score(y_test, predicted_dt)
pre = precision_score(y_test, predicted_dt, average='macro')
re = recall_score(y_test, predicted_dt, average='macro')

acc.append(x)
model_name.append('Decision Tree')
precision.append(pre)
recall.append(re)

print("DecisionTree's Accuracy is: ", x * 100)
```

```
print("Precision: ", pre)
print("Recall: ", re)
```

```
DecisionTree's Accuracy is:  98.4090909090909
Precision:  0.9830972058244786
Recall:  0.9854985784619651
```

Naive Bayes

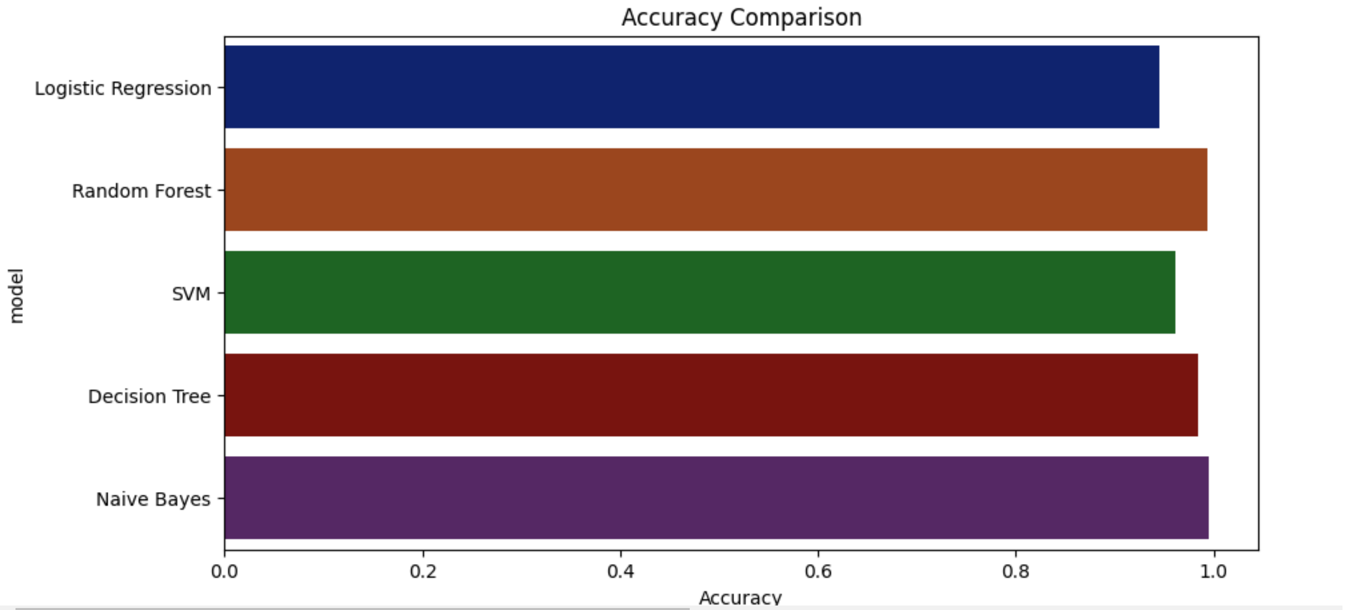
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score

nb = GaussianNB()
nb.fit(x_train, y_train)

predicted_nb = nb.predict(x_test)

x = accuracy_score(y_test, predicted_nb)
pre = precision_score(y_test, predicted_nb, average='macro')
re = recall_score(y_test, predicted_nb, average='macro')

acc.append(x)
model_name.append('Naive Bayes')
precision.append(pre)
recall.append(re)

print("Naive Bayes's Accuracy is: ", x * 100)
print("Precision: ", pre)
print("Recall: ", re)
```

```
Naive Bayes's Accuracy is:  99.54545454545455
Precision:  0.9963636363636365
Recall:  0.9952153110047847
```

Accuracy Comparison

```
plt.figure(figsize=[10,5],dpi = 100)
plt.title('Accuracy Comparison')
plt.xlabel('Accuracy')
plt.ylabel('model')
sns.barplot(x = acc,y = model_name,palette='dark')
```

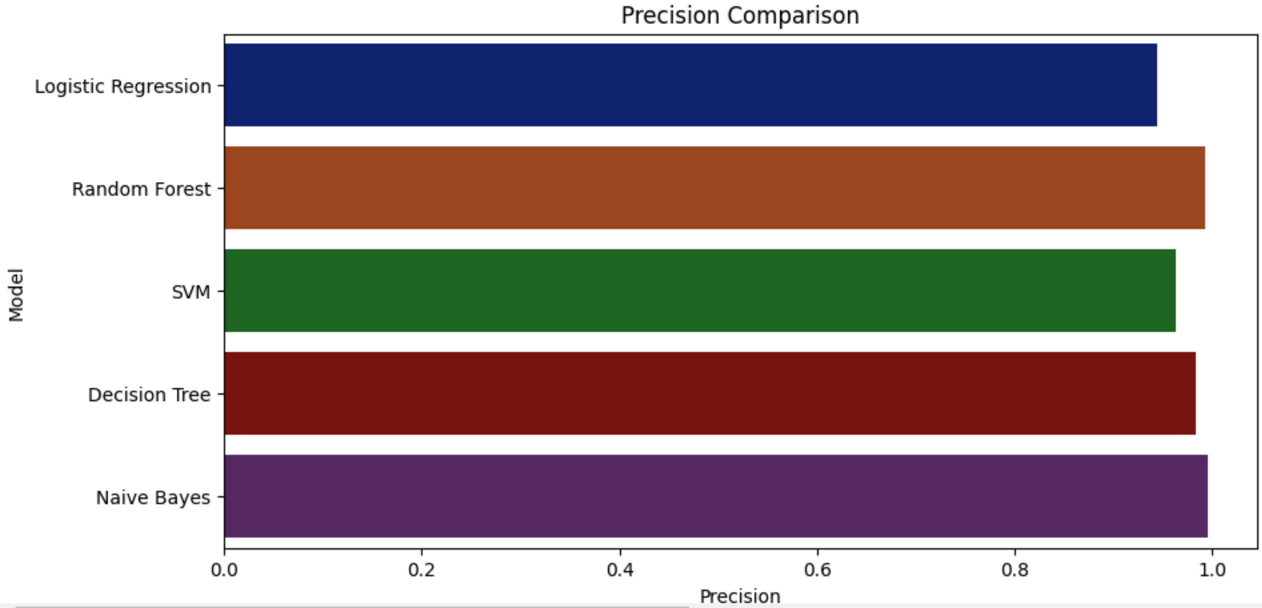
```
<ipython-input-19-cd2fffd397bd>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x = acc,y = model_name,palette='dark')
<Axes: title={'center': 'Accuracy Comparison'}, xlabel='Accuracy', ylabel='model'>
```




```
plt.figure(figsize=[10, 5], dpi=100)
plt.title('Precision Comparison')
plt.xlabel('Precision')
```

