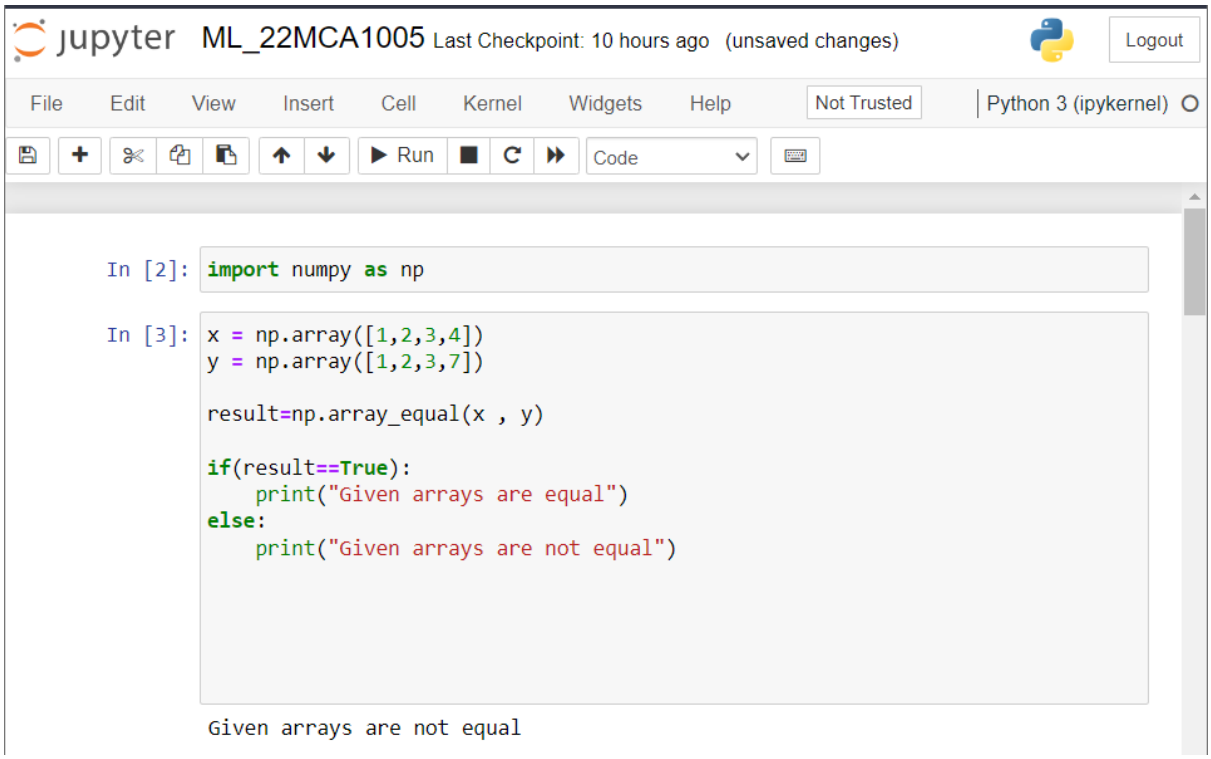
NAME – VIPUL SHRIRAM TAPARE

REGISTER NO – 22MCA1005

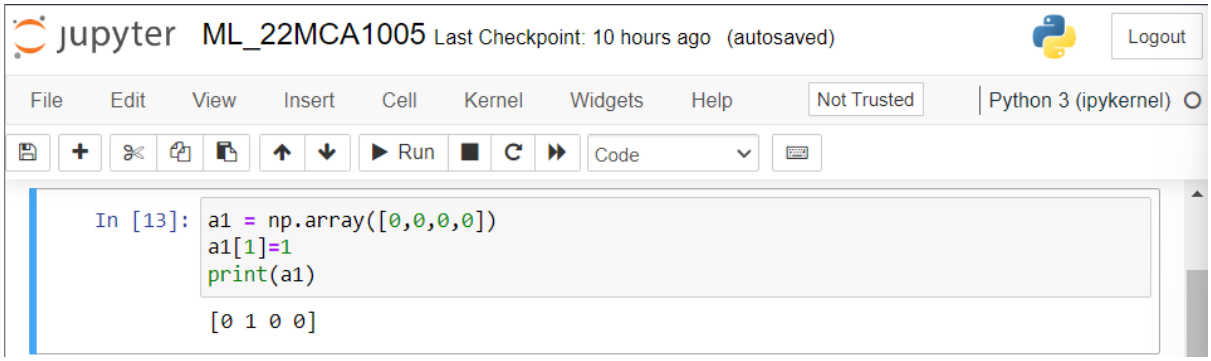
FACULTY NAME – DR. RAJESH M.

MACHINE LEARNING LAB01

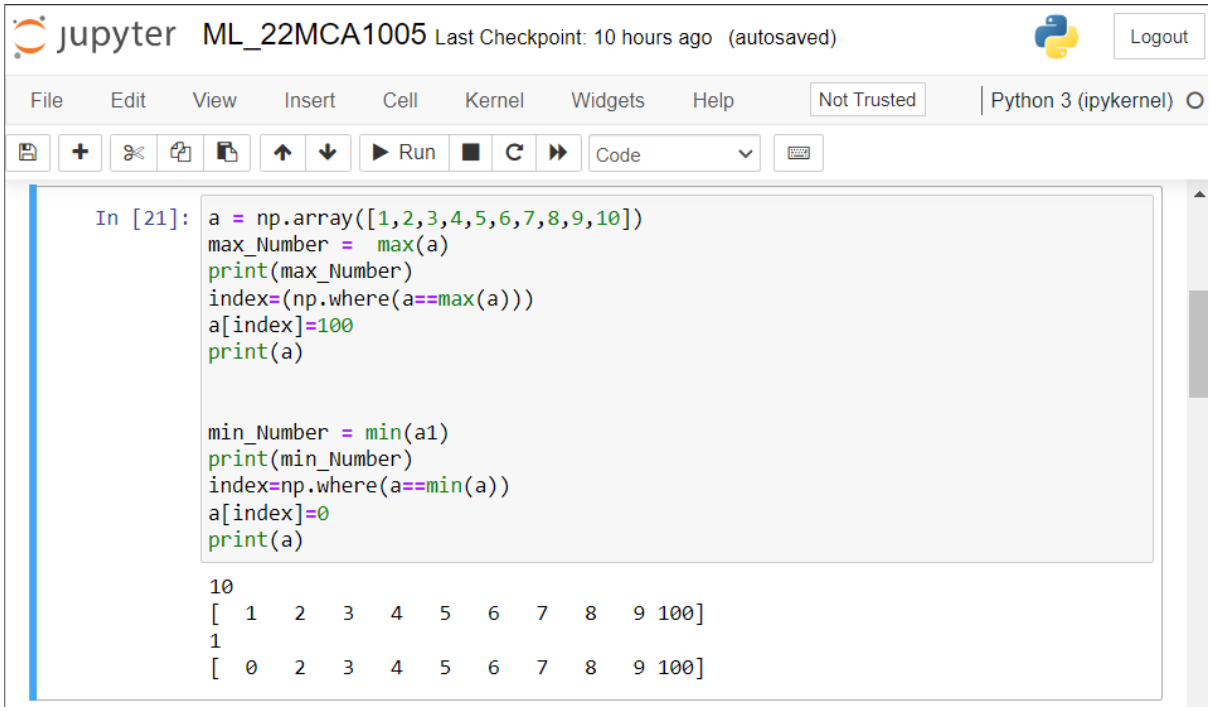
1) **Create two random numpy array of integers X and Y; check whether they are equal or not?**



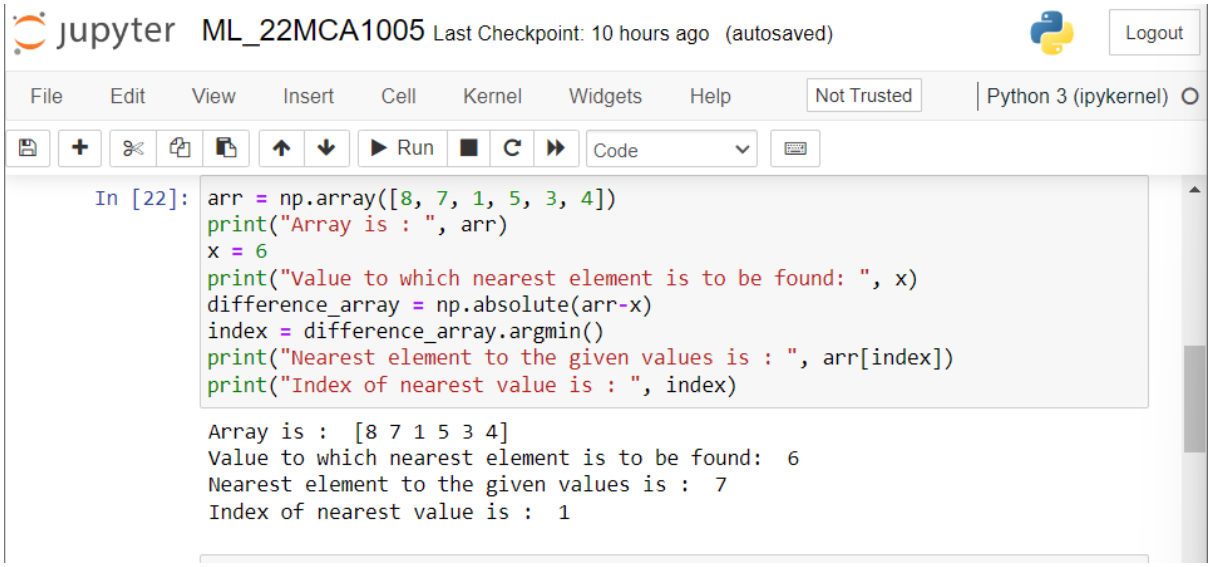
2) Create a numpy array with zeros and make it immutable and check whether able to change the values or not.



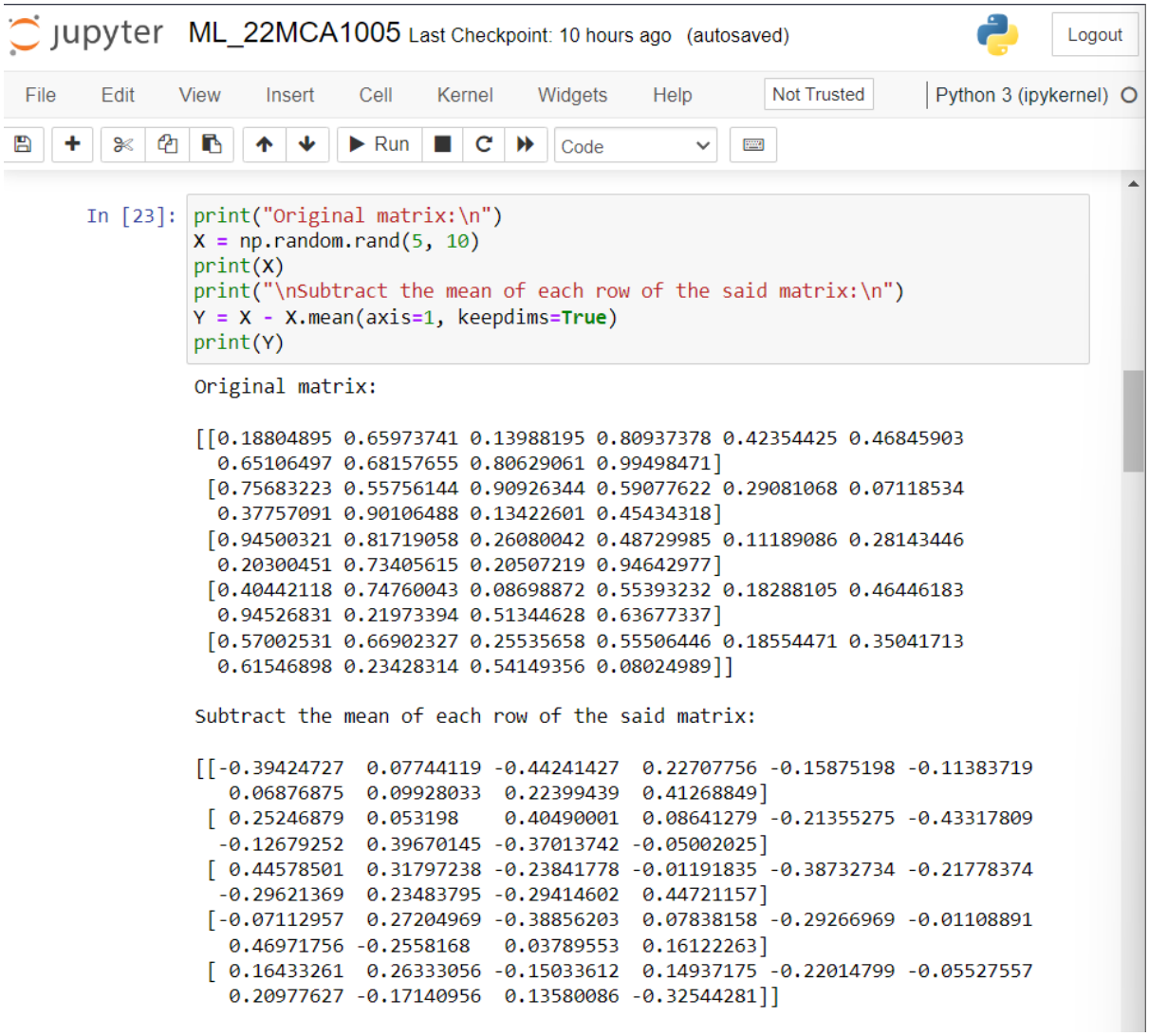
3) Create a random numpy array Y of integers with size '10' and replace the maximum value by '100' and minimum value by '0'.



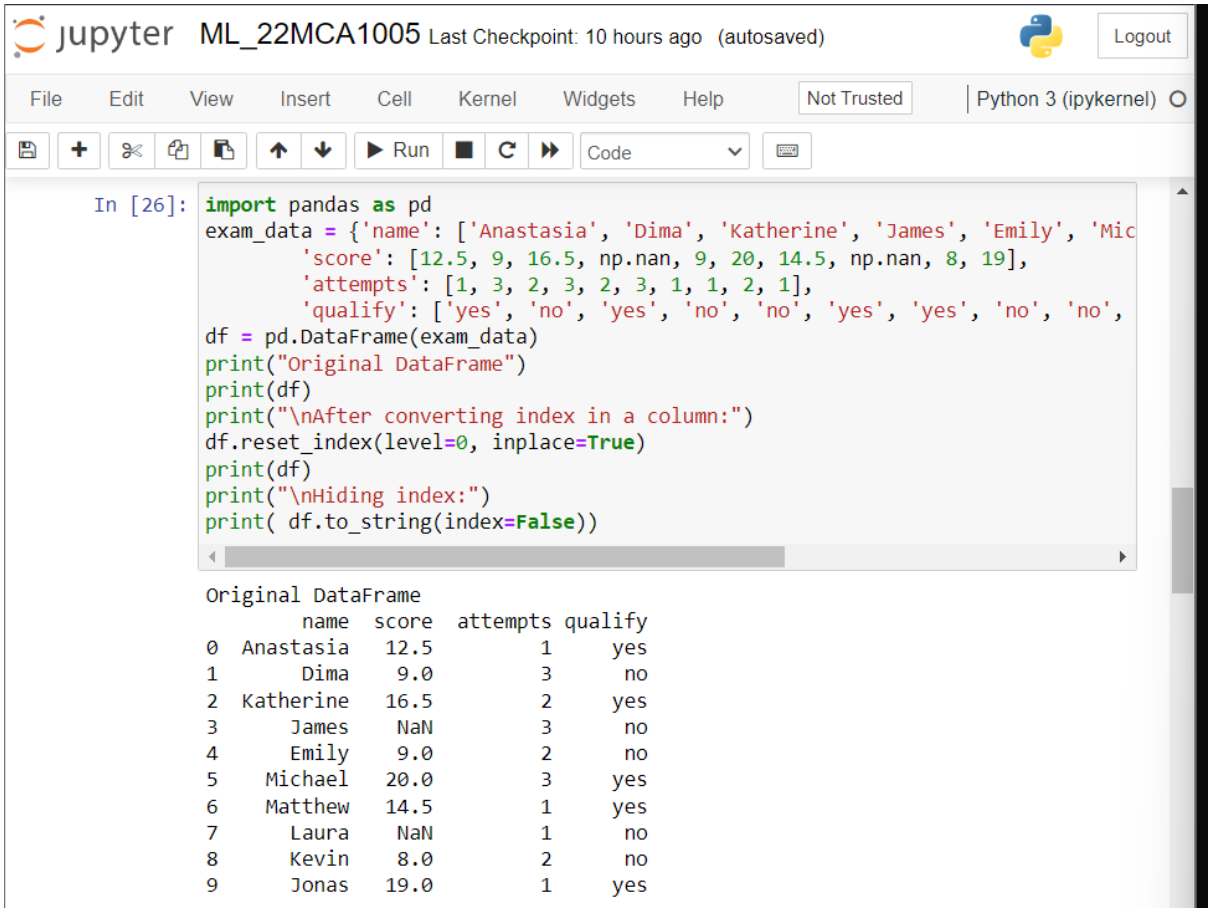
4) Write a program to find the closest value to a given value from a numpy array.

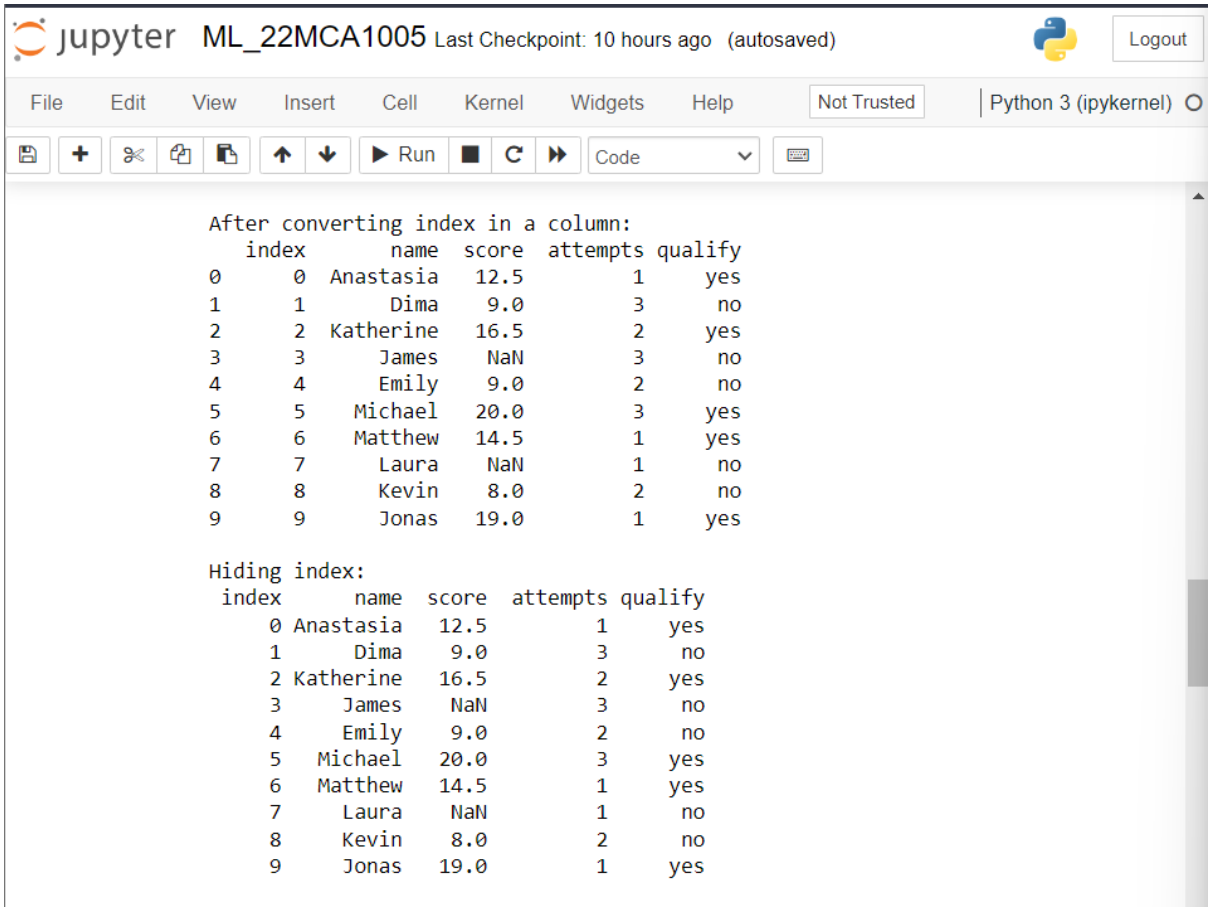


5) Write a program to subtract the mean of each row of a matrix.

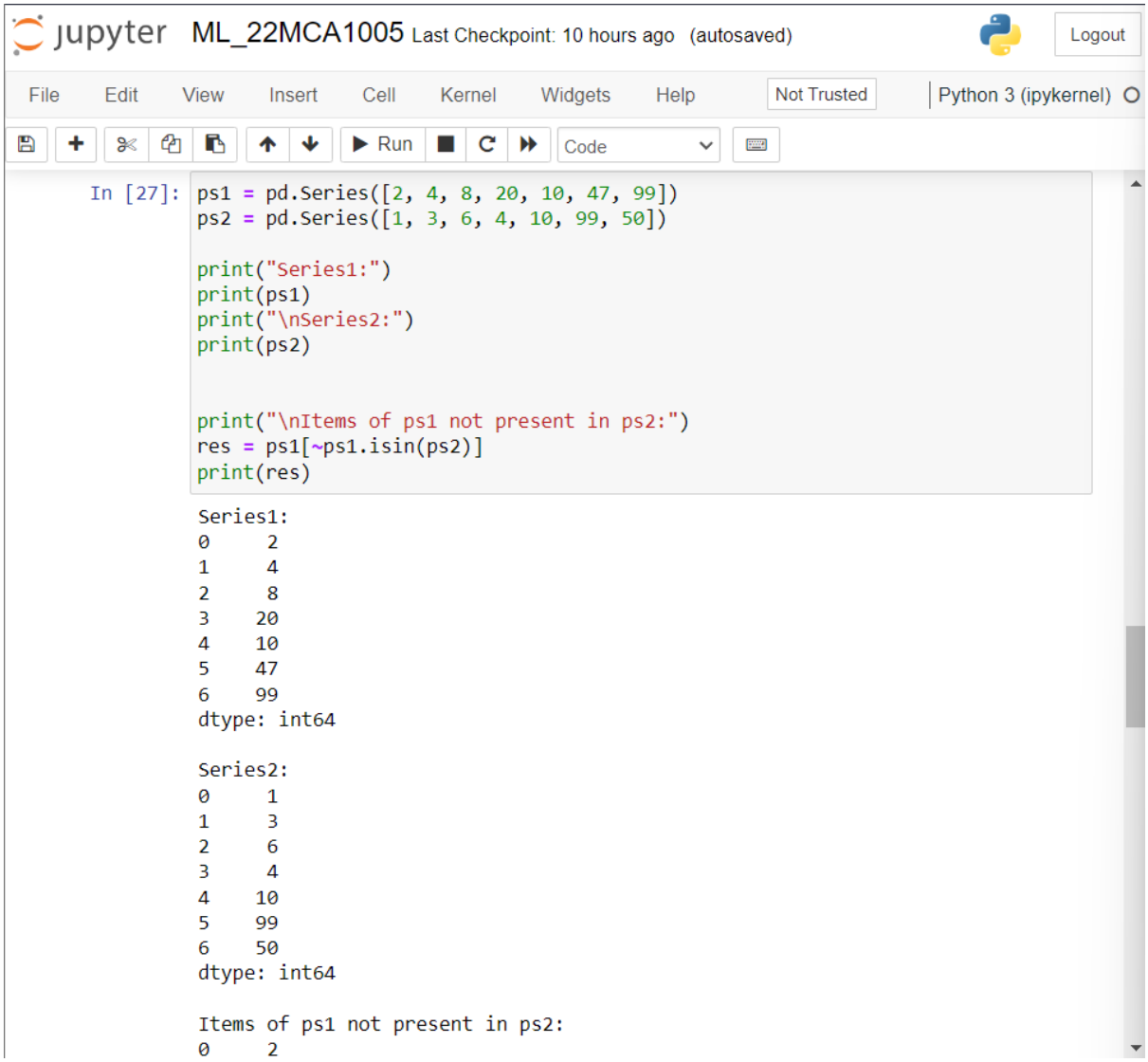


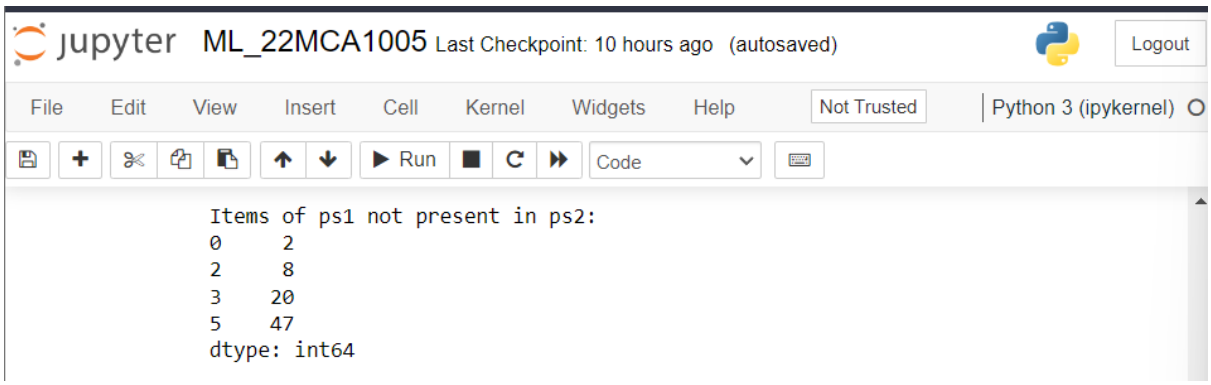
6) Write a program to convert the index of a series into a column of a dataframe.



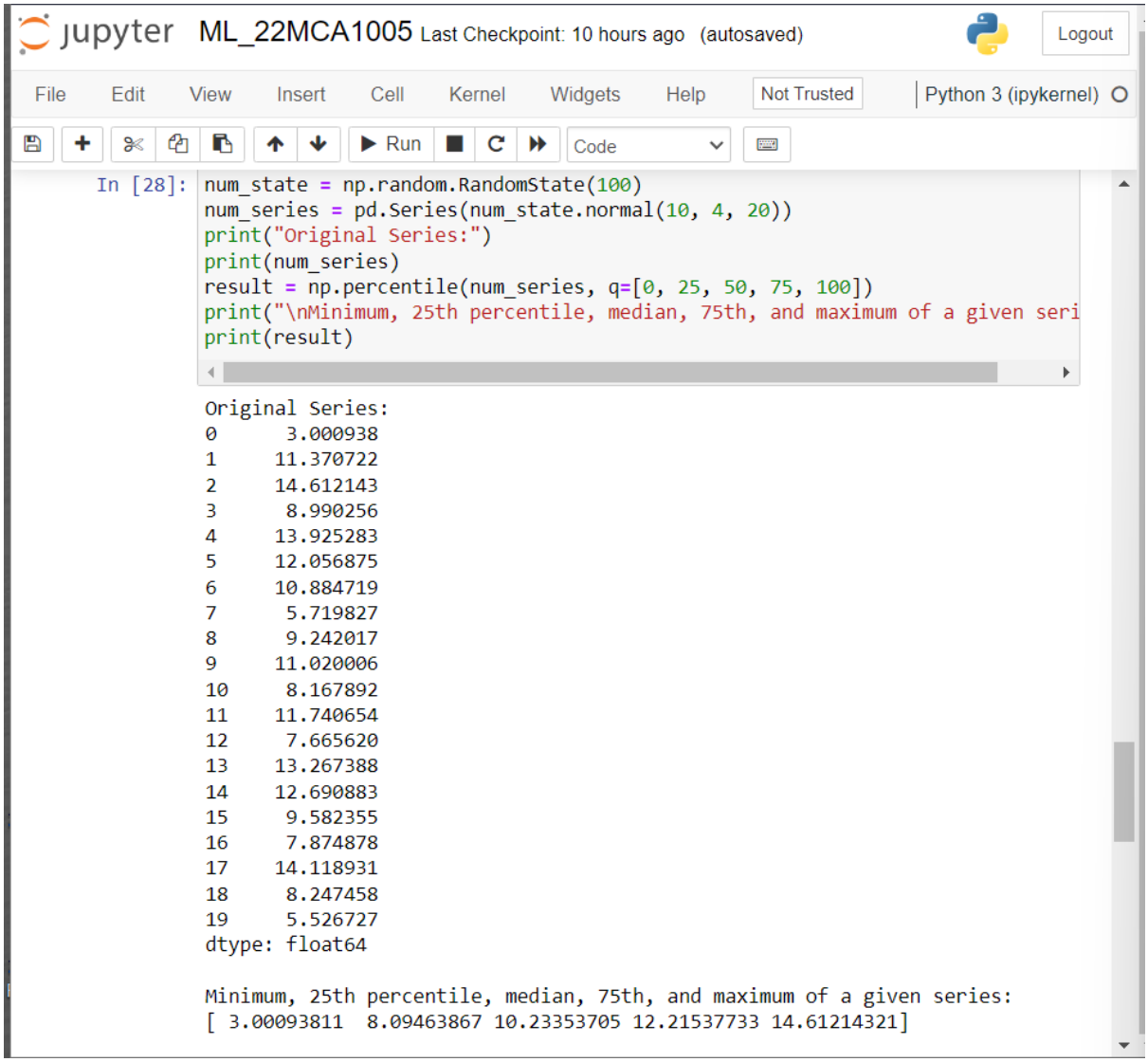

jupyter ML_22MCA1005 Last Checkpoint: 10 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
After converting index in a column:
index name score attempts qualify
0 0 Anastasia 12.5 1 yes
1 1 Dima 9.0 3 no
2 2 Katherine 16.5 2 yes
3 3 James NaN 3 no
4 4 Emily 9.0 2 no
5 5 Michael 20.0 3 yes
6 6 Matthew 14.5 1 yes
7 7 Laura NaN 1 no
8 8 Kevin 8.0 2 no
9 9 Jonas 19.0 1 yes
Hiding index:
index name score attempts qualify
0 Anastasia 12.5 1 yes
1 Dima 9.0 3 no
2 Katherine 16.5 2 yes
3 James NaN 3 no
4 Emily 9.0 2 no
5 Michael 20.0 3 yes
6 Matthew 14.5 1 yes
7 Laura NaN 1 no
8 Kevin 8.0 2 no
9 Jonas 19.0 1 yes

7) Write a program to find out the items of series X not present in series Y?