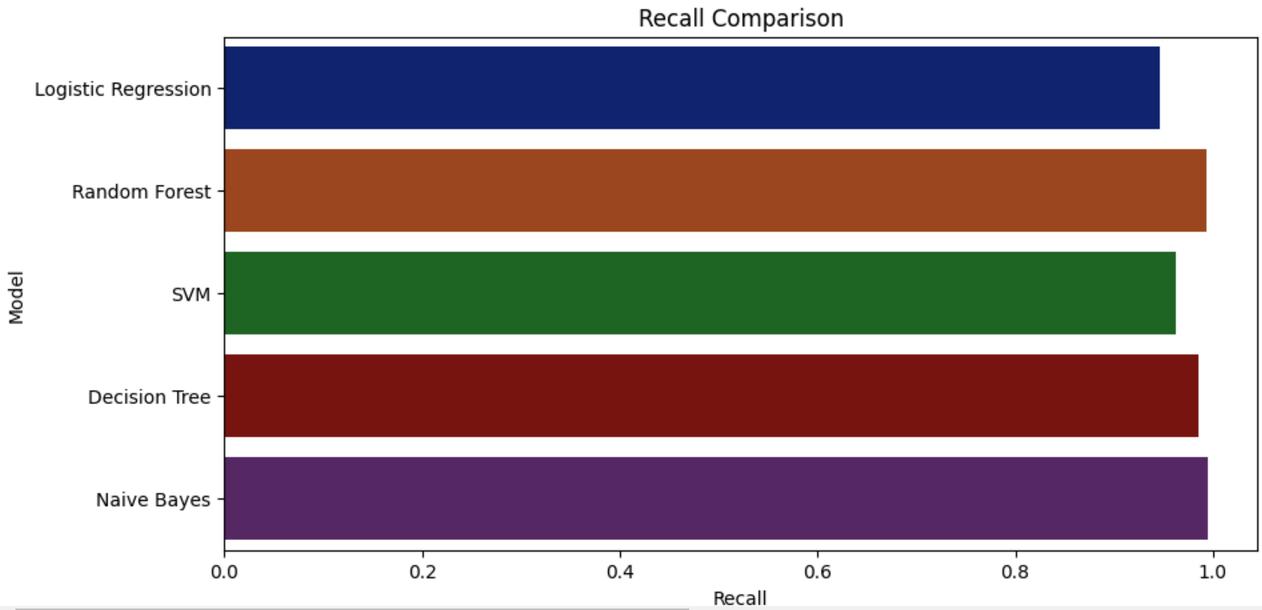
```
plt.ylabel('Model')
sns.barplot(x=precision, y=model_name, palette='dark')
plt.show()
```

```
<ipython-input-25-5143ee05f103>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x=precision, y=model_name, palette='dark')
```




```
plt.figure(figsize=[10, 5], dpi=100)
plt.title('Recall Comparison')
plt.xlabel('Recall')
plt.ylabel('Model')
sns.barplot(x=recall, y=model_name, palette='dark')
plt.show()
```

```
<ipython-input-22-cce685426c0a>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x=recall, y=model_name, palette='dark')
```



Start coding or generate with AI.

ML: ASSIGNMENT - 5 :

Apply different feature selection approaches for the classification/regression task (take any dataset). Compare the performance of different feature selection approach.

```
!pip install feature-engine
```

```
Collecting feature-engine
  Downloading feature_engine-1.8.1-py2.py3-none-any.whl.metadata (9.8 kB)
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.26.4)
Requirement already satisfied: pandas>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (2.2.2)
Requirement already satisfied: scikit-learn>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.5.2)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.13.1)
Requirement already satisfied: statsmodels>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (0.14.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.2.0->feature-engine
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.2.0->feature-engine) (2024.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.2.0->feature-engine) (2024
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.0->feature-engine)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.0->feature-er
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.11.1->feature-engine) ((
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.11.1->feature-engine
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels>=0.11.1->feature-engir
Downloading feature_engine-1.8.1-py2.py3-none-any.whl (364 kB)
                                          364.1/364.1 kB 5.5 MB/s eta 0:00:00
Installing collected packages: feature-engine
Successfully installed feature-engine-1.8.1
```

```
import pandas as pd
import numpy as np
from feature_engine.selection import DropConstantFeatures, DropDuplicateFeatures, DropCorrelatedFeatures, SmartCorrelatedSelection
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
df = pd.read_csv("/content/data.csv")
```

```
df.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

```
df.shape
```

```
(569, 32)
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 32 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave_points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave_points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave_points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	conca
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 31 columns

```
df.isnull().sum()
```

	0
id	0
diagnosis	0
radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave_points_mean	0
symmetry_mean	0
fractal_dimension_mean	0
radius_se	0
texture_se	0
perimeter_se	0
area_se	0
smoothness_se	0
compactness_se	0
concavity_se	0
concave_points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave_points_worst	0
symmetry_worst	0

```
df = df.drop('id',axis=1)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

define x and y

```
x = df.drop('diagnosis',axis=1)
y = df['diagnosis']
```

```
x.shape
```

```
(569, 30)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

```
model = RandomForestClassifier(random_state=42)
model.fit(x_train, y_train)
y_predict = model.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_predict)
precision = precision_score(y_test, y_predict)
```

```
recall = recall_score(y_test, y_predict)
f1 = f1_score(y_test, y_predict)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9707602339181286
Precision: 0.9833333333333333
Recall: 0.9365079365079365
F1 Score: 0.959349593495935
```

DropConstantFeatures method

```
cons = DropConstantFeatures(tol=0.97)
# more than 97% common values
```

```
x_train_cons = cons.fit_transform(x_train)
x_test_cons = cons.transform(x_test)
```

```
a=cons.features_to_drop_
a
```

```
[]
```

All features have some variability, No redundant features

```
model = RandomForestClassifier(random_state=42)
model.fit(x_train_cons, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred_cons = model.predict(x_test_cons)
accuracy = accuracy_score(y_test, y_pred_cons)
precision = precision_score(y_test, y_pred_cons)
recall = recall_score(y_test, y_pred_cons)
f1 = f1_score(y_test, y_pred_cons)
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9707602339181286
Precision: 0.9833333333333333
Recall: 0.9365079365079365
F1 Score: 0.959349593495935
```

DropDuplicateFeatures method

```
dup = DropDuplicateFeatures()
x_train_dup = dup.fit_transform(x_train)
len(dup.features_to_drop_)
```

```
0
```

no duplicate features

```
x_test_dup = dup.transform(x_test)
```

```
model.fit(x_train_dup, y_train)
y_pred_dup = model.predict(x_test_dup)
```

```
accuracy = accuracy_score(y_test, y_pred_dup)
precision = precision_score(y_test, y_pred_dup)
```

```
recall = recall_score(y_test, y_pred_dup)
f1 = f1_score(y_test, y_pred_dup)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9707602339181286
Precision: 0.9833333333333333
Recall: 0.9365079365079365
F1 Score: 0.959349593495935
```

DropCorrelatedFeatures method

```
cor=DropCorrelatedFeatures(threshold=0.9)
x_train_corr = cor.fit_transform(x_train)
x_test_corr = cor.transform(x_test)
len(cor.features_to_drop_)
```

```
10
```

10 features are highly correlated

```
cor.features_to_drop_
```

```
['area_worst',
 'perimeter_mean',
 'perimeter_worst',
 'radius_mean',
 'radius_worst',
 'perimeter_se',
 'radius_se',
 'concave_points_worst',
 'concavity_mean',
 'texture_worst']
```

```
model.fit(x_train_corr, y_train)
y_pred_corr = model.predict(x_test_corr)
```

```
accuracy = accuracy_score(y_test, y_pred_corr)
precision = precision_score(y_test, y_pred_corr)
recall = recall_score(y_test, y_pred_corr)
f1 = f1_score(y_test, y_pred_corr)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9649122807017544
Precision: 0.9523809523809523
Recall: 0.9523809523809523
F1 Score: 0.9523809523809523
```

SmartCorrelatedSelection Method

Selection method:Model performance

```
rf = RandomForestClassifier(n_estimators=10,n_jobs=4,random_state=0)
```

```
sel_corr = SmartCorrelatedSelection(missing_values='raise',estimator=rf,selection_method='model_performance',method="spearman")
```

```
train_smart = sel_corr.fit_transform(x_train,y_train)
x_test_smart = sel_corr.transform(x_test)
```

```
sel_corr.correlated_feature_sets_
```

```
[{'area_mean',
  'area_worst',
  'perimeter_mean',
```

```
  'perimeter_worst',
  'radius_mean',
  'radius_worst'},
 {'area_se', 'perimeter_se', 'radius_se'},
 {'compactness_mean',
  'compactness_se',
  'compactness_worst',
  'concave_points_mean',
  'concave_points_worst',
  'concavity_mean',
  'concavity_worst'},
 {'texture_mean', 'texture_worst'}]
```

```
model.fit(train_smart, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred_smart = model.predict(x_test_smart)
```

```
accuracy = accuracy_score(y_test, y_pred_smart)
precision = precision_score(y_test, y_pred_smart)
recall = recall_score(y_test, y_pred_smart)
f1 = f1_score(y_test, y_pred_smart)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9532163742690059
Precision: 0.9661016949152542
Recall: 0.9047619047619048
F1 Score: 0.9344262295081968
```

Selection method: variance

```
sel_corr_2 = SmartCorrelatedSelection(missing_values='raise',estimator=rf,selection_method='variance',method="spearman")
```

```
train_smart_2 = sel_corr_2.fit_transform(x_train,y_train)
x_test_smart_2 = sel_corr_2.transform(x_test)
```

```
sel_corr_2.correlated_feature_sets_
```

```
[{'area_mean',
  'area_worst',
  'perimeter_mean',
  'perimeter_worst',
  'radius_mean',
  'radius_worst'},
 {'area_se', 'perimeter_se', 'radius_se'},
 {'texture_mean', 'texture_worst'},
 {'compactness_mean',
  'compactness_worst',
  'concave_points_mean',
  'concave_points_worst',
  'concavity_mean',
  'concavity_se',
  'concavity_worst'}]
```

```
model.fit(train_smart_2, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred_smart_2 = model.predict(x_test_smart_2)
```

```
accuracy = accuracy_score(y_test, y_pred_smart_2)
precision = precision_score(y_test, y_pred_smart_2)
recall = recall_score(y_test, y_pred_smart_2)
f1 = f1_score(y_test, y_pred_smart_2)

print("Accuracy:", accuracy)
print("Precision:", precision)
```

```
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9883040935672515
Precision: 0.9841269841269841
Recall: 0.9841269841269841
F1 Score: 0.9841269841269841
```

Mutual Information : SelectKbest

```
from sklearn.feature_selection import mutual_info_classif, SelectKBest
```

```
model = mutual_info_classif(x_train,y_train)
model
```

```
array([0.31949828, 0.11039038, 0.37063367, 0.32865013, 0.07504859,
       0.20383435, 0.36843111, 0.4428238 , 0.04665615, 0.02686585,
       0.2506768 , 0.01583228, 0.25196688, 0.31700637, 0.0192022 ,
       0.04148643, 0.12648959, 0.10231915, 0.02060161, 0.00360293,
       0.44118968, 0.16180683, 0.45333713, 0.43949148, 0.09100639,
       0.21386212, 0.31501979, 0.44824202, 0.07819004, 0.06222421])
```

```
model.shape
```

```
(30,)
```

```
data1 = pd.Series(model)
data1
```

	0
0	0.319498
1	0.110390
2	0.370634
3	0.328650
4	0.075049
5	0.203834
6	0.368431
7	0.442824
8	0.046656
9	0.026866
10	0.250677
11	0.015832
12	0.251967
13	0.317006
14	0.019202
15	0.041486
16	0.126490
17	0.102319
18	0.020602
19	0.003603
20	0.441190
21	0.161807
22	0.453337
23	0.439491
24	0.091006
25	0.213862
26	0.315020
27	0.448242
28	0.078190
29	0.062224

```
data1.index = x_train.columns
data1
```

	0
radius_mean	0.319498
texture_mean	0.110390
perimeter_mean	0.370634
area_mean	0.328650
smoothness_mean	0.075049
compactness_mean	0.203834
concavity_mean	0.368431
concave_points_mean	0.442824
symmetry_mean	0.046656
fractal_dimension_mean	0.026866
radius_se	0.250677
texture_se	0.015832
perimeter_se	0.251967
area_se	0.317006
smoothness_se	0.019202

```
sel = SelectKBest(mutual_info_classif, k=10)
train = sel.fit_transform(x_train,y_train)
```

```
test = sel.transform(x_test)
```

fractal_dimension_se 0.003603

```
selected_features = x_train.columns[sel.get_support()]
```

texture_worst 0.161807

```
selected_features
```

```
Index(['radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean',
       'concave_points_mean', 'radius_worst', 'perimeter_worst', 'area_worst',
       'concavity_worst', 'concave_points_worst'],
      dtype='object')
```

```
selected_features.shape
```

```
(10,)
```

```
model = RandomForestClassifier(random_state=42)
model.fit(train, y_train)
y_pred = model.predict(test)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9473684210526315
Precision: 0.95
Recall: 0.9047619047619048
F1 Score: 0.926829268292683
```

ML: Assignment-6

Train any machine learning classifier on the imbalanced dataset. Then balance the dataset by using oversampling techniques. Compare the model performance before and after oversampling on the credit card fraud detection dataset.