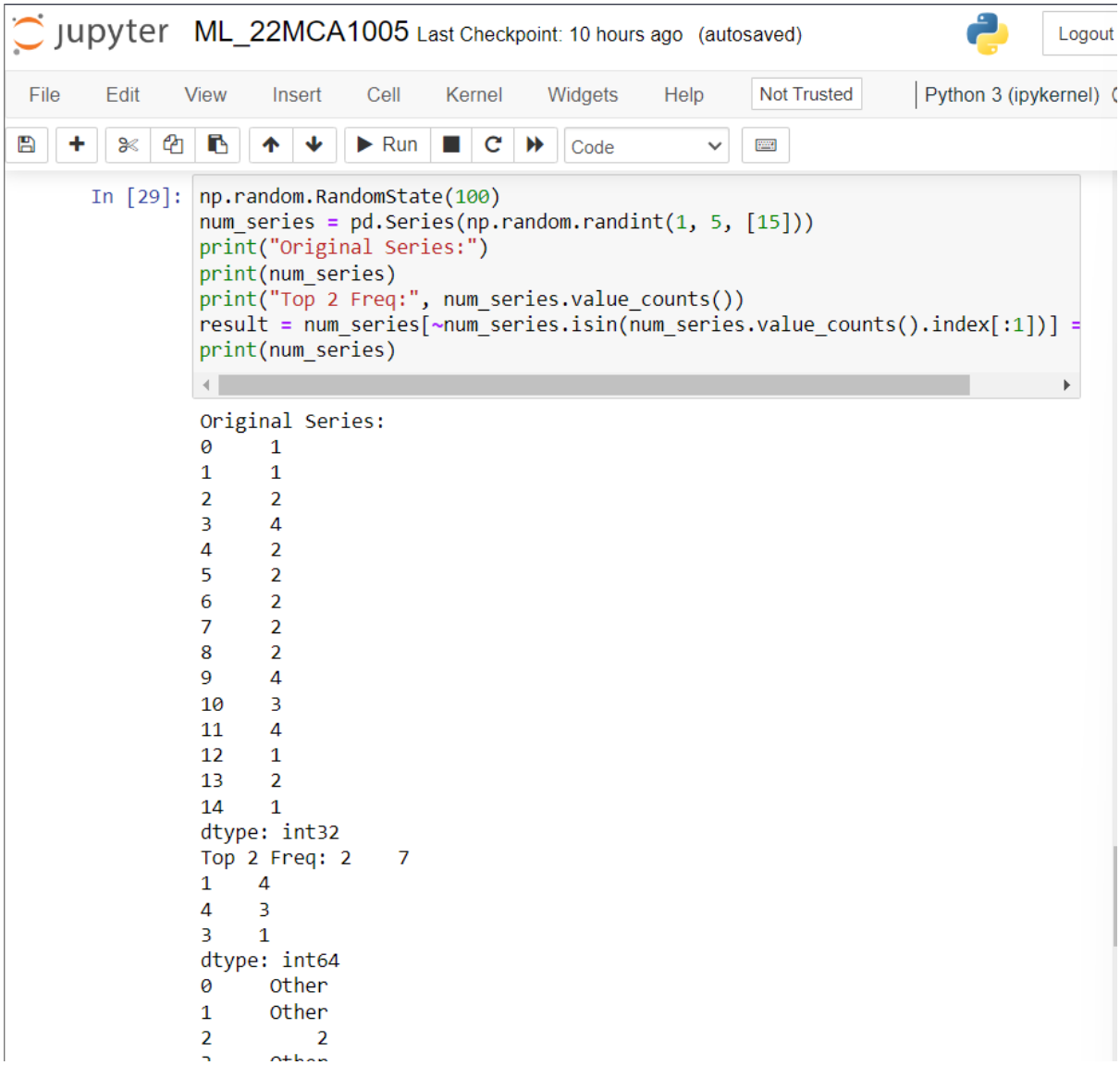


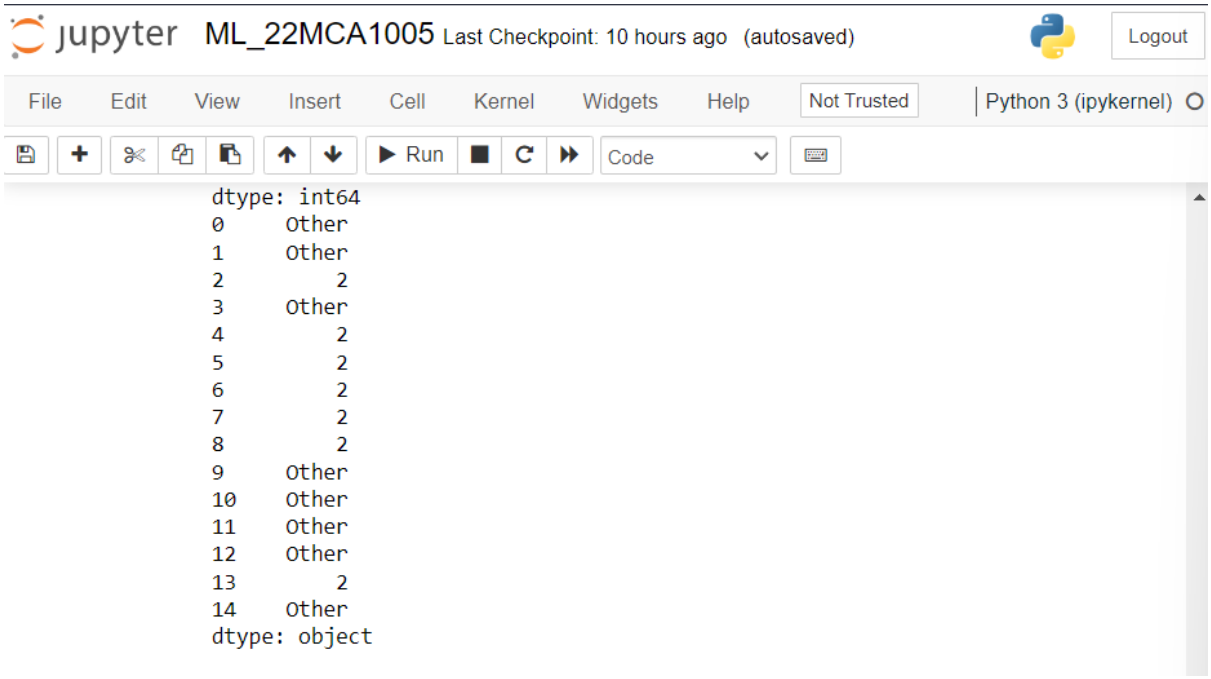


8) Write a program to find out the minimum , 25th percentile , median , 75th and max of a numeric series?

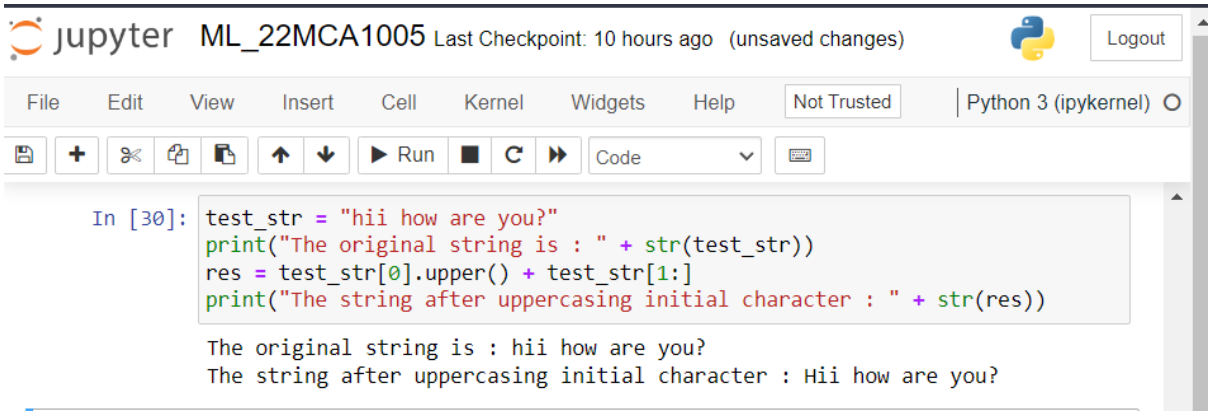


9) Write a program to keep only top 2 most frequent values as it is and replace everything else as 'other'?




jupyter ML_22MCA1005 Last Checkpoint: 10 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
dtype: int64
0 Other
1 Other
2 2
3 Other
4 2
5 2
6 2
7 2
8 2
9 Other
10 Other
11 Other
12 Other
13 2
14 Other
dtype: object

10) Write a program to convert the first character of each element in a series to uppercase.

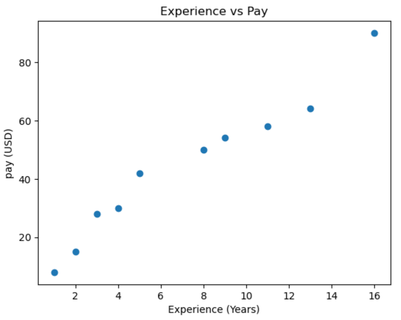


Machine Learning Lab -02

22MCA1005

QUESTION NO 1

In [38]:
```
import matplotlib.pyplot as plt

experience = [[2],[3],[5],[13],[8],[16],[11],[1],[9],[4]]
pay = [[15], [28], [42], [64],[50],[90],[58],[8],[54],[30]]

plt.scatter(experience , pay)

plt.xlabel("Experience (Years)")
plt.ylabel("pay (USD)")
plt.title("Experience vs Pay")
plt.show()
```



QUESTION NO 2

In [39]:
```
import numpy as np
import pandas as pd
x = np.array([2,3,5,13,8,16,11,1,9,4])
y = np.array([15,28,42,64,50,90,58,8,54,30])

xbar = np.mean(x)
ybar = np.mean(y)
print(xbar)
print(ybar)

xMinusxbar = x - xbar
print(xMinusxbar)
yMinusybar = y - ybar
print(yMinusybar)

xMinusxbarMyMinusybar = xMinusxbar * yMinusybar
print(xMinusxbarMyMinusybar)

xsquare = xMinusxbar*xMinusxbar
print(xsquare)

table = pd.DataFrame({'x':x , 'y':y , 'xMinusxbar':xMinusxbar , 'yMinusybar':yMinusybar, 'xMinusxbarMyMinusybar':xMinusxbarMyMinusybar ,'xsquare':xsquare})
print(table)
```

```
7.2
43.9
[-5.2 -4.2 -2.2  5.8  0.8  8.8  3.8 -6.2  1.8 -3.2]
[-28.9 -15.9  -1.9  20.1   6.1  46.1  14.1 -35.9  10.1 -13.9]
[150.28  66.78   4.18 116.58   4.88 405.68  53.58 222.58  18.18  44.48]
[27.04 17.64  4.84 33.64  0.64 77.44 14.44 38.44  3.24 10.24]
   x   y  xMinusxbar  yMinusybar  xMinusxbarMyMinusybar  xsquare
0  2  15        -5.2       -28.9                 150.28    27.04
1  3  28        -4.2       -15.9                  66.78    17.64
2  5  42        -2.2        -1.9                   4.18     4.84
3  13  64         5.8        20.1                 116.58    33.64
4  8  50         0.8         6.1                   4.88     0.64
5  16  90         8.8        46.1                 405.68    77.44
6  11  58         3.8        14.1                  53.58    14.44
7  1  8        -6.2       -35.9                 222.58    38.44
8  9  54         1.8        10.1                  18.18     3.24
9  4  30        -3.2       -13.9                  44.48    10.24
```

QUESTION NO 3

In [40]:
```
xysum = np.sum(xMinusxbarMyMinusybar)
xsqrsum = np.sum(xsquare)
w1 = xysum/xsqrsum
print(w1)
w0 = ybar - (w1*xbar)
print(w0)
```

```
4.776801405975395
9.50702987697715
```

QUESTION NO 4

In [41]:
```
import numpy as np
x = np.array([2, 3, 5, 13, 8, 16, 11, 1, 9, 4])
y = np.array([15, 28, 42, 64, 50, 90, 58, 8, 54, 30])
mean_x = np.mean(x)
mean_y = np.mean(y)
xy_diff_product_sum = np.sum((x - mean_x) * (y - mean_y))
x_diff_squared_sum = np.sum((x - mean_x)**2)
w1 = xy_diff_product_sum / x_diff_squared_sum
w0 = mean_y - w1 * mean_x
print("w0 =", w0)
print("w1 =", w1)
```

```
w0 = 9.50702987697715
w1 = 4.776801405975395
```