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

```
from imblearn.over_sampling import SMOTE
import pandas as pd
import numpy as np
```

```
data = pd.read_csv("/content/creditcard.csv")
data
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27783
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63867
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77167
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00527
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79827
...	...	...	...	...	...	...	...	...	...	...	...	...	.
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.11186
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.92438
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.57822
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.80004
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.64307

284807 rows × 31 columns

```
data.columns
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

```
data.isnull().sum()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
Amount	0
Class	0

```
#col = ['V18','V19','V20','V21','V22','V23','V24','V25','V26','V27','V28','Amount','Class']
#data[col] = data[col].ffill()
```

```
data['Class'].value_counts()
```

Class	count
0	284315
1	492

```
x = data.drop(['Class'],axis=1)
y = data['Class']
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
```

```
x_test = sc.transform(x_test)
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
model = GaussianNB()

model.fit(x_train,y_train)
y_pred = model.predict(x_test)

print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
0.9784651756141521
[[83480  1816]
 [   24   123]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85296
           1       0.06      0.84      0.12       147

    accuracy                           0.98     85443
   macro avg       0.53      0.91      0.55     85443
weighted avg       1.00      0.98      0.99     85443
```

```
print(x_train.shape)
print(y_train.shape)
```

```
(199364, 30)
(199364,)
```

SMOTE

```
sm = SMOTE(random_state=0)
x_train_sm,y_train_sm = sm.fit_resample(x_train,y_train)
```

```
y_train_sm.value_counts()
```

	count
Class	
0	199019
1	199019

dtype: int64

```
model.fit(x_train_sm,y_train_sm)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred_sm = model.predict(x_test)
print("Accuracy on smote:", accuracy_score(y_test, y_pred_sm))
print("Classification report on smote\n", classification_report(y_test, y_pred_sm))
```

```
Accuracy on smote: 0.9762414709221352
Classification report on smote
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     85296
           1       0.06      0.86      0.11       147

    accuracy                           0.98     85443
   macro avg       0.53      0.92      0.55     85443
weighted avg       1.00      0.98      0.99     85443
```

RandomOverSampler

```
from imblearn.over_sampling import RandomOverSampler
```

```
ran = RandomOverSampler(random_state=42)
x_train_ran, y_train_ran = ran.fit_resample(x_train, y_train)
```

```
y_train_ran.value_counts()
```

Class	count
0	199019
1	199019

dtype: int64

```
model.fit(x_train_ran, y_train_ran)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred_ran = model.predict(x_test)
print("Accuracy on RandomOverSampler:", accuracy_score(y_test, y_pred_ran))
print("Classification report on RandomOverSampler:\n", classification_report(y_test, y_pred_ran))
```

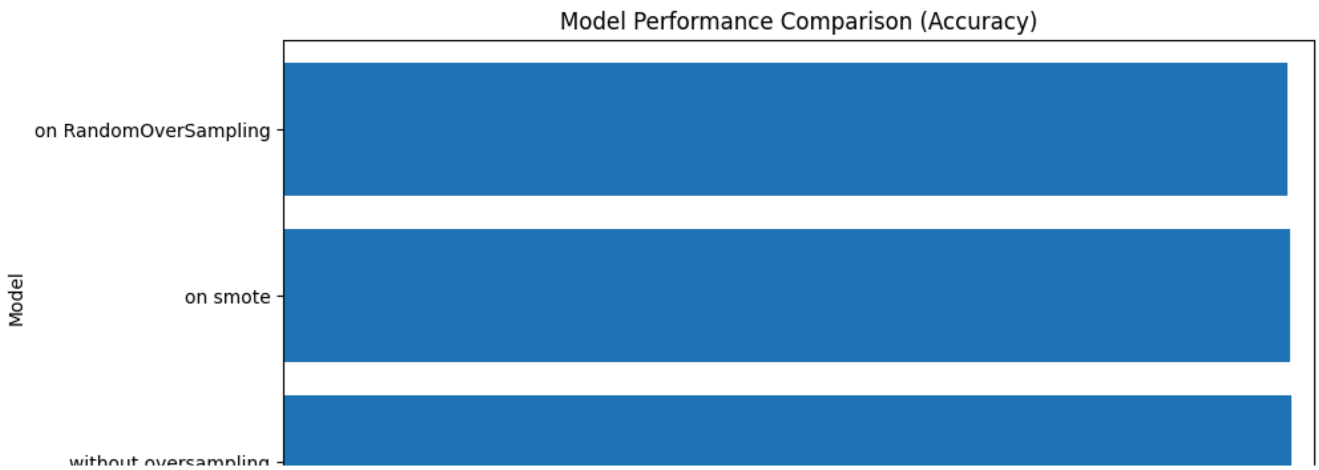
```
Accuracy on RandomOverSampler: 0.9744859145863324
Classification report on RandomOverSampler:
               precision    recall  f1-score   support

           0       1.00      0.97      0.99     85296
           1       0.06      0.86      0.10       147

    accuracy                           0.97     85443
   macro avg       0.53      0.92      0.55     85443
weighted avg       1.00      0.97      0.99     85443
```

Comparison

```
import matplotlib.pyplot as plt
```

```
model_names = ['without oversampling','on smote','on RandomOverSampling']
accuracies = [accuracy_score(y_test, y_pred),accuracy_score(y_test, y_pred_sm), accuracy_score(y_test, y_pred_ran)]
```

```
plt.figure(figsize=(10,5))
plt.barh(model_names, accuracies)
plt.xlabel('Accuracy')
plt.ylabel('Model')
plt.title('Model Performance Comparison (Accuracy)')
plt.xlim(0, 1)
plt.show()
```



ML: Assignment-7

Implement K-Means clustering, hierarchical and DBSCAN clustering for the customer segmentation and perform the comparative analysis of all the clustering algorithm.

https://www.kaggle.com/datasets/datascientistanna/customers-dataset

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
```

```
df = pd.read_csv("/content/Customers.csv")
df
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1	Male	19	15000	39	Healthcare	1	4
1	2	Male	21	35000	81	Engineer	3	3
2	3	Female	20	86000	6	Engineer	1	1
3	4	Female	23	59000	77	Lawyer	0	2
4	5	Female	31	38000	40	Entertainment	2	6
...	...	...	...	...	...	...	...	...
1995	1996	Female	71	184387	40	Artist	8	7
1996	1997	Female	91	73158	32	Doctor	7	7
1997	1998	Male	87	90961	14	Healthcare	9	2
1998	1999	Male	77	182109	4	Executive	7	2
1999	2000	Male	90	110610	52	Entertainment	5	2

2000 rows × 8 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              2000 non-null   int64 
 1   Gender                  2000 non-null   object
 2   Age                     2000 non-null   int64 
 3   Annual Income ($)       2000 non-null   int64 
 4   Spending Score (1-100)  2000 non-null   int64 
 5   Profession              1965 non-null   object
 6   Work Experience         2000 non-null   int64 
 7   Family Size             2000 non-null   int64 
dtypes: int64(6), object(2)
memory usage: 125.1+ KB
```

```
df.describe()
```

	CustomerID	Age	Annual Income ($)	Spending Score (1-100)	Work Experience	Family Size
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1000.500000	48.960000	110731.821500	50.962500	4.102500	3.768500
std	577.494589	28.429747	45739.536688	27.934661	3.922204	1.970749
min	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	500.750000	25.000000	74572.000000	28.000000	1.000000	2.000000
50%	1000.500000	48.000000	110045.000000	50.000000	3.000000	4.000000
75%	1500.250000	73.000000	149092.750000	75.000000	7.000000	5.000000
max	2000.000000	99.000000	189974.000000	100.000000	17.000000	9.000000

```
df.shape
```

```
(2000, 8)
```

```
df.columns
```

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income ($)',
       'Spending Score (1-100)', 'Profession', 'Work Experience',
       'Family Size'],
      dtype='object')
```

```
df.isnull().sum()
```

	0
CustomerID	0
Gender	0
Age	0
Annual Income ($)	0
Spending Score (1-100)	0
Profession	35
Work Experience	0
Family Size	0

Preprocessing

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df['Profession'] = le.fit_transform(df['Profession'])

df
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1	1	19	15000	39	5	1	4
1	2	1	21	35000	81	2	3	3
2	3	0	20	86000	6	2	1	1
3	4	0	23	59000	77	7	0	2
4	5	0	31	38000	40	3	2	6
...	...	...	...	...	...	...	...	...
1995	1996	0	71	184387	40	0	8	7
1996	1997	0	91	73158	32	1	7	7
1997	1998	1	87	90961	14	5	9	2
1998	1999	1	77	182109	4	4	7	2
1999	2000	1	90	110610	52	3	5	2

2000 rows × 8 columns

```
df = df.drop(['CustomerID'], axis=1)
```

```
df.shape
```

```
(2000, 7)
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
x = sc.fit_transform(df)
```

K-Means Clustering

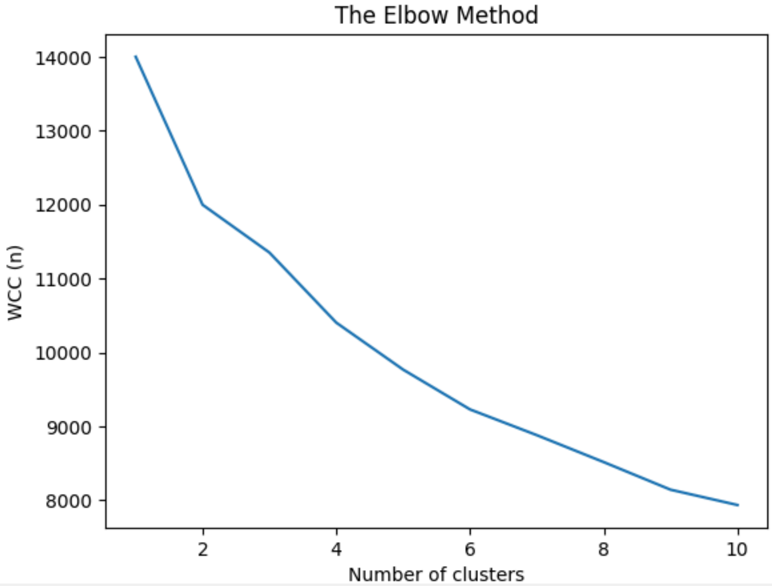
```
n=[]
for i in range (1,11):
  kmean = KMeans(n_clusters=i,random_state=0)
  kmean.fit(x)
  n.append(kmean.inertia_)
print(n)
```

```
[14000.000000000002, 11998.877359162923, 11351.651199656559, 10404.115639438844, 9768.690117600767, 9230.470525195193, 8881.9688048
```

```
import matplotlib.pyplot as plt
```

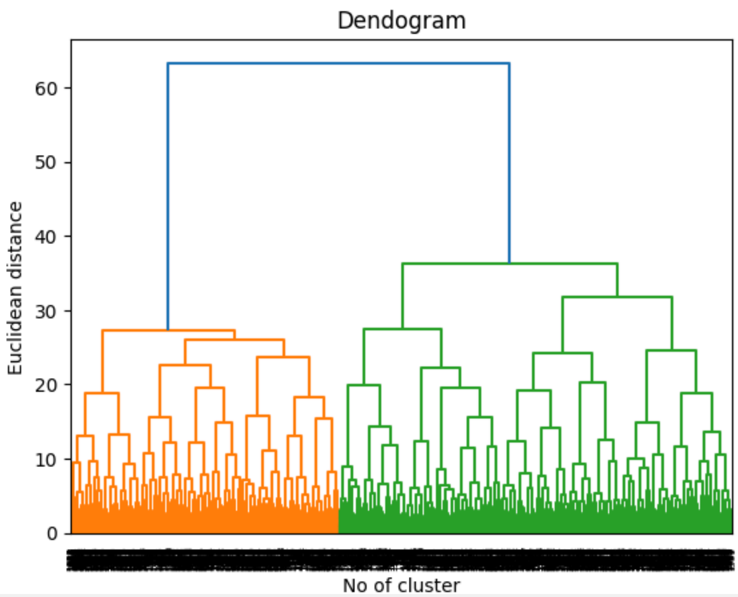
```
plt.plot(range(1,11),n)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCC (n)')
```

```
Text(0, 0.5, 'WCC (n)')
```




```
import scipy.cluster.hierarchy as sch

dendo = sch.dendrogram(sch.linkage(x, method='ward'))
plt.title("Dendogram")
plt.xlabel("No of cluster")
plt.ylabel("Euclidean distance")
```