QUESTION NO 5

In [42]:
```
import numpy as np
x = np.array([2, 3, 5, 13, 8, 16, 11, 1, 9, 4])
y = np.array([15, 28, 42, 64, 50, 90, 58, 8, 54, 30])
X = np.vstack((np.ones(len(x)), x)).T
coeffs = np.linalg.inv(X.T @ X) @ X.T @ y
w0 = coeffs[0]
w1 = coeffs[1]
print("w0 =", w0)
print("w1 =", w1)
```

```
w0 = 9.507029876977143
w1 = 4.776801405975396
```

QUESTION NO 6

In [43]:
```
 import numpy as np
x = np.array([2, 3, 5, 13, 8, 16, 11, 1, 9, 4])
y = np.array([15, 28, 42, 64, 50, 90, 58, 8, 54, 30])
alpha = 0.01
num_iterations = 1000
w0 = 0.0
w1 = 0.0
m = float(len(x))

for i in range(num_iterations):
            y_p = w0 + w1 * x
            errors = y_p - y
            grad_w0 = (1/m) * np.sum(errors)
            grad_w1 = (1/m) * np.sum(errors * x)
            w0 = w0 - alpha * grad_w0
            w1 = w1 - alpha * grad_w1
print("w0 =", w0)
print("w1 =", w1)
```

```
w0 = 9.072928403325314
w1 = 4.818869050208264
```

QUESTION NO 7

In [44]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

data = {'x': [2, 3, 5, 13, 8, 16, 11, 1, 9, 4],
'y': [15, 28, 42, 64, 50, 90, 58, 8, 54, 30]}
df = pd.DataFrame(data)

X_train, X_test, y_train, y_test = train_test_split(df[['x']],
df['y'],test_size=0.2)

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

y_pred = regressor.predict(X_test)

print("w0 (intercept) =", regressor.intercept_)
print("w1 (slope) =", regressor.coef_[0])
```

```
w0 (intercept) = 12.128816083395378
w1 (slope) = 4.536857781087119
```

QUESTION NO 8

In [45]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

data = {'x': [2, 3, 5, 13, 8, 16, 11, 1, 9, 4],
'y': [15, 28, 42, 64, 50, 90, 58, 8, 54, 30]}
df = pd.DataFrame(data)

X_train, X_test, y_train, y_test = train_test_split(df[['x']],
df['y'],test_size=0.2)
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print("Mean Absolute Error (MAE) =", mae)
print("Mean Squared Error (MSE) =", mse)
print("Root Mean Squared Error (RMSE) =", rmse)
print("R-squared (R^2) score =", r2)
```

```
Mean Absolute Error (MAE) = 3.2262284620293524
Mean Squared Error (MSE) = 13.903564829653032
Root Mean Squared Error (RMSE) = 3.728748426704734
R-squared (R^2) score = 0.9634357269437133
```

22MCA1005

Decision tree

In [1]:
```
import pandas as pd
import numpy as np
import math
```

In [2]:
```
data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 'Soft Skill', 'Soft Skill', 'Soft Skill', 'Hard Skill', 'Soft Skill'],
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'Selected', 'Rejected', 'Rejected', 'Selected']
})

print(data)
```

```
  Age Group Certified  Skill Type   Status
0       Old       Yes  Soft Skill  Rejected
1    Middle        No  Hard Skill  Selected
2    Middle       Yes  Soft Skill  Rejected
3     Young        No  Hard Skill  Selected
4    Middle       Yes  Hard Skill  Selected
5     Young       Yes  Soft Skill  Selected
6     Young        No  Soft Skill  Rejected
7       Old       Yes  Soft Skill  Rejected
8       Old        No  Hard Skill  Rejected
9    Middle        No  Soft Skill  Selected
```

1. Consider the following dataset and calculate the entropy and information gain w.r.t the target attribute named "Status".

In [3]:
```
status_counts = data['Status'].value_counts()
total_count = len(data)
entropy_status = 0
for count in status_counts:
    probability = count / num_instances
    entropy_status += -probability * math.log2(probability)
print(f"Entropy of Status: {entropy_status.3f}")

for attribute in ['Age Group', 'Certified', 'Skill Type']:
    print(f"\nAttribute: {attribute}")
    attribute_value_counts = data[attribute].value_counts()
    for value, count in attribute_value_counts.items():
        value_subset = data[data[attribute] == value]
        value_subset_size = len(value_subset)
        value_status_counts = value_subset['Status'].value_counts()
        value_entropy_status = 0
        for value_count in value_status_counts:
            value_probability = value_count / value_subset_size
            value_entropy_status += -value_probability * math.log2(value_probability)
        entropy_attribute += count / num_instances * value_entropy_status
    information_gain = entropy_status - entropy_attribute
    print(f"Information gain for {attribute}: {information_gain.3f}")
```

```
Entropy of Status: 1.000
Information gain of Age Group: 0.000
Information gain of Certified: 0.125
Information gain of Skill Type: 0.000
```

2. From the above calculated values of gain, design a decision tree for the above given data set.

In [4]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 'Soft Skill', 'Soft Skill', 'Soft Skill', 'Hard Skill', 'Soft Skill'],
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'Selected', 'Rejected', 'Rejected', 'Selected']
})

le = LabelEncoder()
data['Age Group'] = le.fit_transform(data['Age Group'])
data['Certified'] = le.fit_transform(data['Certified'])
data['Skill Type'] = le.fit_transform(data['Skill Type'])
data['Status'] = le.fit_transform(data['Status'])

dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

plt.figure(figsize=(12, 8))
plot_tree(dt, feature_names=X.columns, class_names=['Rejected', 'Selected'], filled=True)
plt.show()
```

3. Transform the designed decision tree into decision rules.

In [5]:
```
def tree_to_rules(tree, feature_names):
    rules = []
    for feature in feature_names:
        if feature != 'Status' and feature in data.columns:
            threshold = np.mean(data[data[feature] == value]['Status'])
            rule = f"{feature} == {value} => Status = {'Selected' if threshold > 0.5 else 'Rejected'}"
            rules.append(rule)
    return rules

rules = tree_to_rules(dt, data.columns[:-1])
for rule in rules:
    print(rule)
```

```
Age Group == 0 => Status = Rejected
Age Group == 1 => Status = Selected
Age Group == 2 => Status = Selected
Certified == 0 => Status = Selected
Certified == 1 => Status = Rejected
Skill Type == 0 => Status = Rejected
Skill Type == 1 => Status = Rejected
```

4. Use the designed decision tree or rules to predict the 'Status' of the given

employee = Young No Hard Skill • Old Yes Soft Skill • Middle Yes Hard Skill

In [6]:
```
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.preprocessing import LabelEncoder
import pandas as pd

data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 'Soft Skill', 'Soft Skill', 'Soft Skill', 'Hard Skill', 'Soft Skill'],
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'Selected', 'Rejected', 'Rejected', 'Selected']
})

le = LabelEncoder()
data['Age Group'] = le.fit_transform(data['Age Group'])
data['Certified'] = le.fit_transform(data['Certified'])
data['Skill Type'] = le.fit_transform(data['Skill Type'])
data['Status'] = le.fit_transform(data['Status'])

dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

tree_rules = export_text(dt, feature_names=X.columns.tolist())

print(tree_rules)
```

```
|--- Age Group <= 1.50
|   |--- Skill Type <= 0.50
|   |   |--- Certified <= 0.50
|   |   |   |--- class: 1
|   |   |--- Certified >  0.50
|   |   |   |--- class: 0
|   |--- Age Group >  1.50
|   |   |--- class: 0
|--- Age Group >  1.50
|   |--- class: 1
```

5. Design a function named find_entropy in python for finding the entropy of the attributes given in the above dataset.

In [7]:
```
import pandas as pd
import math

data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 'Soft Skill', 'Soft Skill', 'Soft Skill', 'Hard Skill', 'Soft Skill'],
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'Selected', 'Rejected', 'Rejected', 'Selected']
})

def find_entropy(df, attribute):

    value_counts = df[attribute].value_counts()

    total_instances = len(df)

    entropy = 0

    for value_count in value_counts:
        probability = value_count / total_instances
        entropy -= probability * math.log2(probability)

    return entropy

entropy = find_entropy(data, 'Age Group')
print(f"Entropy of 'Age Group': {entropy}")
```

```
Entropy of 'Age Group': 1.5709505944546684
```

6. Design a function named find_gain in python for finding the information gain of the attributes given in the above dataset w.r.t to the 'Status' attribute

In [8]:
```
import math

def find_entropy(df):

    entropy = 0
    num_records = len(df)
    status_values = df['Status'].unique()
    for s in status_values:
        num_s = len(df[df['Status']==s])
        p = num_s/num_records
        entropy -= p * math.log2(p)
    return entropy

def find_gain(df, attribute):

    total_entropy = find_entropy(df)
    values = df[attribute].unique()
    gain = total_entropy
    for v in values:
        num_v = len(df[df[attribute]==v])
        p = num_v/len(df.index)
        entropy_v = find_entropy(df[df[attribute]==v])
        gain = total_entropy - entropy
    return gain
find_entropy(data)
find_gain(data, 'Status')
```

Out[8]:
```
1.0
```

7. Load the above dataset as data frame in python

In [9]:
```
import pandas as pd

data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 'Soft Skill', 'Soft Skill', 'Soft Skill', 'Hard Skill', 'Soft Skill'],
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'Selected', 'Rejected', 'Rejected', 'Selected']
})
print(data)
```

```
  Age Group Certified  Skill Type   Status
0       Old       Yes  Soft Skill  Rejected
1    Middle        No  Hard Skill  Selected
2    Middle       Yes  Soft Skill  Rejected
3     Young        No  Hard Skill  Selected
4    Middle       Yes  Hard Skill  Selected
5     Young       Yes  Soft Skill  Selected
6     Young        No  Soft Skill  Selected
7       Old       Yes  Soft Skill  Rejected
8       Old        No  Hard Skill  Rejected
9    Middle        No  Soft Skill  Selected
```

8. Design and visualize the decision tree using scikit learn package for the given dataset.

In [10]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

model = DecisionTreeClassifier()

model.fit(X, y)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4,4), dpi=300)
tree.plot_tree(model, feature_names=X.columns, class_names['Not Selected', 'Selected'], filled=True)
```

Out[10]:
```
[Text(0.6666666666666666, 0.875, 'Age Group <= 1.5\nentropy = 1.0\nsamples = 10\nvalue = [5, 5]\nclass = Not Selected'),
 Text(0.5, 0.625, 'Age Group <= 0.5\nentropy = 0.469\nsamples = 5\nvalue = [1, 4]\nclass = Not Selected'),
 Text(0.3333333333333333, 0.375, 'Certified <= 0.5\nentropy = 0.918\nsamples = 3\nvalue = [1, 2]\nclass = Not Selected'),
 Text(0.16666666666666666, 0.125, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = Not Selected'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = Not Selected'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = Not Selected'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 5\nvalue = [4, 1]\nclass = Selected')]
```

9. Design and visualize the decision tree using scikit-learn package for the Irish(training) dataset from Kaggle.

In [11]:
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

iris = load_iris()

iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

iris_df['target'] = iris.target

X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]

tree = DecisionTreeClassifier()

tree.fit(X, y)

plot_tree(tree)
```

Out[11]:
```
[Text(0.5, 0.9166666666666666, 'X[3] <= 0.8\nentropy = 0.667\nsamples = 150\nvalue = [50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'X[3] <= 0.8\nentropy = 0.5\nsamples = 100\nvalue = [50, 50, 0]'),
 Text(0.3076923076923077, 0.5833333333333334, 'X[2] <= 4.95\nentropy = 0.168\nsamples = 54\nvalue = [49, 5, 0]'),
 Text(0.15384615384615385, 0.4166666666666667, 'X[2] <= 3.05\nentropy = 0.041\nsamples = 48\nvalue = [47, 1, 0]'),
 Text(0.07692307692307693, 0.25, 'X[1] <= 2.45\nentropy = 0.375\nsamples = 8\nvalue = [7, 1, 0]'),
 Text(0.038461538461538464, 0.08333333333333333, 'entropy = 0.0\nsamples = 7\nvalue = [7, 0, 0]'),
 Text(0.11538461538461539, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.23076923076923078, 0.25, 'X[1] <= 3.1\nentropy = 0.444\nsamples = 40\nvalue = [40, 0, 0]'),
 Text(0.19230769230769232, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.2692307692307692, 0.08333333333333333, 'entropy = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(0.46153846153846156, 0.4166666666666667, 'X[1] <= 3.1\nentropy = 0.444\nsamples = 6\nvalue = [2, 4, 0]'),
 Text(0.38461538461538464, 0.25, 'X[2] <= 5.45\nentropy = 0.444\nsamples = 3\nvalue = [2, 1, 0]'),
 Text(0.34615384615384615, 0.08333333333333333, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0, 0]'),
 Text(0.4230769230769231, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.5384615384615384, 0.25, 'X[2] <= 5.45\nentropy = 0.444\nsamples = 3\nvalue = [0, 3, 0]'),
 Text(0.5, 0.08333333333333333, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.5769230769230769, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.5384615384615384, 0.5833333333333334, 'X[2] <= 4.95\nentropy = 0.043\nsamples = 46\nvalue = [1, 45, 0]'),
 Text(0.5, 0.4166666666666667, 'X[1] <= 3.1\nentropy = 0.444\nsamples = 3\nvalue = [1, 2, 0]'),
 Text(0.46153846153846156, 0.25, 'X[0] <= 6.95\nentropy = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.4230769230769231, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.5, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.5769230769230769, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0, 0]'),
 Text(0.5769230769230769, 0.4166666666666667, 'entropy = 0.0\nsamples = 43\nvalue = [0, 43, 0]'),
 Text(0.7692307692307693, 0.75, 'X[2] <= 4.85\nentropy = 0.168\nsamples = 50\nvalue = [0, 0, 50]'),
 Text(0.6923076923076923, 0.5833333333333334, 'X[1] <= 3.1\nentropy = 0.444\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.6538461538461539, 0.4166666666666667, 'X[0] <= 6.05\nentropy = 0.918\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.6153846153846154, 0.25, 'X[2] <= 4.75\nentropy = 1.0\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(0.5769230769230769, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.6538461538461539, 0.08333333333333333, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.6923076923076923, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.7307692307692307, 0.4166666666666667, 'entropy = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.8461538461538461, 0.5833333333333334, 'X[3] <= 1.75\nentropy = 0.043\nsamples = 44\nvalue = [0, 0, 46]'),
 Text(0.8076923076923077, 0.4166666666666667, 'X[2] <= 5.15\nentropy = 0.5\nsamples = 2\nvalue = [0, 0, 1]'),
 Text(0.7692307692307693, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8846153846153846, 0.4166666666666667, 'entropy = 0.0\nsamples = 42\nvalue = [0, 0, 42]')]
```

10.Evaluate the designed model on the Irish dataset iwith various metrics.

In [12]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

df = pd.read_csv('iris.csv')

X = df.drop(['Species'], axis=1)
y = df['Species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

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

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

22MCA1005 - ML(LAB 04)

In [1]:
```
import pandas as pd
train_dataset = pd.read_csv("D:/2nd sem/machine learning/train.csv")
```

In [2]:
```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.
train_dataset.head()
```

Out[2]:

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

In [3]:
```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_dataset.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
```

In [4]:
```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_dataset.describe()
```

Out[4]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [5]:
```
#3. From the above analysis if any attributes are not relevant in accessing the survival pattern
#of the passenger then drop those columns.

train_dataset = train_dataset.drop(['PassengerId','Name','Ticket','Cabin'],axis=1)
train_dataset.head()
```

Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [6]:
```
#4.Check for the missing values in the modified dataset and fill the missing
# values with appropriate methods.

train_dataset.isnull().sum()
median = train_dataset['Age'].median()
train_dataset['Age'].fillna(median,inplace=True)
mode_embarked = train_dataset['Embarked'].mode()[0]
train_dataset['Embarked'].fillna(mode_embarked, inplace=True)
train_dataset.isnull().sum()
```

Out[6]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [7]:
```
#5. Split the modified dataset into 80-20 ratio for training and testing.

from sklearn.model_selection import train_test_split

a = train_dataset.drop('Survived', axis=1)
b = train_dataset['Survived']
features = ["Pclass", "Sex", "SibSp", "Parch"]
a = pd.get_dummies(train_dataset[features])

print(a)
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.2, random_state=42)
```

```
     Pclass  SibSp  Parch  Sex_female  Sex_male
0         3      1      0           0         1
1         1      1      0           1         0
2         3      0      0           1         0
3         1      1      0           1         0
4         3      0      0           0         1
..      ...    ...    ...         ...       ...
886       2      0      0           0         1
887       1      0      0           1         0
888       3      1      2           1         0
889       1      0      0           0         1
890       3      0      0           0         1

[891 rows x 5 columns]
```

In [8]:
```
#6. Apply Logistic Regression and design a model on the training data.

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

model = LogisticRegression()

model.fit(a_train, b_train)
b_pred = model.predict(a_test)
accuracy = accuracy_score(b_test, b_pred)
print("Accuracy of the logistic regression model: {:.2f}%".format(accuracy * 100))
```

```
Accuracy of the logistic regression model: 79.33%
```

In [9]:
```
#7 Fit the created model on the test data.

b_pred = model.predict(a_test)

accuracy = accuracy_score(b_test, b_pred)

print("Accuracy of the logistic regression model: {:.2f}%".format(accuracy * 100))
```

```
Accuracy of the logistic regression model: 79.33%
```

In [ ]:
```

```

22MCA1005
Polynomial Regression

1.Load the salary dataset working environment.

In [1]:
```
import pandas as pd
df = pd.read_csv("position_salaries.csv")
```

In [2]:
```
df
```

Out[2]:

	Position	Years of Experience	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

2.Perform exploratory analysis on the loaded dataset and draw your inferences

In [3]:
```
print(df.head())
```

```
            Position  Years of Experience  Salary
0   Business Analyst                    1   45000
1  Junior Consultant                    2   50000
2  Senior Consultant                    3   60000
3            Manager                    4   80000
4    Country Manager                    5  110000
```

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Position            10 non-null     object
 1   Years of Experience  10 non-null     int64 
 2   Salary              10 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 368.0+ bytes
None
```

In [5]:
```
print(df.describe())
```

```
       Years of Experience         Salary
count             10.00000      10.000000
mean               5.50000  249500.000000
std                3.02765  299373.883668
min                1.00000   45000.000000
25%                3.25000   65000.000000
50%                5.50000  130000.000000
75%                7.75000  275000.000000
max               10.00000 1000000.000000
```

In [6]:
```
df.dropna(inplace=True)
```

In [7]:
```
df
```

Out[7]:

	Position	Years of Experience	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

In [8]:
```
print(df['Salary'].mean())
```

```
249500.0
```

In [9]:
```
print(df['Salary'].median())
```

```
130000.0
```

In [10]:
```
print(df['Salary'].mode())
```

```
0      45000
1      50000
2      60000
3      80000
4     110000
5     150000
6     200000
7     300000
8     500000
9    1000000
Name: Salary, dtype: int64
```

In [11]:
```
print(df['Salary'].std())
```

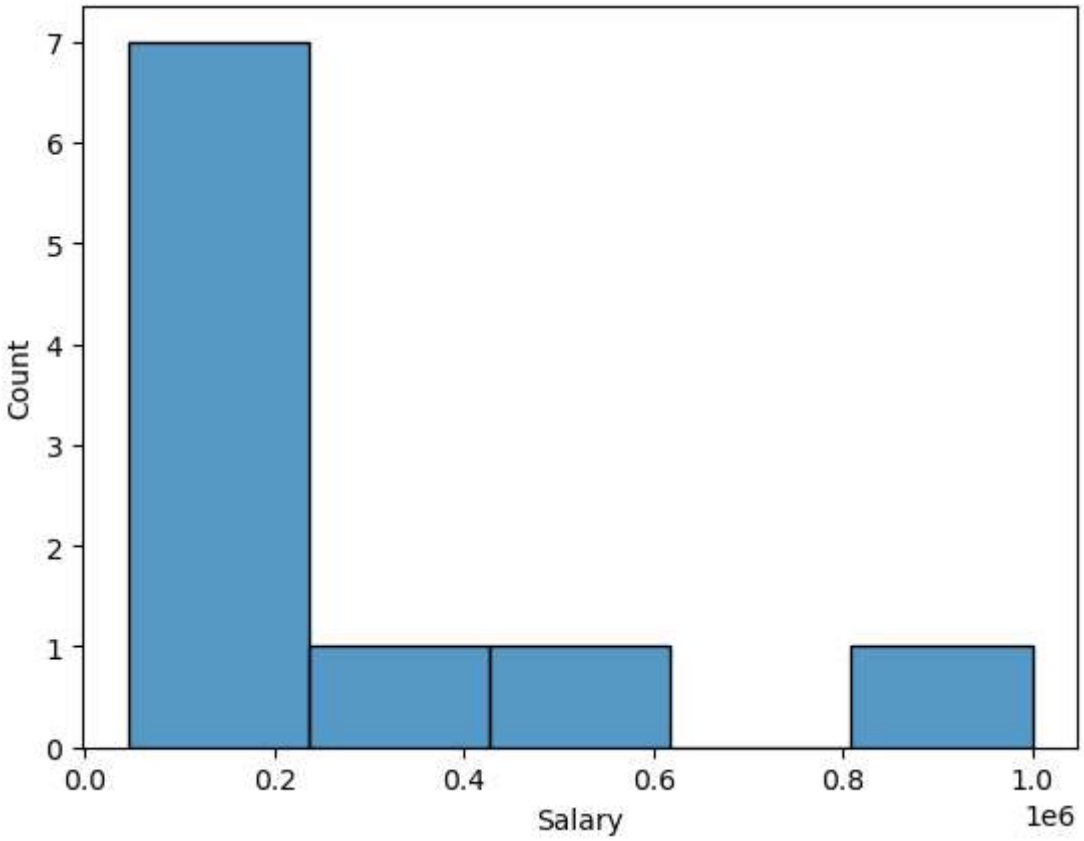
```
299373.88366760087
```

In [12]:
```
print(df['Salary'].min())
print(df['Salary'].max())
```

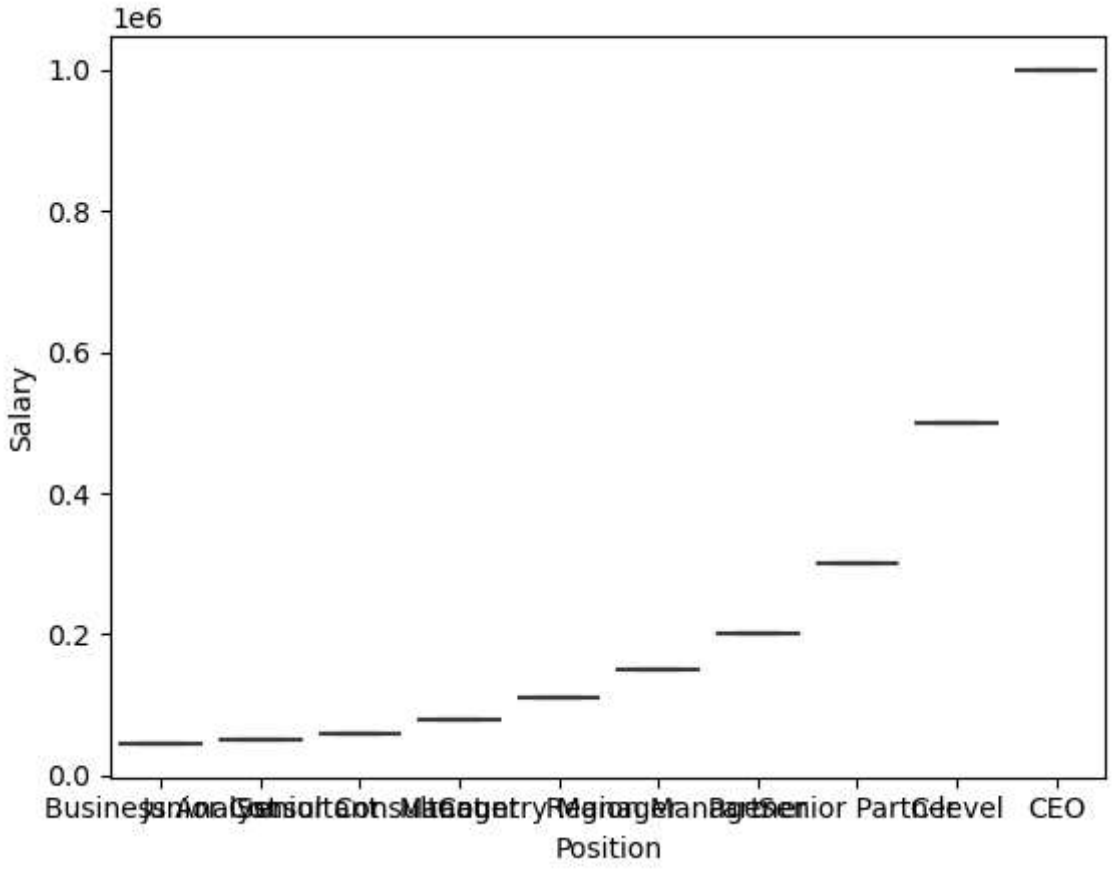
```
45000
1000000
```

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

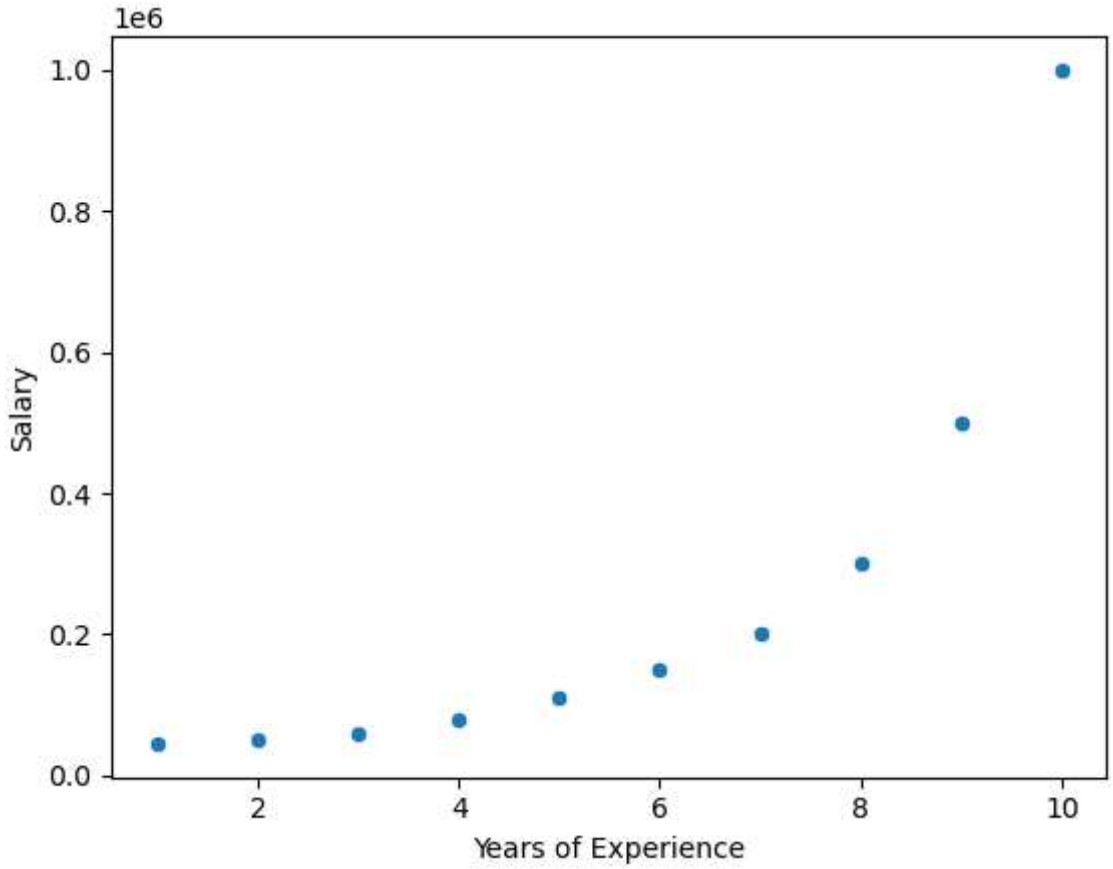
In [15]:
```
sns.histplot(df['Salary'], kde=False)
plt.show()
```



In [17]:
```
sns.boxplot(x='Position', y='Salary', data=df)
plt.show()
```



In [19]:
```
sns.scatterplot(x='Years of Experience', y='Salary', data=df)
plt.show()
```



In [21]:
```
grouped_df = df.groupby('Position').mean()
print(grouped_df)
```

```
                   Years of Experience     Salary
Position
Business Analyst                   1.0    45000.0
C-level                            9.0   500000.0
CEO                               10.0  1000000.0
Country Manager                    5.0   110000.0
Junior Consultant                  2.0    50000.0
Manager                            4.0    80000.0
Partner                            7.0   200000.0
Region Manager                     6.0   150000.0
Senior Consultant                  3.0    60000.0
Senior Partner                     8.0   300000.0
```