```
Text(0, 0.5, 'Euclidean distance')
```




```
model = KMeans(2,random_state=0)
model.fit(x)
```

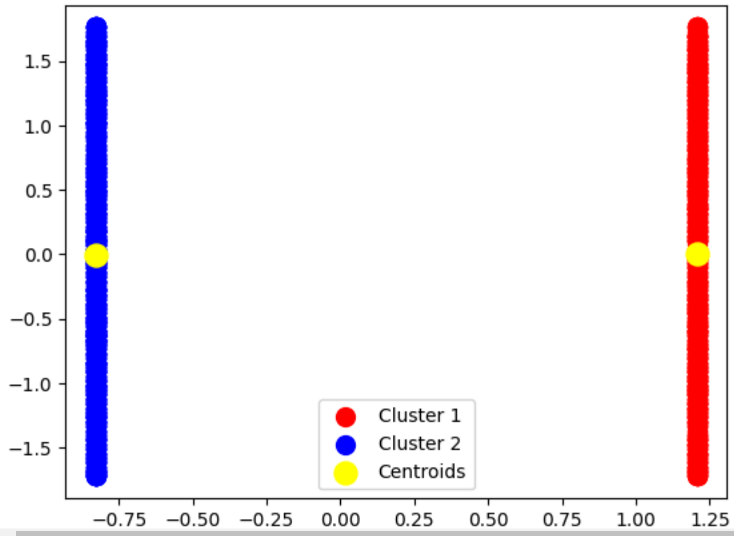
```
y = model.predict(x)
y
```

```
array([0, 0, 1, ..., 0, 0, 0], dtype=int32)
```

```
from sklearn.metrics import silhouette_score
score1 = silhouette_score(x, y)
score1
```

```
0.15323739617718343
```

```
plt.scatter(x[y == 0, 0], x[y == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(x[y == 1, 0], x[y == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1],c='yellow',s=150,label='Centroids')
plt.legend()
plt.show()
```



This value is chosen because its the highest score among the all values for k. Value suggest poor performaance of the model and overlapping of clusters.

Hierarchical Clustering: AgglomerativeClustering

```
from sklearn.cluster import AgglomerativeClustering
```

```
model = AgglomerativeClustering(n_clusters=2,linkage='ward')  #affinity='euclidean'

y_2 = model.fit_predict(x)
y_2
```

```
array([1, 1, 0, ..., 1, 1, 1])
```

```
score2 = silhouette_score(x,y_2)
score2
```

```
0.15323739617718343
```

DBSCAN

```
from sklearn.cluster import DBSCAN
```

```
model = DBSCAN(eps=1.5, min_samples=8) # - means noice
y_3 = model.fit_predict(x)
y_3
```

```
array([ 0,  0,  1, ...,  0, -1,  0])
```

```
np.unique(y)
```

```
array([0, 1], dtype=int32)
```

```
n_data = pd.DataFrame(y)
n_data.value_counts()
```

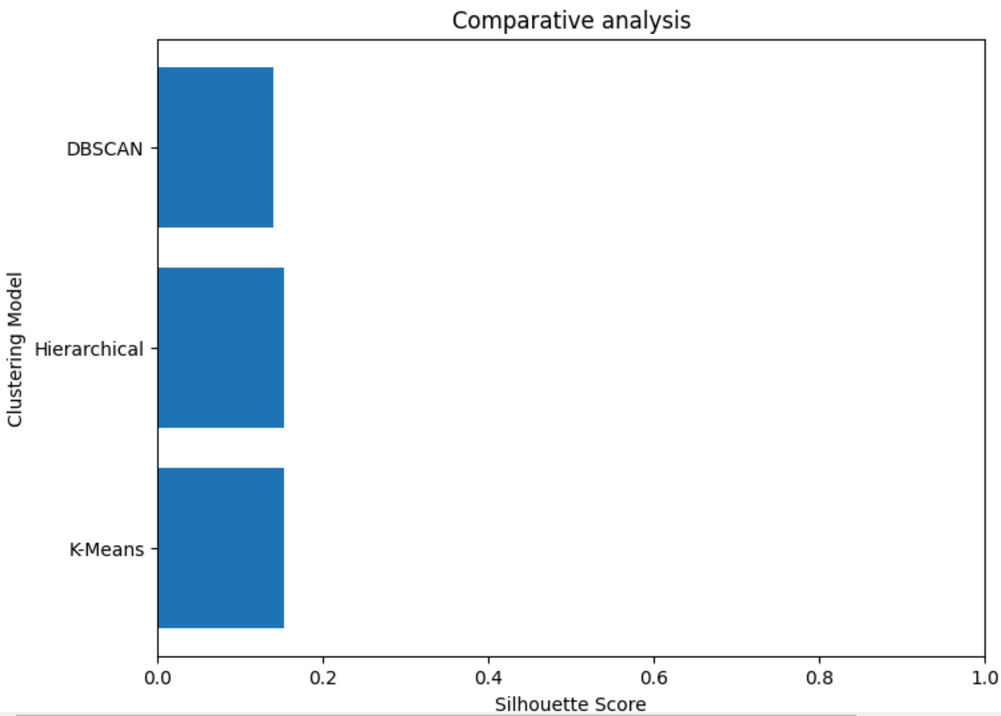
	count
0	
1	1186
0	814

```
score3 = silhouette_score(x,y_3)
score3
```

```
0.14020464701529012
```

Comparative analysis

```
scores = [score1, score2, score3]
model_names = ['K-Means', 'Hierarchical', 'DBSCAN']

plt.figure(figsize=(8, 6))
plt.barh(model_names, scores)
plt.ylabel('Clustering Model')
plt.xlabel('Silhouette Score')
plt.title('Comparative analysis')
plt.xlim(0, 1)

plt.show()
```



Hierarchical and K-Means Clustering silhouette score is same which is 0.1532, while DBSCAN clustering's silhouette score is slighly lesser than Hierarchical and K-Means Clustering which is 0.14020.

ML: ASSIGNMENT - 8

Apply Principle Component Analysis (PCA) method to reduce the dimension of a dataset and then perform the comparative analysis before and after applying PCA on the below dataset.

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
```

```
df = pd.read_csv("/content/bank_transactions.csv")
```

```
df
```

	TransactionID	CustomerID	CustomerDOB	CustGender	CustLocation	CustAcc
0	T1	C5841053	10/1/94	F	JAMSHEDPUR	
1	T2	C2142763	4/4/57	M	JHAJJAR	
2	T3	C4417068	26/11/96	F	MUMBAI	
3	T4	C5342380	14/9/73	F	MUMBAI	
4	T5	C9031234	24/3/88	F	NAVI MUMBAI	
...	...	...	...	...	...	
1048562	T1048563	C8020229	8/4/90	M	NEW DELHI	
1048563	T1048564	C6459278	20/2/92	M	NASHIK	
1048564	T1048565	C6412354	18/5/89	M	HYDERABAD	
1048565	T1048566	C6420483	30/8/78	M	VISAKHAPATNAM	
1048566	T1048567	C8337524	5/3/84	M	PUNE	

```
df.columns
```

```
Index(['TransactionID', 'CustomerID', 'CustomerDOB', 'CustGender',
       'CustLocation', 'CustAccountBalance', 'TransactionDate',
       'TransactionTime', 'TransactionAmount (INR)'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048567 entries, 0 to 1048566
Data columns (total 9 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   TransactionID            1048567 non-null  object
 1   CustomerID               1048567 non-null  object
 2   CustomerDOB              1045170 non-null  object
 3   CustGender               1047467 non-null  object
 4   CustLocation             1048416 non-null  object
 5   CustAccountBalance       1046198 non-null  float64
 6   TransactionDate          1048567 non-null  object
 7   TransactionTime          1048567 non-null  int64
 8   TransactionAmount (INR)  1048567 non-null  float64
dtypes: float64(2), int64(1), object(6)
memory usage: 72.0+ MB
```

```
df.describe()
```

	CustAccountBalance	TransactionTime	TransactionAmount (INR)
count	1.046198e+06	1.048567e+06	1.048567e+06
mean	1.154035e+05	1.570875e+05	1.574335e+03
std	8.464854e+05	5.126185e+04	6.574743e+03
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	4.721760e+03	1.240300e+05	1.610000e+02
50%	1.679218e+04	1.642260e+05	4.590300e+02
75%	5.765736e+04	2.000100e+05	1.200000e+03
max	1.150355e+08	2.359590e+05	1.560035e+06

PREPROCESSING

```
df.drop(['CustomerID','TransactionID'],axis=1,inplace=True)
df
```

	CustomerDOB	CustGender	CustLocation	CustAccountBalance	TransactionDa
0	10/1/94	F	JAMSHEDPUR	17819.05	2/8/
1	4/4/57	M	JHAJJAR	2270.69	2/8/
2	26/11/96	F	MUMBAI	17874.44	2/8/
3	14/9/73	F	MUMBAI	866503.21	2/8/
4	24/3/88	F	NAVI MUMBAI	6714.43	2/8/
...	...	...	...	...	
1048562	8/4/90	M	NEW DELHI	7635.19	18/9/
1048563	20/2/92	M	NASHIK	27311.42	18/9/
1048564	18/5/89	M	HYDERABAD	221757.06	18/9/
1048565	30/8/78	M	VISAKHAPATNAM	10117.87	18/9/
1048566	5/3/84	M	PUNE	75734.42	18/9/

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048567 entries, 0 to 1048566
Data columns (total 7 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   CustomerDOB              1045170 non-null  object
 1   CustGender               1047467 non-null  object
 2   CustLocation             1048416 non-null  object
 3   CustAccountBalance       1046198 non-null  float64
 4   TransactionDate          1048567 non-null  object
 5   TransactionTime          1048567 non-null  int64
 6   TransactionAmount (INR)  1048567 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 56.0+ MB
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['CustGender'] = le.fit_transform(df['CustGender'])
df['CustLocation'] = le.fit_transform(df['CustLocation'])
```

```
df['CustomerDOB'] = pd.to_datetime(df['CustomerDOB'])
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])
df.head()
```

	CustomerDOB	CustGender	CustLocation	CustAccountBalance	TransactionDate	Transa
0	1994-10-01	0	3586	17819.05	2016-02-08	
1	2057-04-04	1	3648	2270.69	2016-02-08	
2	1996-11-26	0	5268	17874.44	2016-02-08	
3	2073-09-14	0	5268	866503.21	2016-02-08	

LETS CREATE A NEW FEATURE USING CustomerDOB AND TransactionDate AND DROP DATES COLS

```
df['Age'] = (df['TransactionDate'] - df['CustomerDOB']).dt.days // 365
```

```
df.head()
```

	CustomerDOB	CustGender	CustLocation	CustAccountBalance	TransactionDate	Transa
0	1994-10-01	0	3586	17819.05	2016-02-08	
1	2057-04-04	1	3648	2270.69	2016-02-08	
2	1996-11-26	0	5268	17874.44	2016-02-08	
3	2073-09-14	0	5268	866503.21	2016-02-08	

SOME ENTRIES OF AGE WERE IN -VE BECAUSE OF WRONG DATA SO REMOVE THOSE ROWS WITH -VE AGE

```
df.shape
```

```
(1048567, 8)
```

```
df = df[df['Age'] >= 0]
df.shape
```

```
(950897, 8)
```

```
df.isnull().sum()
```

	0
CustomerDOB	0
CustGender	0
CustLocation	0
CustAccountBalance	2168
TransactionDate	0
TransactionTime	0
TransactionAmount (INR)	0
Age	0

dtype: int64

```
df.fillna(df.ffill(),inplace=True)
df.isnull().sum()
```

	0
CustomerDOB	0
CustGender	0
CustLocation	0
CustAccountBalance	0
TransactionDate	0
TransactionTime	0
TransactionAmount (INR)	0
Age	0

dtype: int64

```
df = df.drop(columns=['CustomerDOB', 'TransactionDate'])
df.head()
```

	CustGender	CustLocation	CustAccountBalance	TransactionTime	TransactionAmount (INR)
0	0	3586	17819.05	143207	25.0
2	0	5268	17874.44	142712	459.0
4	0	5657	6714.43	181156	1762.5
6	0	5268	973.46	173806	566.0

```
from sklearn.preprocessing import StandardScaler
cols_to_scale = ['CustAccountBalance', 'TransactionTime', 'TransactionAmount (INR)']
scaler = StandardScaler()
df[cols_to_scale] = scaler.fit_transform(df[cols_to_scale])
```

```
df
```

	CustGender	CustLocation	CustAccountBalance	TransactionTime	TransactionAm (
0	0	3586	-0.153331	-0.272096	-0.23
2	0	5268	-0.153215	-0.281681	-0.16
4	0	5657	-0.176692	0.462755	0.05
6	0	5268	-0.188769	0.320429	-0.14
7	1	5268	0.009192	0.257127	-0.21
...	...	...	...	...	
1048562	1	5792	-0.174755	0.533783	-0.10
1048563	1	5629	-0.133363	0.512676	-0.16
1048564	1	3394	0.275689	0.504524	-0.10
1048565	1	9137	-0.169532	0.531498	-0.07
1048566	1	6719	-0.031496	0.464033	-0.04

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

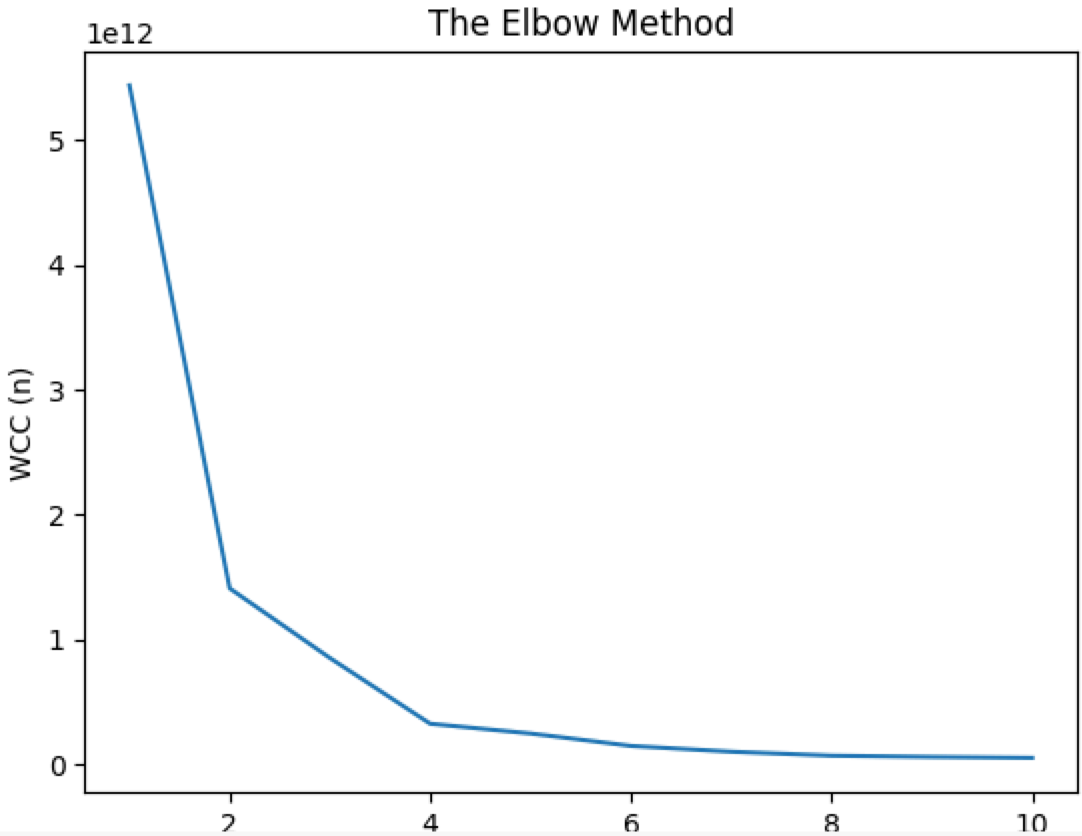
```
n=[]
for i in range (1,11):
  kmean = KMeans(n_clusters=i,random_state=0)
  kmean.fit(df)
  n.append(kmean.inertia_)
print(n)
```

```
[5439315157803.642, 1410258022086.181, 851837844535.1045, 324900554289.9498, 24662892
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(range(1,11),n)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCC (n)')
```

```
Text(0, 0.5, 'WCC (n)')
```




```
kmeans = KMeans(n_clusters=4, random_state=42)
pred = kmeans.fit_predict(df)
```

```
silhouette_before = silhouette_score(df, pred)
silhouette_before
```

```
0.6401222255583267
```

With PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.90)
x_pca = pca.fit_transform(df)
```

```
pca.explained_variance_ratio_
```

```
array([0.9979168])
```

```
x_pca.shape
```

```
(90329, 1)
```

```
n=[]
for i in range (1,11):
  kmean = KMeans(n_clusters=i,random_state=0)
  kmean.fit(x_pca)
  n.append(kmean.inertia_)
print(n)
```

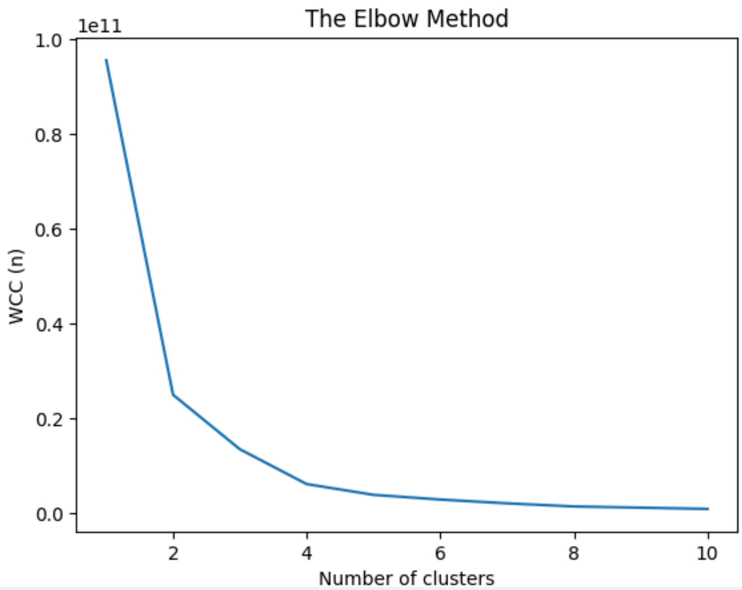
```
[95563327234.497, 24967905936.863937, 13476285026.362082, 6120359303.33825, 3851363298.5731835, 2857683502.3422747, 2058027290.9685013, 1395533150.17107, 114319534
```

```
[95563327234.497, 24967905936.863937, 13476285026.362082, 6120359303.33825, 3851363298.5731835, 2857683502.3422747, 2058027290.9685
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(range(1,11),n)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCC (n)')
```

```
Text(0, 0.5, 'WCC (n)')
```




```
kmeans = KMeans(n_clusters=4, random_state=42)
pred = kmeans.fit_predict(x_pca)
```

```
silhouette_before = silhouette_score(x_pca, pred)
```

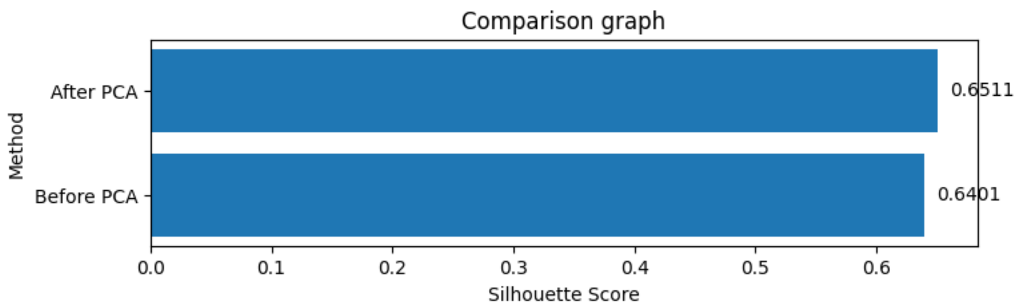
```
silhouette_after_pca = silhouette_before
```

```
silhouette_after_pca
```

```
0.6511111995729372
```

```
silhouette_before_pca = 0.6401222255583267
```

```
values = [silhouette_before_pca, silhouette_after_pca]
labels = ['Before PCA', 'After PCA']
plt.figure(figsize=(8, 2))
plt.barh(labels, values)
plt.title('Comparison graph')
plt.xlabel('Silhouette Score')
plt.ylabel('Method')
for i, v in enumerate(values):
    plt.text(v + 0.01, i, f"{v:.4f}", va='center')
plt.show()
```



Start coding or generate with AI.