In [30]:
```
corr_matrix = df.corr()
```

In [31]:
```
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

3.Apply LinearRegression and design a model on the training data.

In [34]:

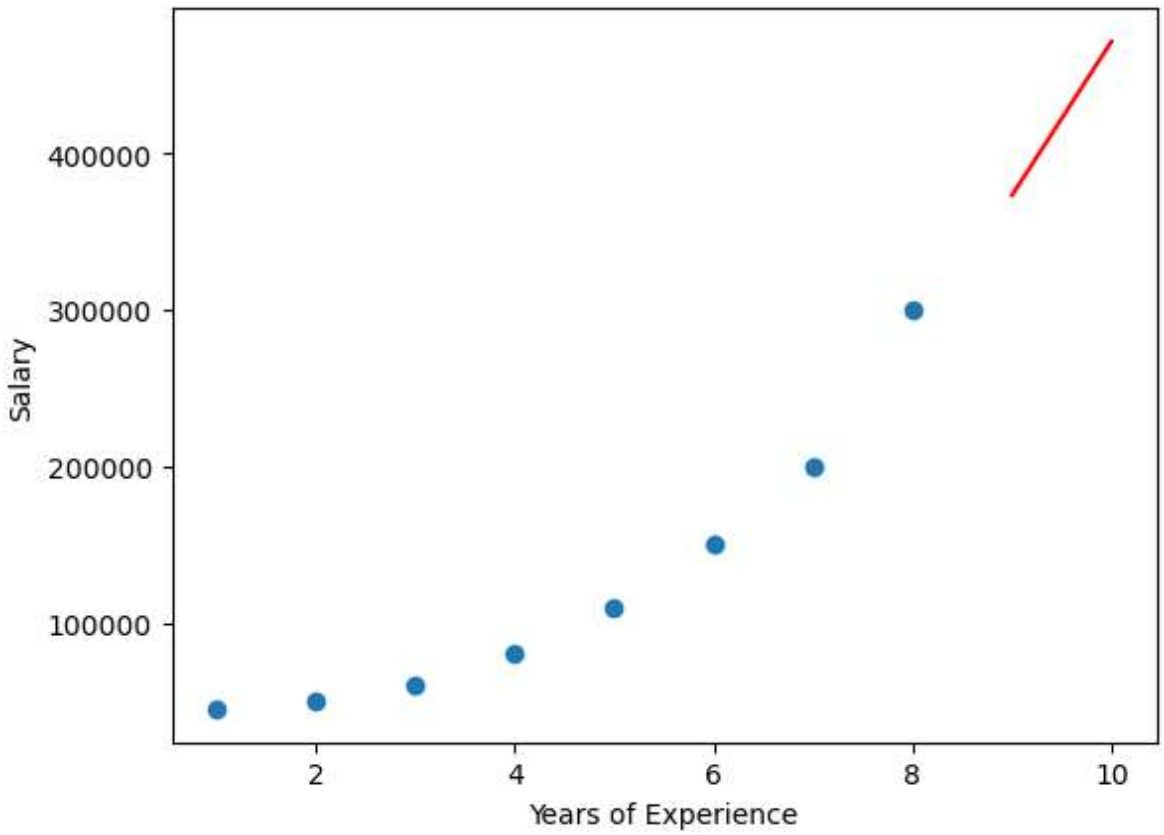
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score


df = pd.read_csv("position_salaries.csv")


df.dropna(inplace=True)

X = df[['Years of Experience']]
y = df['Salary']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc


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("Mean Squared Error:", mse)
print("R-Squared Score:", r2)


print("Coefficient:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Mean Squared Error: 7840057409.334131
R-Squared Score: 0.8451346684575974
Coefficient: [87887.93103448]
Intercept: -240258.62068965525
```

4.Apply PolynomialRegression by manual method and design a model on the training data.

In [39]:

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


df = pd.read_csv('position_salaries.csv')


df.dropna(inplace=True)

X = df[['Years of Experience']]
y = df['Salary']

train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

degree = 2


poly_features = np.polyfit(X_train['Years of Experience'], y_train, degree)

def polynomial_regression(degree, X_train, y_train, X_test):

    poly_features = np.polyfit(X_train['Years of Experience'], y_train, degree
    polynomial = np.poly1d(poly_features)


    y_pred = polynomial(X_test['Years of Experience'])

    return y_pred, poly_features


y_pred, poly_features = polynomial_regression(degree, X_train, y_train, X_test

plt.scatter(X_train, y_train)
plt.plot(X_test, y_pred, color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()


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

print("Mean Squared Error:", mse)
print("R-Squared Score:", r2)


print("Polynomial Features:", poly_features)
```




```
Mean Squared Error: 147726779513.88864
R-Squared Score: -1.363628472222218
Polynomial Features: [  6458.33333333 -24375.         69375.        ]
```

5. Fit the created model on the test data by manual method

In [37]:

```
df = pd.read_csv('position_salaries.csv')


df.dropna(inplace=True)

X = df[['Years of Experience']]
y = df['Salary']


train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]


degree = 2

poly_features = np.polyfit(X_train['Years of Experience'], y_train, degree)


def polynomial_function(x, poly_features):
    y_pred = 0
    for i in range(len(poly_features)):
        y_pred += poly_features[i] * x**i
    return y_pred


y_pred = []
for i in range(len(X_test)):
    pred = polynomial_function(X_test.iloc[i], poly_features)
    y_pred.append(pred)


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

print("Mean Squared Error:", mse)
print("R-Squared Score:", r2)
```

```
Mean Squared Error: 28282854210069.688
R-Squared Score: -451.525667361115
```

6.Apply PolynomialRegression using the python library and design a model on the training data.

In [45]:

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


df = pd.read_csv('position_salaries.csv')

df.dropna(inplace=True)


X = df[['Years of Experience']]
y = df['Salary']

train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]


degree = 2


poly = PolynomialFeatures(degree=degree)
X_poly_train = poly.fit_transform(X_train)


poly_reg = LinearRegression()
poly_reg.fit(X_poly_train, y_train)


X_poly_test = poly.transform(X_test)
y_pred = poly_reg.predict(X_poly_test)


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

print("Mean Squared Error:", mse)
print("R-Squared Score:", r2)
```

```
Mean Squared Error: 147726779513.88876
R-Squared Score: -1.3636284722222203
```

7.Fit the created model on the test data using the python library

In [46]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression


df = pd.read_csv('position_salaries.csv')


df.dropna(inplace=True)


X = df[['Years of Experience']]
y = df['Salary']


train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]


degree = 2


poly = PolynomialFeatures(degree=degree)
X_poly_train = poly.fit_transform(X_train)


poly_reg = LinearRegression()
poly_reg.fit(X_poly_train, y_train)


X_poly_test = poly.transform(X_test)
y_pred = poly_reg.predict(X_poly_test)

print(y_pred)
```

```
[373125.         471458.33333333]
```

In [ ]:
```

```

```
Vipul Shriram Tapare
22MCA1005
SVM(Lab Assignment 06)
```

In [1]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

```
Que.1. Load the breast cancer dataset
```

In [10]:
```
data = load_breast_cancer()
```

```
Que.2.Split the dataset into training and testing by 90:10 ratios
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, te
```

```
Que.3. Perform data preprocessing by standardizing the data
```

In [12]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
Que.4.Apply linear SVM on the training dataset
```

In [13]:
```
linear_svm = LinearSVC()
linear_svm.fit(X_train, y_train)
```

Out[13]:
```
LinearSVC()
```

```
Que.5.Use the above model and fit the test dataset
```

In [14]:
```
linear_svm_pred = linear_svm.predict(X_test)
```

```
Que.6.Display the accuracy and confusion matrix of evaluated model on test
data
```

In [15]:
```
linear_svm_acc = accuracy_score(y_test, linear_svm_pred)
linear_svm_cm = confusion_matrix(y_test, linear_svm_pred)
print("Linear SVM accuracy:", linear_svm_acc)
print("Linear SVM confusion matrix:\n", linear_svm_cm)
```

```
Linear SVM accuracy: 0.9824561403508771
Linear SVM confusion matrix:
 [[23  1]
 [ 0 33]]
```

Que.7.Apply SVM on the training dataset using polynomial kernel

In [16]:
```
poly_svm = SVC(kernel='poly', degree=3)
poly_svm.fit(X_train, y_train)
```

Out[16]:
```
SVC(kernel='poly')
```

Que.8.Use the above model and fit the test dataset

In [17]:
```
poly_svm_pred = poly_svm.predict(X_test)
```

Que.9.Display the accuracy and confusion matrix of evaluated model on test data

In [18]:
```
poly_svm_acc = accuracy_score(y_test, poly_svm_pred)
poly_svm_cm = confusion_matrix(y_test, poly_svm_pred)
print("Polynomial SVM accuracy:", poly_svm_acc)
print("Polynomial SVM confusion matrix:\n", poly_svm_cm)
```

```
Polynomial SVM accuracy: 0.9298245614035088
Polynomial SVM confusion matrix:
 [[20  4]
 [ 0 33]]
```

Que.10.Apply SVM on the training dataset using RBF kernel

In [19]:
```
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
```

Out[19]:
```
SVC()
```

Que.11.Use the above model and fit the test dataset

In [20]:
```
rbf_svm_pred = rbf_svm.predict(X_test)
```

Que.12.Display the accuracy and confusion matrix of evaluated model on test data

In [21]:
```
rbf_svm_acc = accuracy_score(y_test, rbf_svm_pred)
rbf_svm_cm = confusion_matrix(y_test, rbf_svm_pred)
print("RBF SVM accuracy:", rbf_svm_acc)
print("RBF SVM confusion matrix:\n", rbf_svm_cm)
```

```
RBF SVM accuracy: 0.9649122807017544
RBF SVM confusion matrix:
 [[22  2]
 [ 0 33]]
```

In [ ]:

In [1]:
```
#Name: VIPUL TAPARE
#Reg: 22MCA1005
```

In [2]:
```
#1.Load the dataset (Iris.csv).
import pandas as pd
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

data = pd.read_csv("Iris.csv")
print(data)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

In [3]:
```
#2.Split dataset into test and train (20:80)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.drop('Species', axis=1), data['Species'], test_size=0.2, random_state=0)
```

In [4]:
```
#3.Build KNN classifier with k value as 2 for identifying the flower Species
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred_2 = knn.predict(X_test)
```

In [5]:
```
#4.Build KNN classifier with k value as 4 for identifying the flower Species.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
y_pred_4 = knn.predict(X_test)
print(knn.score(X_test,y_test)*100)
```

```
100.0
```

In [6]:
```
#5. Evaluate the step-3 and step-4.
print("Predicted values for Step-3")
print(y_pred_2)
print("\nPredicted values for Step-4")
print(y_pred_4)
```

```
Predicted values for Step-3
['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-setosa']

Predicted values for Step-4
['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-setosa']
```

In [7]:
```
#6.Design a method for calculating the distance between data points for the given dataset
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
d = euclidean_distance(X_train['SepalLengthCm'],X_train['SepalWidthCm'])
print(d)
```

```
32.75561020649745
```

In [8]:
```
#7. Design a method for finding the nearest neighbours of a given data point using the above method.
import numpy as np

def find_nearest_neighbor(X, y, x_query):
    distances = []
    for i, x in enumerate(X):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y[i]))
    distances.sort()
    return distances[0][1]
```

In [9]:
```
#8.Design a method predicting the data point using the above two methods.
import numpy as np

def predict_knn(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    counts = np.bincount(neighbors)
    return np.argmax(counts)
```

In [10]:
```
#9.Choose any dataset from Kaggle or UCI repository suitable for regression and apply KNN algorithm for regression.

import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/Concrete_Data.xls"
df = pd.read_excel(url)
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

def predict_knn_regression(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    return np.mean(neighbors)

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_pred = []
for i, x in enumerate(X_test):
    y_pred.append(predict_knn_regression(X_train, y_train, x, k=5))
```

In [11]:
```
#10.Evaluate the designed regression model with appropriate metric
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
print(f'R-squared: {r2}')
```

```
Mean Absolute Error: 6.459309144164602
Mean Squared Error: 67.95464317359179
Root Mean Squared Error: 8.243460630923872
R-squared: 0.7362839326848325
```

In [ ]:

In [ ]:

In [ ]:
```
# vipul tapare
# 22MCA1005
```

In [3]:
```
# Que1  Load the dataset (titanic).
import pandas as pd
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

data = pd.read_csv("titanic.csv")
data
```

Out[3]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	em
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	
...	...	...	...	...	...	...	...	...	...	...	
1305	3.0	0.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	NaN	
1306	3.0	0.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	NaN	
1307	3.0	0.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	NaN	
1308	3.0	0.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	NaN	
1309	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

1310 rows × 14 columns

In [8]:
```
#PreProcessing
data = data[['pclass','sex', 'age' , 'sibsp','parch','fare','survived']]
data['sex'] = data['sex'].map({'male':0 ,'female':1})
data = data.dropna()
data.head()
```

Out[8]:

	pclass	sex	age	sibsp	parch	fare	survived
0	1.0	1.0	29.0000	0.0	0.0	211.3375	1.0
1	1.0	0.0	0.9167	1.0	2.0	151.5500	1.0
2	1.0	1.0	2.0000	1.0	2.0	151.5500	0.0
3	1.0	0.0	30.0000	1.0	2.0	151.5500	0.0
4	1.0	1.0	25.0000	1.0	2.0	151.5500	0.0

In [9]:
```
#2.Split dataset into test and train (20:80)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.drop('survived', axis
```

In [10]:
```
#3.Build KNN classifier with k value as 2 for identifying the flower Species
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred_2 = knn.predict(X_test)
```

In [11]:
```
#4.Build KNN classifier with k value as 4 for identifying the flower Species.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
y_pred_4 = knn.predict(X_test)
print(knn.score(X_test,y_test)*100)
```

```
69.85645933014354
```

In [12]:
```
#5. Evaluate the step-3 and step-4.
print("Predicted values for Step-3")
print(y_pred_2)
print("\nPredicted values for Step-4")
print(y_pred_4)
```

```
Predicted values for Step-3
[1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 1.
 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1.
 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0.
 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0.]

Predicted values for Step-4
[1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.
 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.]
```

In [14]:
```
#6.Design a method for calculating the distance between data points for the gi
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
d = euclidean_distance(X_train['sex'],X_train['age'])
print(d)
```

```
957.5926267700947
```

In [18]:
```
#7. Design a method for finding the nearest neighbours of a given data point u
import numpy as np

def find_nearest_neighbor(X, y, x_query):
    distances = []
    for i, x in enumerate(X):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y[i]))
    distances.sort()
    return distances[0][1]
```

In [19]:
```
#8.Design a method predicting the data point using the above two methods.
import numpy as np

def predict_knn(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    counts = np.bincount(neighbors)
    return np.argmax(counts)
```

In [ ]:
```

```

Ex - 09 22MCA1005
Vipul Tapare

RANDOM FOREST CLASSIFIER

1. Load the dataset (iris.csv).

In [2]:
```
import pandas as pd
iris_df = pd.read_csv("Iris.csv")
```

In [3]:
```
iris_df.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

2. Load the dataset (Churnprediction.csv).

In [4]:
```
churn_df = pd.read_csv("telecom_customer_churn.csv")
```

In [5]:
```
churn_df.head()
```

Out[5]:

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Nur Refe
0	0002-ORFBO	Female	37	Yes	0	Frazier Park	93225	34.827662	-118.999073	
1	0003-MKNFE	Male	46	No	0	Glendale	91206	34.162515	-118.203869	
2	0004-TLHLJ	Male	50	No	0	Costa Mesa	92627	33.645672	-117.922613	
3	0011-IGKFF	Male	78	Yes	0	Martinez	94553	38.014457	-122.115432	
4	0013-EXCHZ	Female	75	Yes	0	Camarillo	93010	34.227846	-119.079903	

5 rows × 38 columns

3. Drop columns that are not required for classification of Churn Risk.

In [6]:
```
iris_df = iris_df.drop(['Id'],axis=1)
churn_df = churn_df.drop(['Customer ID'], axis=1)
```

4. If require perform data preprocessing.

In [27]:
```
# There is no need for data preprocessing in this case.
```

5. Split dataset into test and train (20:80).

In [7]:
```
pip install numpy==1.21.3
```

```
Collecting numpy==1.21.3
  Downloading numpy-1.21.3-cp39-cp39-win_amd64.whl (14.0 MB)
     ---------------------------------------- 14.0/14.0 MB 6.1 MB/s eta 0:00:
00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.21.5
    Uninstalling numpy-1.21.5:
      Successfully uninstalled numpy-1.21.5
Successfully installed numpy-1.21.3
Note: you may need to restart the kernel to use updated packages.

ERROR: pip's dependency resolver does not currently take into account all the
packages that are installed. This behaviour is the source of the following de
pendency conflicts.
daal4py 2021.6.0 requires daal==2021.4.0, which is not installed.
```

In [9]:
```
from sklearn.model_selection import train_test_split

# Separate the features from the target variable
X = iris_df.drop('Species', axis=1)
y = iris_df['Species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [10]:
```
X_churn = churn_df.drop('Churn Category', axis=1)
y_churn = churn_df['Churn Category']

X_train_churn, X_test_churn, y_train_churn, y_test_churn = train_test_split(X_
```

6. Build any three classification models for identifying Churn Risk.

In [11]:
```
# Logistic Regression

from sklearn.linear_model import LogisticRegression

logistic_regression = LogisticRegression(max_iter=1000)
logistic_regression.fit(X_train, y_train)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

# Random Forest

from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
```

Out[11]:
```
RandomForestClassifier()
```

7. Build Voting ensemble classifier on the training dataset.

In [12]:
```
from sklearn.ensemble import VotingClassifier

voting_classifier = VotingClassifier(estimators=[('lr', logistic_regression),
voting_classifier.fit(X_train, y_train)
```

Out[12]:
```
VotingClassifier(estimators=[('lr', LogisticRegression(max_iter=1000)),
                             ('dt', DecisionTreeClassifier()),
                             ('rf', RandomForestClassifier())])
```

8. Build Bagging ensemble classifier on the training dataset.

In [13]:
```
import warnings
# Ignore the FutureWarning for the base_estimator parameter
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [14]:
```
from sklearn.ensemble import BaggingClassifier

bagging_classifier = BaggingClassifier(base_estimator=logistic_regression, n_e
bagging_classifier.fit(X_train, y_train)
```

Out[14]:
```
BaggingClassifier(base_estimator=LogisticRegression(max_iter=1000),
                  n_estimators=100)
```

9. Build Boosting ensemble classifier on the training dataset.

In [15]:
```
from sklearn.ensemble import AdaBoostClassifier

adaboost_classifier = AdaBoostClassifier(base_estimator=logistic_regression, n
adaboost_classifier.fit(X_train, y_train)
```

Out[15]:
```
AdaBoostClassifier(base_estimator=LogisticRegression(max_iter=1000),
                   n_estimators=100)
```

10. Fit the models designed from step-5 to step-8 on the test dataset.

In [16]:
```
logistic_regression.score(X_test, y_test)
```

Out[16]:
```
0.9333333333333333
```

In [17]:
```
decision_tree.score(X_test, y_test)
```

Out[17]:
```
0.9
```

In [18]:
```
random_forest.score(X_test, y_test)
```

Out[18]:
```
0.9333333333333333
```

In [19]:
```
voting_classifier.score(X_test, y_test)
```

Out[19]:
```
0.9333333333333333
```

In [20]:
```
bagging_classifier.score(X_test, y_test)
```

Out[20]:
```
0.9333333333333333
```

In [21]:
```
adaboost_classifier.score(X_test, y_test)
```

Out[21]:
```
0.9333333333333333
```

11. Evaluate the designed models from step-5 to step-8 with appropriate classification metrics.

In [22]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

accuracy_score(y_test, logistic_regression.predict(X_test))
```

Out[22]:
```
0.9333333333333333
```

In [ ]:
```

```

In [ ]:
```
#Name: Vipul Tapare
#Reg: 22MCA1005
```

In [3]:
```
#1.Load the dataset (iris.csv).
import pandas as pd
ds = pd.read_csv("diabetes.csv")
print(ds)
print("22MCA1005")
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6   
1              1       85             66             29        0  26.6   
2              8      183             64              0        0  23.3   
3              1       89             66             23       94  28.1   
4              0      137             40             35      168  43.1   
..           ...      ...            ...            ...      ...   ...   
763           10      101             76             48      180  32.9   
764            2      122             70             27        0  36.8   
765            5      121             72             23      112  26.2   
766            1      126             60              0        0  30.1   
767            1       93             70             31        0  30.4   

     DiabetesPedigreeFunction  Age  Outcome  
0                       0.627   50        1  
1                       0.351   31        0  
2                       0.672   32        1  
3                       0.167   21        0  
4                       2.288   33        1  
..                        ...  ...      ...  
763                     0.171   63        0  
764                     0.340   27        0  
765                     0.245   30        0  
766                     0.349   47        1  
767                     0.315   23        0  

[768 rows x 9 columns]
22MCA1005
```

In [5]:
```
#2.check if there are missing values are present in the dataset.
ds.isnull()
print("22MCA1005")
```

```
22MCA1005
```

In [15]:
```
#3. Perform data preprocessing.
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
ds=pd.read_csv("diabetes.csv",names=names)
print(ds)
print("22MCA1005")
```

```
            preg     plas           pres           skin     test  mass  \
0    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
1              6      148             72             35        0  33.6
2              1       85             66             29        0  26.6
3              8      183             64              0        0  23.3
4              1       89             66             23       94  28.1
..           ...      ...            ...            ...      ...   ...
764           10      101             76             48      180  32.9
765            2      122             70             27        0  36.8
766            5      121             72             23      112  26.2
767            1      126             60              0        0  30.1
768            1       93             70             31        0  30.4

                         pedi  age    class
0    DiabetesPedigreeFunction  Age  Outcome
1                       0.627   50        1
2                       0.351   31        0
3                       0.672   32        1
4                       0.167   21        0
..                        ...  ...      ...
764                     0.171   63        0
765                      0.34   27        0
766                     0.245   30        0
767                     0.349   47        1
768                     0.315   23        0

[769 rows x 9 columns]
22MCA1005
```

In [6]:
```
#4.Split dataset into test and train (20:80).
from sklearn.model_selection import train_test_split

X=ds.iloc[:,:-1]
y=ds.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
print("22MCA1005")
```

```
22MCA1005
```

In [7]:
```
#5.Build any three classification models for identifying diabetes.

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, rar
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
print(y_pred)
print("22MCA1005")
```

```
[1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0]
22MCA1005

C:\Users\vipul\AppData\Local\Temp\ipykernel_23580\2036093327.py:12: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  classifier.fit(x_train, y_train)
```

In [13]:
```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
ds=pd.read_csv("diabetes.csv")
X=ds.iloc[:,:-1]
y=ds.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

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

```
[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0
 1 0 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0
 1 0 1 0 0 0]

C:\Users\vipul\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: D
ataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [16]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
ds=pd.read_csv("diabetes.csv")
X=ds.iloc[:,:-1]
y=ds.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print(y_pred)
print("22MCA1005")
```

```
[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0
 1 0 1 0 0 0]
22MCA1005

C:\Users\vipul\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: D
ataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [18]:
```
#6.Build Voting ensemble classifier on the training dataset.
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
ds=pd.read_csv("diabetes.csv")
X=ds.iloc[:,:-1]
y=ds.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
classifier1 = DecisionTreeClassifier()
classifier2 = LogisticRegression(solver='lbfgs',max_iter=100)
classifier3 = SVC()

voting_classifier = VotingClassifier(
    estimators=[('dt', classifier1), ('lr', classifier2), ('svc', classifier3)
    voting='hard'
)

voting_classifier.fit(X_train, y_train)

y_pred = voting_classifier.predict(X_test)
print(y_pred)
print("22MCA1005")
```

```
[0 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0
 1 0 1 0 0 0]
22MCA1005
```

```
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:9
8: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:13
3: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
814: ConvergenceWarning: lbfgs failed to converge (status=1):
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 (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

In [20]:
```
#7.Build Bagging ensemble classifier on the training dataset.
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("diabetes.csv")
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
base_classifier = DecisionTreeClassifier()

bagging_classifier = BaggingClassifier(
    base_estimator=base_classifier,
    n_estimators=10,
    random_state=42
)

bagging_classifier.fit(X_train, y_train)

y_pred = bagging_classifier.predict(X_test)
print(y_pred)
print("22MCA1005")
```

```
[0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1
 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 1 0 0 0 0 0]
22MCA1005

C:\Users\vipul\anaconda3\lib\site-packages\sklearn\ensemble\_bagging.py:719:
DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

In [21]:
```
#8.Build Boosting ensemble classifier on the training dataset.
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
ds=pd.read_csv("diabetes.csv")
X=ds.iloc[:,:-1]
y=ds.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
boosting_classifier = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1
)

boosting_classifier.fit(X_train, y_train)

y_pred = boosting_classifier.predict(X_test)

print(y_pred)
print("22MCA1005")
```

```
[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0
 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 1 0 1 0 0 0]
22MCA1005

C:\Users\vipul\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:494: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [23]:
```
#9.Fit the models designed from step-5 to step-8 on the test dataset.
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, rar
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_test, y_test)
y_pred= classifier.predict(x_train)
print("Random Forest")
print(y_pred)

#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_test, y_test)
y_pred = model.predict(x_train)
print("Naive Bayes")
print(y_pred)


#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(x_test, y_test)
y_pred = classifier.predict(x_train)
```

```
print("Logistic Regression")
print(y_pred)
```

```
Random Forest
[1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 1
 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0
 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1
 1 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0
 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1
 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0
 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Naive Bayes
[1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0
 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0
 1 0 0 1 1 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1
 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1
 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0]
Logistic Regression
[1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0
 0 1 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1
 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1
 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1
 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1
 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0]
```

```
C:\Users\student\AppData\Local\Temp/ipykernel_1976/3890763543.py:10: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples,), for example using ravel().
  classifier.fit(x_test, y_test)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Da
taConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Da
taConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

In [24]:
```
#10.Evaluate the designed models from step-5 to step-8 with appropria
import pandas as pd
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("diabetes.csv")
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar

from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Random Forest Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Random Forest Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Random Forest F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Random Forest Classification")
print(report)
print("22MCA1005")


#Naive Bayes
from sklearn.naive_bayes import GaussianNB

x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

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

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Naive Bayes Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Naive Bayes Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Naive Bayes Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Naive Bayes F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Naive Bayes Classification")
print(report)

#Logistic Regression
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Logistic Regression Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Logistic Regression Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Logistic Regression F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Logistic Regression Classification")
print(report)
print("22MCA1005")
```

```
Random Forest Accuracy:  0.7662337662337663
Random Forest Precision:  0.6585365853658537
Random Forest Recall:  0.5510204081632653
Random Forest F1 score:  0.6
Random Forest Classification
              precision    recall  f1-score   support

           0       0.81      0.87      0.83       105
           1       0.66      0.55      0.60        49

    accuracy                           0.77       154
   macro avg       0.73      0.71      0.72       154
weighted avg       0.76      0.77      0.76       154

22MCA1005
Naive Bayes Accuracy:  0.8051948051948052
Naive Bayes Precision:  0.7021276595744681
Naive Bayes Recall:  0.673469387755102
Naive Bayes F1 score:  0.6875000000000001
Naive Bayes Classification
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       105
           1       0.70      0.67      0.69        49

    accuracy                           0.81       154
   macro avg       0.78      0.77      0.77       154
weighted avg       0.80      0.81      0.80       154

Logistic Regression Accuracy:  0.8051948051948052
Logistic Regression Precision:  0.7209302325581395
Logistic Regression Recall:  0.6326530612244898
Logistic Regression F1 score:  0.6739130434782609
Logistic Regression Classification
              precision    recall  f1-score   support

           0       0.84      0.89      0.86       105
           1       0.72      0.63      0.67        49

    accuracy                           0.81       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.80      0.81      0.80       154

22MCA1005
```

```
C:\Users\vipul\AppData\Local\Temp\ipykernel_23580\407173190.py:20: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  classifier.fit(x_train, y_train)
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: D
ataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: D
ataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:

In [1]:
```
#Name: Vipul Tapare
#Reg: 22MCA1005
```

In [2]:
```
#1.Load the dataset (Mall Customers.csv)

import pandas as pd
data = pd.read_csv("Mall_Customers.csv")
print(data)
```

```
     CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

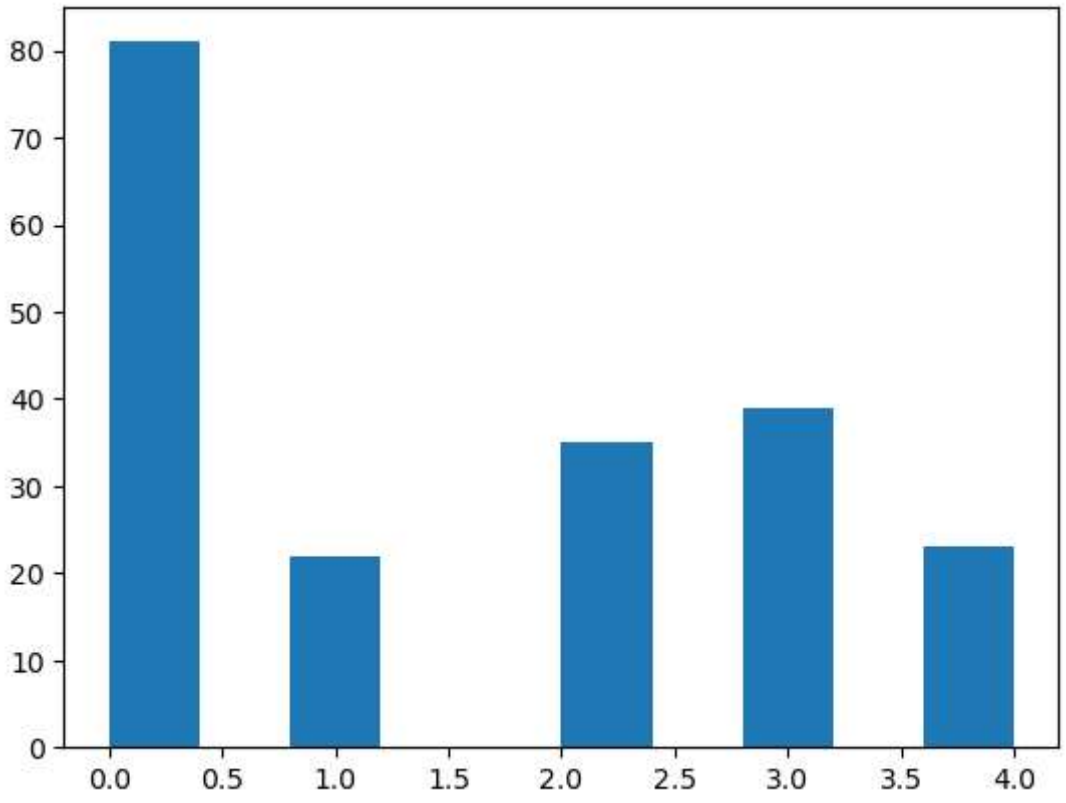
In [3]:
```
#2. If require perform data preprocessing.
data.isnull().sum()
```

Out[3]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

In [4]:
```
#3. Perform k-means clustering using sklearn with arbitrary number of
from sklearn.cluster import KMeans
import numpy as np

X= data.iloc[:, [3,4]].values

n_clusters = 5

kmeans = KMeans(n_clusters=n_clusters)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

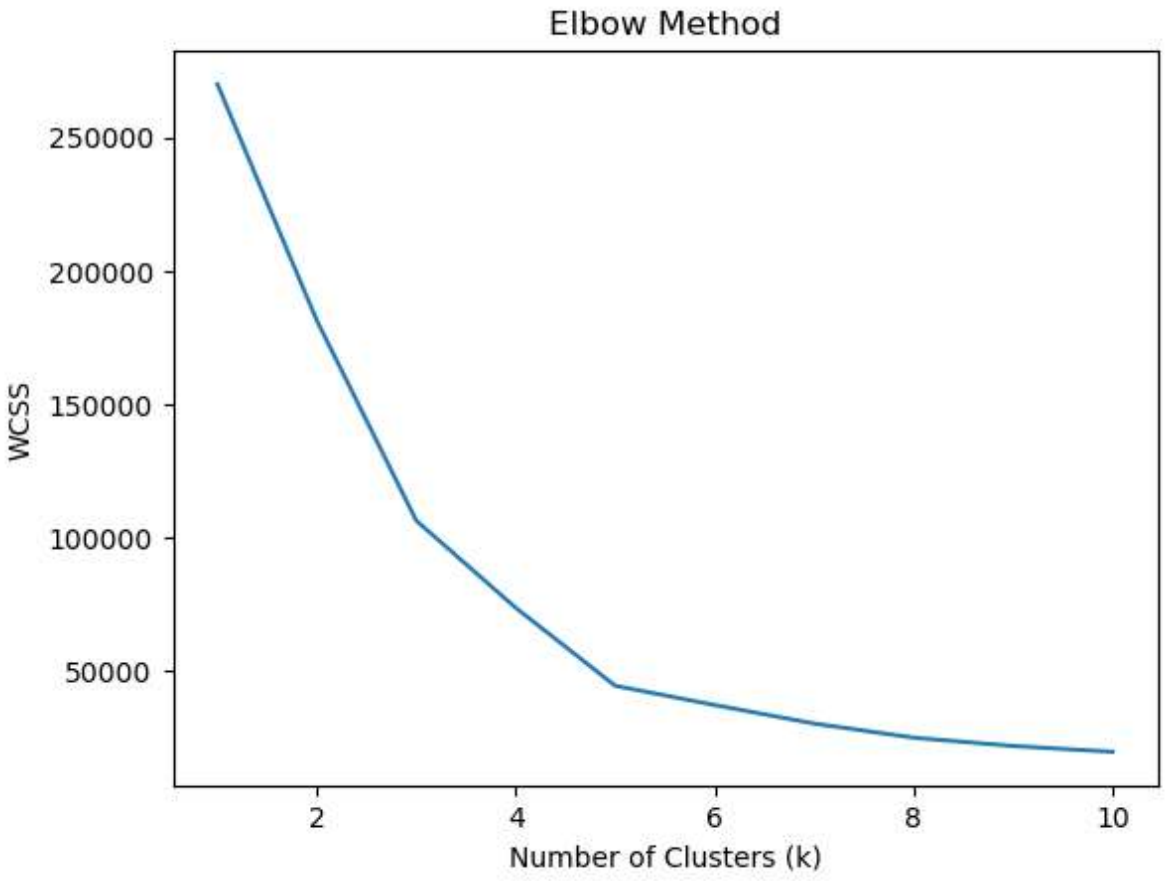
```
Cluster Labels:
[4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
 1 4 1 4 1 4 0 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 3 2 3 0 3 2 3 2 3 0 3 2 3 2 3 2 3 2 3 0 3 2 3 2 3
 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3]
Cluster Centers:
[[55.2962963  49.51851852]
 [25.72727273 79.36363636]
 [88.2        17.11428571]
 [86.53846154 82.12820513]
 [26.30434783 20.91304348]]
```

In [5]:
```
#4.Draw the inferences you find out from the clustering process.

import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



In [6]:
```
#5. Apply elbow method and find the optimal number clusters for the given data
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X= data.iloc[:, [3,4]].values

wcss = []

max_clusters = 10

for k in range(1, max_clusters+1):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, max_clusters+1), wcss)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.title('Elbow Method')
plt.show()
```

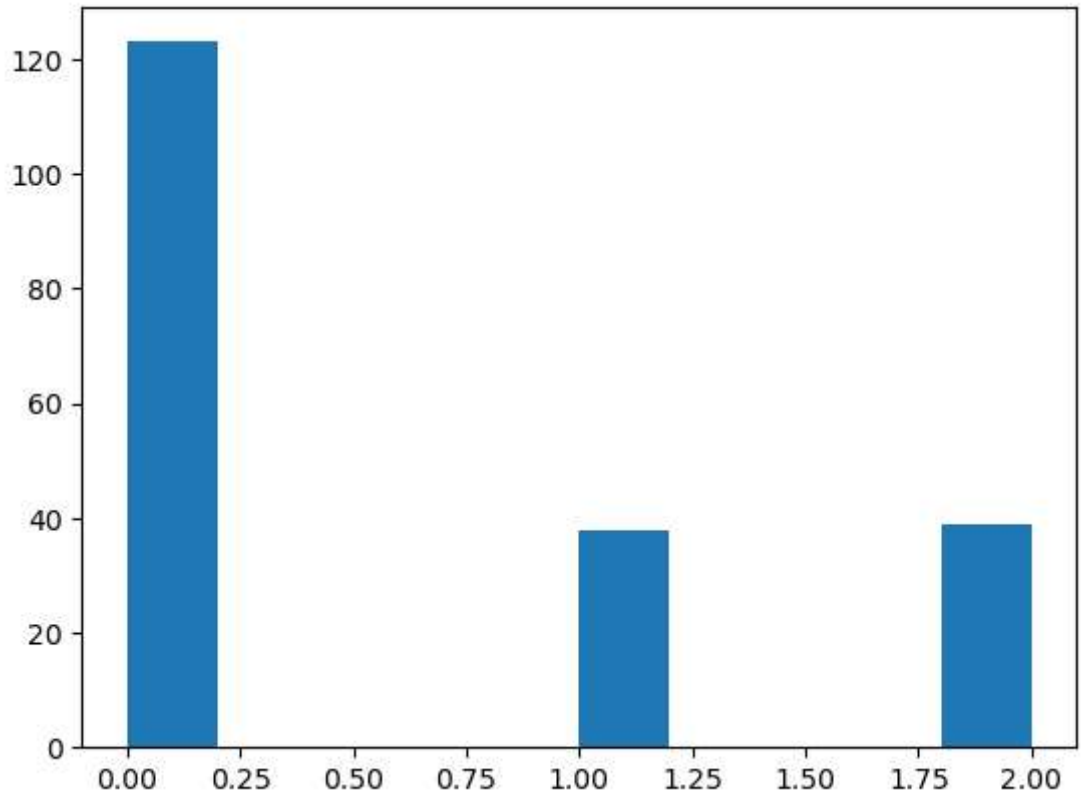
```
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: U
serWarning: KMeans is known to have a memory leak on Windows with MKL, when t
here are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [7]:
```
#6.  Perform  K-means  clusteringusing  sklearn  with  optimal  number  of clu

from sklearn.cluster import KMeans
import numpy as np

X= data.iloc[:, [3,4]].values

optimal_clusters = 3

kmeans = KMeans(n_clusters=optimal_clusters)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

```
Cluster Labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2]
Cluster Centers:
[[44.15447154 49.82926829]
 [87.         18.63157895]
 [86.53846154 82.12820513]]
```

In [8]:
```
#7. Draw the inferences you find out from the clustering process.
import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



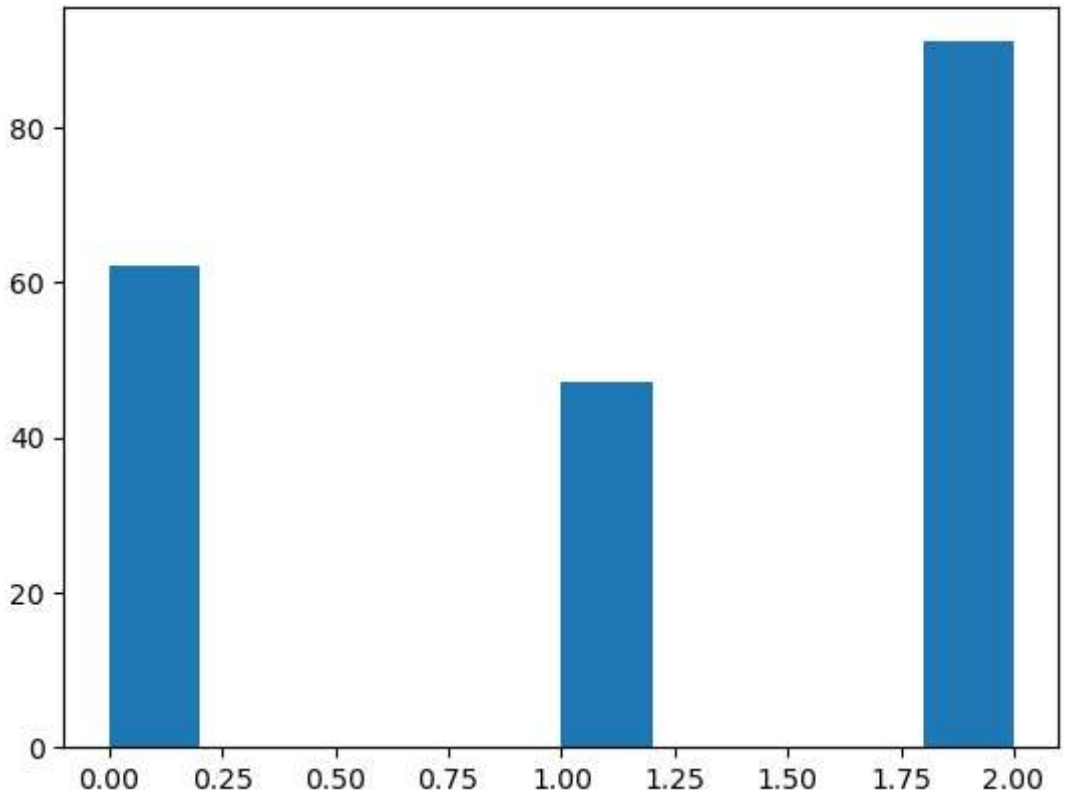
In [9]:
```
#8. Which attributes are strongly correlated with Spending Score?
import pandas as pd

correlation_matrix = data.corr()
spending_score_corr = correlation_matrix['Spending Score (1-100)']

spending_score_corr = spending_score_corr.sort_values(ascending=False)

print(spending_score_corr)
```

```
Spending Score (1-100)    1.000000
CustomerID                0.013835
Annual Income (k$)        0.009903
Age                      -0.327227
Name: Spending Score (1-100), dtype: float64
```

In [10]:
```
#9. Apply K-means clustering using sklearn with optimal number of clusters alo
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

optimal_clusters = 3

correlation_matrix = data.corr()
highly_correlated_features = correlation_matrix['Spending Score (1-100)'].abs(

X = data[highly_correlated_features].values

kmeans = KMeans(n_clusters=optimal_clusters)
kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

```
Cluster Labels:
[2 0 1 0 2 0 1 0 1 0 1 0 1 0 1 0 2 0 1 0 2 0 1 0 1 0 2 2 1 0 1 0 1 0 1 0 1
 0 1 0 2 0 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 0 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 0 1 0 2 0 1 0 1 0 2 0 1 0 1 0 1 0 2 0 1 0 2 0
 1 0 1 0 1 0 1 0 1 0 1 0 2 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 2 0 1 0 2
 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
Cluster Centers:
[[80.74193548 29.56451613]
 [14.59574468 42.95744681]
 [47.78021978 43.05494505]]
```

In [11]:
```
#10. Draw the inferences you find out from the clustering process
import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



In [ ]: