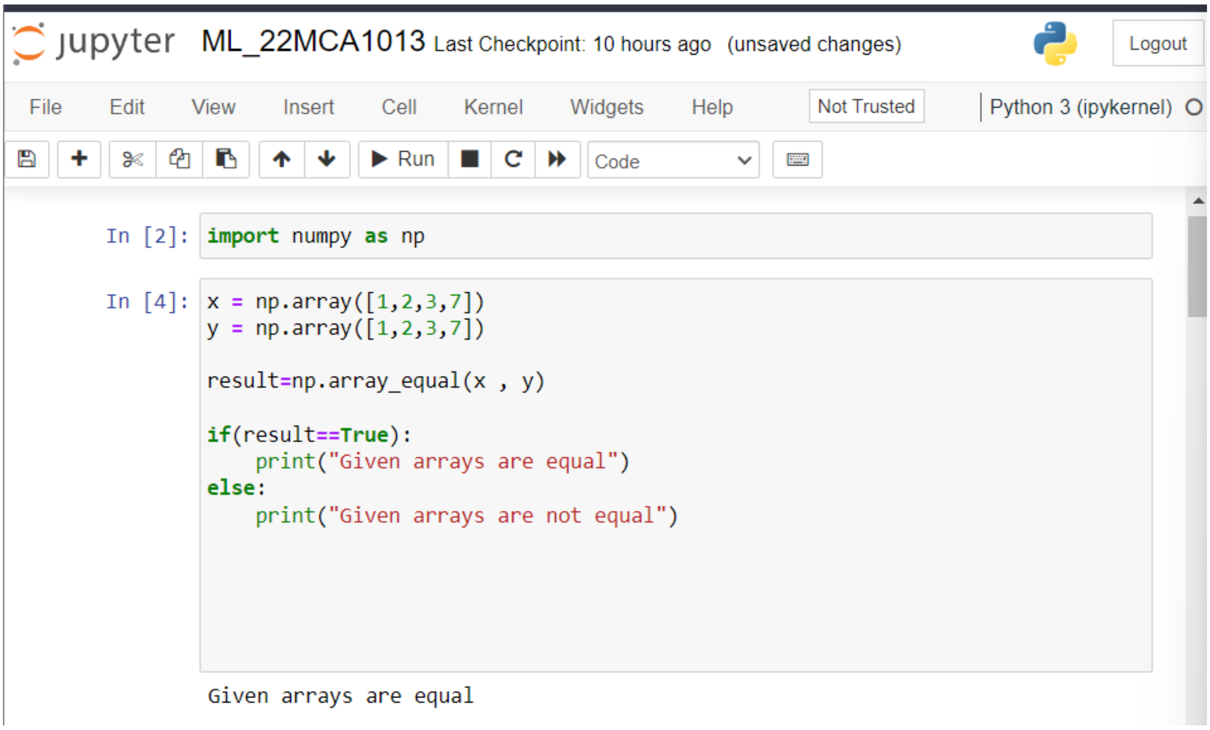
Name -Tushar Mathur

Register Number – 22MCA1013

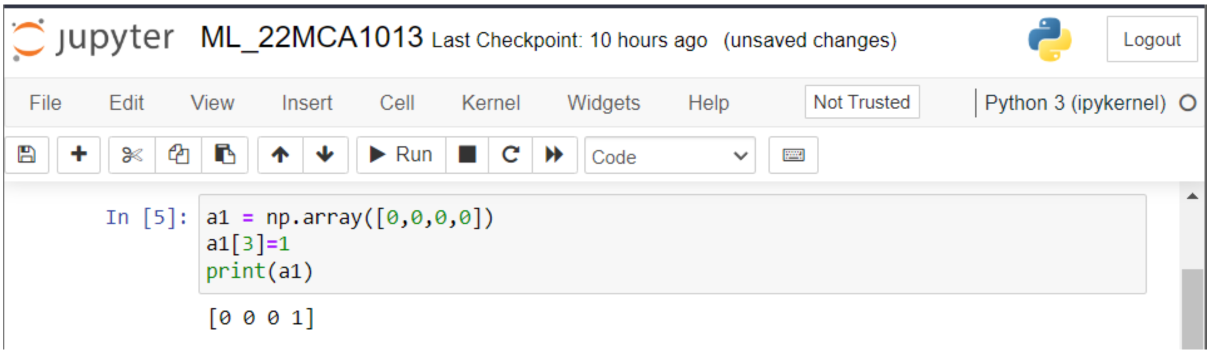
Faculty Name - Dr. Rajesh M.

Machine Learning Lab 01

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'.

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [7]:

```
a = np.array([1,2,3,44,5,60,7,8,19,50])
max_Number =  max(a)
print(max_Number)
index=(np.where(a==max(a)))
a[index]=100
print(a)


min_Number = min(a1)
print(min_Number)
index=np.where(a==min(a))
a[index]=0
print(a)
```

```
60
[  1   2   3  44   5 100   7   8  19  50]
0
[  0   2   3  44   5 100   7   8  19  50]
```

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

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (unsaved changes)

```
In [8]: arr = np.array([8, 7, 1, 5, 3, 4])
        print("Array is : ", arr)
        x = 2
        print("Value to which nearest element is to be found: ", x)
        difference_array = np.absolute(arr-x)
        index = difference_array.argmin()
        print("Nearest element to the given values is : ", arr[index])
        print("Index of nearest value is : ", index)
```

```
Array is :  [8 7 1 5 3 4]
Value to which nearest element is to be found:  2
Nearest element to the given values is :  1
Index of nearest value is :  2
```

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

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (autosaved)

In [23]:
```python
print("Original matrix:\n")
X = np.random.rand(5, 10)
print(X)
print("\nSubtract the mean of each row of the said matrix:\n")
Y = X - X.mean(axis=1, keepdims=True)
print(Y)
```

```
Original matrix:

[[0.18804895 0.65973741 0.13988195 0.80937378 0.42354425 0.46845903
  0.65106497 0.68157655 0.80629061 0.99498471]
 [0.75683223 0.55756144 0.90926344 0.59077622 0.29081068 0.07118534
  0.37757091 0.90106488 0.13422601 0.45434318]
 [0.94500321 0.81719058 0.26080042 0.48729985 0.11189086 0.28143446
  0.20300451 0.73405615 0.20507219 0.94642977]
 [0.40442118 0.74760043 0.08698872 0.55393232 0.18288105 0.46446183
  0.94526831 0.21973394 0.51344628 0.63677337]
 [0.57002531 0.66902327 0.25535658 0.55506446 0.18554471 0.35041713
  0.61546898 0.23428314 0.54149356 0.08024989]]

Subtract the mean of each row of the said matrix:

[[-0.39424727  0.07744119 -0.44241427  0.22707756 -0.15875198 -0.11383719
   0.06876875  0.09928033  0.22399439  0.41268849]
 [ 0.25246879  0.053198    0.40490001  0.08641279 -0.21355275 -0.43317809
  -0.12679252  0.39670145 -0.37013742 -0.05002025]
 [ 0.44578501  0.31797238 -0.23841778 -0.01191835 -0.38732734 -0.21778374
  -0.29621369  0.23483795 -0.29414602  0.44721157]
 [-0.07112957  0.27204969 -0.38856203  0.07838158 -0.29266969 -0.01108891
   0.46971756 -0.2558168   0.03789553  0.16122263]
 [ 0.16433261  0.26333056 -0.15033612  0.14937175 -0.22014799 -0.05527557
   0.20977627 -0.17140956  0.13580086 -0.32544281]]
```

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

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [26]:

```python
import pandas as pd
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Mic
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no',
df = pd.DataFrame(exam_data)
print("Original DataFrame")
print(df)
print("\nAfter converting index in a column:")
df.reset_index(level=0, inplace=True)
print(df)
print("\nHiding index:")
print( df.to_string(index=False))
```

```
Original DataFrame
        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
```

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```
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?

```
In [13]: ps1 = pd.Series([5, 6, 7, 8, 9, 12, 3, 44])
         ps2 = pd.Series([1, 3, 6, 4, 10, 99, 50, 23])

         print("Series1:")
         print(ps1)
         print("\nSeries2:")
         print(ps2)


         print("\nItems of ps1 not present in ps2:")
         res = ps1[~ps1.isin(ps2)]
         print(res)
```

```
Series1:
0     5
1     6
2     7
3     8
4     9
5    12
6     3
7    44
dtype: int64

Series2:
0     1
1     3
2     6
3     4
4    10
5    99
6    50
7    23
dtype: int64
```

```
Items of ps1 not present in ps2:
0     5
2     7
3     8
4     9
5    12
7    44
dtype: int64
```

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

jupyter ML_22MCA1013 Last Checkpoint: 10 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [28]:
```python
num_state = np.random.RandomState(100)
num_series = pd.Series(num_state.normal(10, 4, 20))
print("Original Series:")
print(num_series)
result = np.percentile(num_series, q=[0, 25, 50, 75, 100])
print("\nMinimum, 25th percentile, median, 75th, and maximum of a given series
print(result)
```

```
Original Series:
0      3.000938
1     11.370722
2     14.612143
3      8.990256
4     13.925283
5     12.056875
6     10.884719
7      5.719827
8      9.242017
9     11.020006
10     8.167892
11    11.740654
12     7.665620
13    13.267388
14    12.690883
15     9.582355
16     7.874878
17    14.118931
18     8.247458
19     5.526727
dtype: float64

Minimum, 25th percentile, median, 75th, and maximum of a given series:
[ 3.00093811  8.09463867 10.23353705 12.21537733 14.61214321]
```

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

jupyter ML_22MCA1013 Last Checkpoint: 11 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [29]:

```
np.random.RandomState(100)
num_series = pd.Series(np.random.randint(1, 5, [15]))
print("Original Series:")
print(num_series)
print("Top 2 Freq:", num_series.value_counts())
result = num_series[~num_series.isin(num_series.value_counts().index[:1])] = '
print(num_series)
```

```
Original Series:
0     1
1     1
2     2
3     4
4     2
5     2
6     2
7     2
8     2
9     4
10    3
11    4
12    1
13    2
14    1
dtype: int32
Top 2 Freq: 2    7
1    4
4    3
3    1
dtype: int64
0     Other
1     Other
2         2
3     Other
```

jupyter ML_22MCA1013 Last Checkpoint: 11 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```
3     4
4     2
5     2
6     2
7     2
8     2
9     4
10    3
11    4
12    1
13    2
14    1
dtype: int32
Top 2 Freq: 2    7
1    4
4    3
3    1
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.

jupyter ML_22MCA1013 Last Checkpoint: 11 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```
dtype: object
```

In [15]:

```
test_str = "my name is tushar"
print("The original string is : " + str(test_str))
res = test_str[0].upper() + test_str[1:]
print("The string after uppercasing initial character : " + str(res))
```

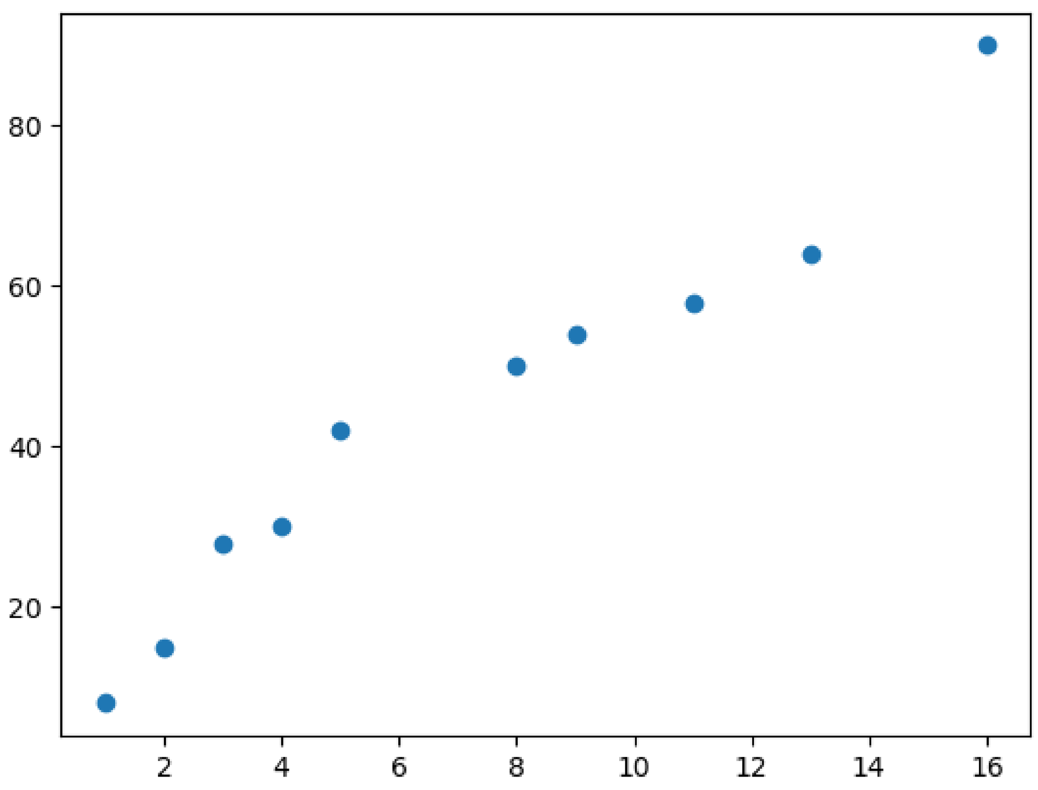
```
The original string is : my name is tushar
The string after uppercasing initial character : My name is tushar
```

Machine Learning 02

TUSHAR MATHUR

22MCA1013

In [38]:
```python
#1 Consider the following sample dataset and write a program to project it in the form of a scatter plot and  observe any
# relationship that exists in between the experience and pay.

import matplotlib.pyplot as plt
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])
plt.scatter(x, y)
plt.show()
```



In [14]:
```python
#2 Fill(Find)the following table by using the respective values of xand y
import matplotlib.pyplot as plt
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)

xsub = x - xbar
ysub = y - ybar
mxsys = xsub*ysub
xsqr = (xsub*xsub)


data = pd.DataFrame({'x': x, 'y': y, 'xsub': xsub, 'ysub': ysub, 'mxsys': mxsys, 'xsqr': xsqr})
print(data)
print(np.sum(mxsys))
print(np.sum(xsqrsum))
```

```
    x   y  xsub  ysub   mxsys   xsqr
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
1087.2
227.60000000000002
```

In [15]:
```python
#3 Use the above table and solve the following equations (using OLS method) for finding the values of w0 and w1.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

xysum = np.sum(mxsys)
xsqrsum = np.sum(xsqr)

w1 = xysum/xsqrsum
print(w1)

w0 = ybar - (w1*xbar)
print(w0)
```

```
4.776801405975395
9.50702987697715
```

In [33]:
```python
#4 Write a python program to find the above two values (w0 and w1) for any given dataset.
import matplotlib.pyplot as plt
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)

xsub = x - xbar
ysub = y - ybar
mxsys = xsub*ysub
xsqr = (xsub*xsub)

xysum = np.sum(mxsys)
xsqrsum = np.sum(xsqr)

w1 = xysum/xsqrsum
print(w1)

w0 = ybar - (w1*xbar)
print(w0)
```

```
4.776801405975395
9.50702987697715
```

In [17]:
```python
#5 Write a python program to find the above two values (w0 and w1) for any given dataset using Normal Equations.
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
cof = np.linalg.inv(X.T @ X) @ X.T @ y
w0 = cof[0]
w1 = cof[1]
print("w0 =", w0)
print("w1 =", w1)
```

```
w0 = 9.507029876977714
w1 = 4.776801405975396
```

In [26]:
```python
#6 Write a python program to find the above two values (w0 and w1) for any given dataset using gradient descent.
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
noi = 1000
w0 = 0.0
w1 = 0.0
m = float(len(x))
for i in range(noi):
    yp = w0 + w1 * x
    errors = yp - y
    gw0 = (1/m) * np.sum(errors)
    gw1 = (1/m) * np.sum(errors * x)
    w0 = w0 - alpha * gw0
    w1 = w1 - alpha * gw1

print("w0 =", w0)
print("w1 =", w1)
```

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

In [31]:
```python
#7 Write a python program to store the above dataset as pandas data-frame and implement linear regression model by splitting
#the dataset into training set and testing set (80:20).
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)
Xt, X_tst, yt, 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) = 7.812500000000007
w1 (slope) = 4.937499999999999
```

In [37]:
```python
#8 Identify and display the appropriate metrics for measuring the accuracy of the linearmodel created above.
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) = 2.3159722222222197
Mean Squared Error (MSE) = 5.5335018539951895
Root Mean Squared Error (RMSE) = 2.352339655320887
R-squared (R^2) score = 0.9446649814600481
```

LAB3-ML_LAB-CAM-ASSIGNMENT-3

TUSHAR MATHUR

22MCA1013

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

In [2]:
```python
# Define the dataset
data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '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  Rejected
5     Young        No  Soft Skill  Selected
6     Young       Yes  Soft Skill  Selected
7       Old        No  Soft Skill  Rejected
8       Old        No  Hard Skill  Rejected
9    Middle        No  Soft Skill  Selected
```

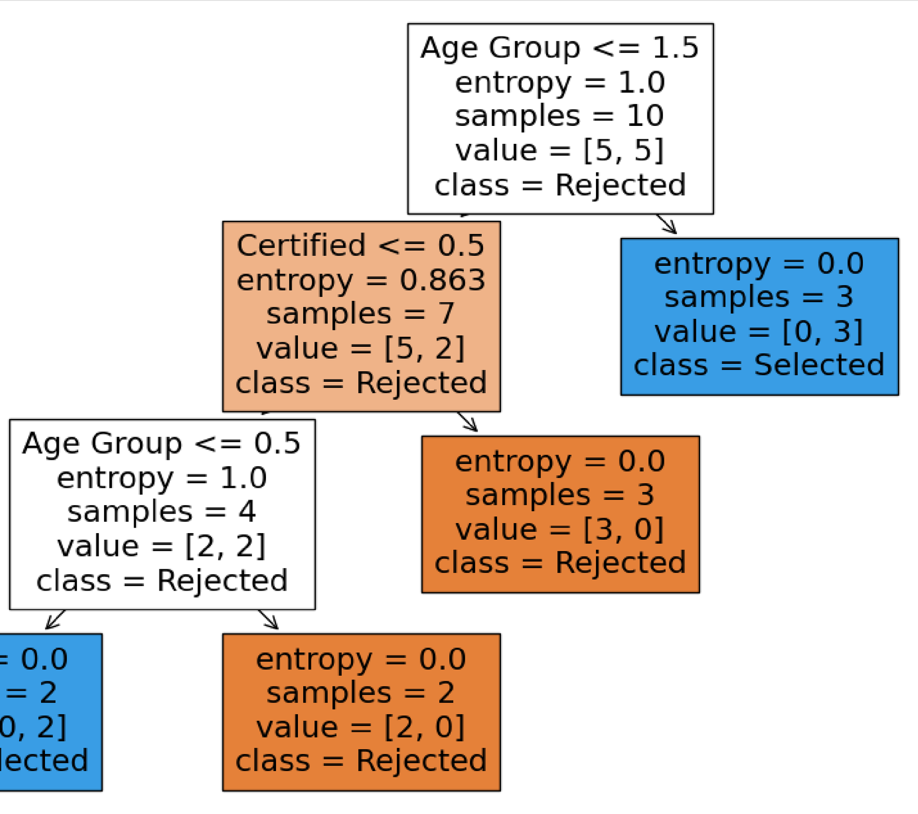
1. Consider the following dataset and calculate the entropy and information gain w.r.t the target attribute named “Status”.

In [3]:
```python
# Calculate the entropy of the target attribute "Status"
status_counts = data['Status'].value_counts()
num_instances = 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}")

# Calculate the information gain of each attribute w.r.t. the "Status" attribute
for attribute in ['Age Group', 'Certified', 'Skill Type']:
    entropy_attribute = 0
    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 of {attribute}: {information_gain:.3f}")
```

```
Entropy of Status: 1.000
Information gain of Age Group: 0.600
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]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load the dataset
data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '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']
})

# Encode categorical variables
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'])

# Build a decision tree classifier using information gain
dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

# Plot the decision tree
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]:
```python
# Transform the decision tree into decision rules
def tree_to_rules(tree, feature_names):
    rules = []
    for feature in feature_names:
        for value in np.unique(data[feature]):
            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

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

```
Age Group == 0 => Status = Rejected
Age Group == 1 => Status = Rejected
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.

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

# Load the dataset
data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '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']
})

# Encode categorical variables
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'])

# Build a decision tree classifier using information gain
dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

# Export the decision tree as text rules
tree_rules = export_text(dt, feature_names=X.columns.tolist())

# Print the decision rules
print(tree_rules)
```

```
|--- Age Group <= 1.50
|   |--- Age Group <= 0.50
|   |   |--- Certified <= 0.50
|   |   |   |--- class: 1
|   |   |--- Certified >  0.50
|   |   |   |--- class: 0
|   |--- Age Group >  0.50
|   |   |--- 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]:
```python
import pandas as pd
import math

# Load the dataset
data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 'Old', 'Middle'],
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '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 f_en(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

# Example usage:
entropy = f_en(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]:
```python
import math

def find_entropy(df):
    '''
    Find entropy of the given dataset.

    Parameters:
    df (pandas.DataFrame): input dataset.

    Returns:
    float: entropy of the dataset.
    '''
    entropy = 0
    num_records = len(df)
    classes = df['Status'].unique()
    for c in classes:
        num_c = len(df[df['Status']==c])
        p = num_c/num_records
        entropy -= p * math.log2(p)
    return entropy

def find_gain(df, attribute):
    '''
    Find information gain of the given attribute w.r.t. the 'Status' attribute.

    Parameters:
    df (pandas.DataFrame): input dataset.
    attribute (str): name of the attribute to calculate information gain for.

    Returns:
    float: information gain of the attribute.
    '''
    total_entropy = find_entropy(df)
    values = df[attribute].unique()
    entropy = 0
    for v in values:
        num_v = len(df[df[attribute]==v])
        df_v = df[df[attribute]==v]
        entropy += (num_v/len(df))*find_entropy(df_v)
    gain = total_entropy - entropy
    return gain
find_entropy(data)
find_gain(data, 'Status')
```

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

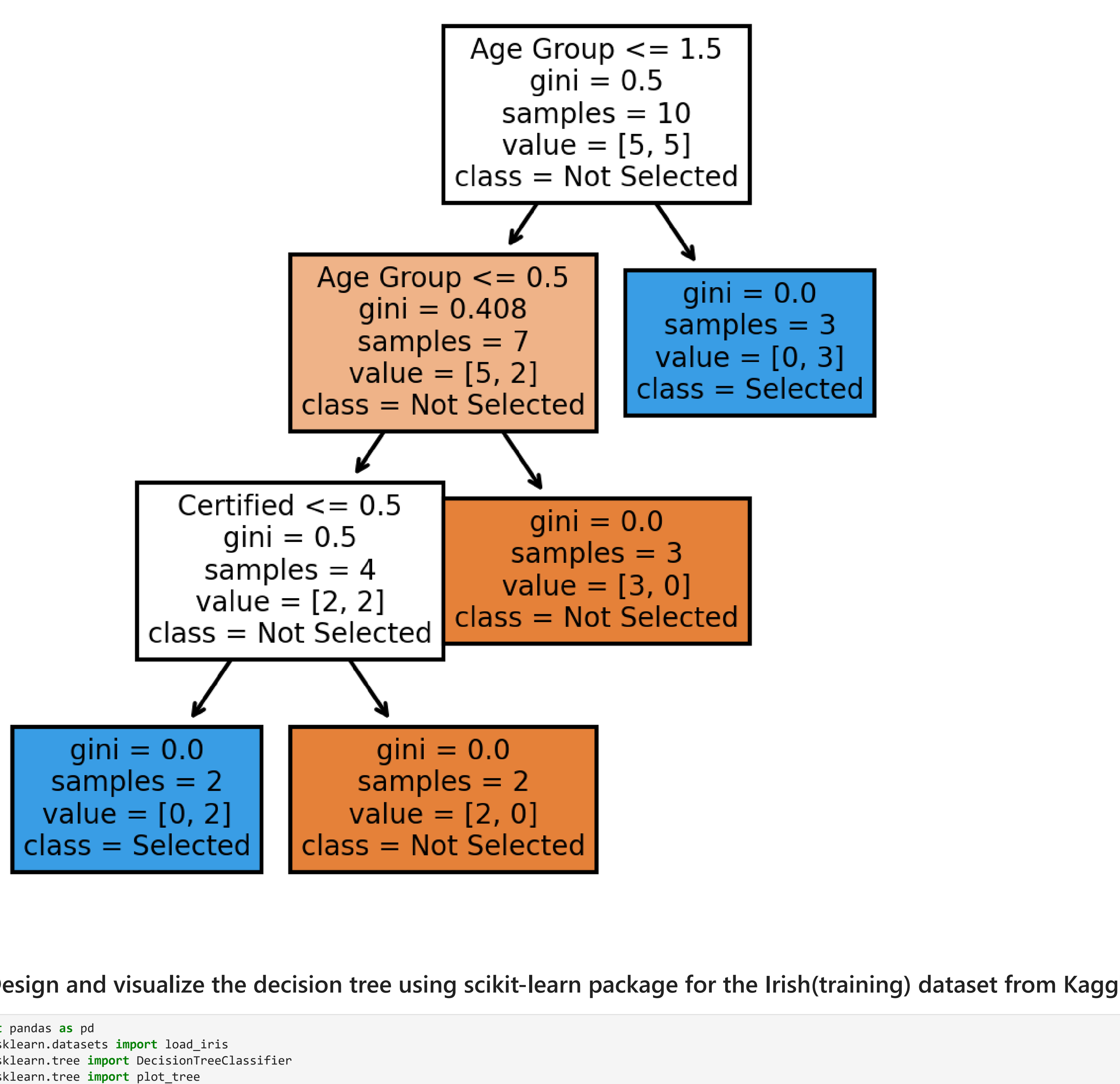
7. Load the above dataset as data frame in python

In [9]:
```python
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', 'No', 'Yes', 'No', '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']
})
```

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

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

# create decision tree model
model = DecisionTreeClassifier()

# fit model with data
model.fit(X, y)

# plot decision tree
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)
```

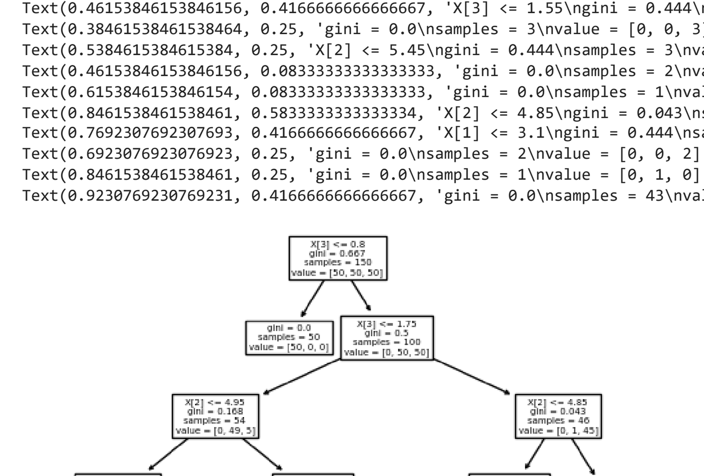
Out[10]:
```
[Text(0.6666666666666666, 0.875, 'Age Group <= 1.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]\nclass = Not Selected'),
 Text(0.5, 0.625, 'Age Group <= 0.5\ngini = 0.408\nsamples = 7\nvalue = [5, 2]\nclass = Not Selected'),
 Text(0.3333333333333333, 0.375, 'Certified <= 0.5\ngini = 0.5\nsamples = 4\nvalue = [2, 2]\nclass = Not Selected'),
 Text(0.16666666666666666, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = Selected'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass = Not Selected'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = Not Selected'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = Selected')]
```



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

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

# Load Iris dataset
iris = load_iris()

# Create a pandas DataFrame from the dataset
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Add target column to the DataFrame
iris_df['target'] = iris.target

# Create feature matrix X and target vector y
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]

# Create a decision tree classifier object
tree = DecisionTreeClassifier()

# Fit the classifier to the data
tree.fit(X, y)

# Visualize the decision tree
plot_tree(tree)
```

Out[11]:
```
[Text(0.5, 0.9166666666666666, 'X[3] <= 0.8\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'X[3] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'X[2] <= 4.95\ngini = 0.168\nsamples = 54\nvalue = [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'X[3] <= 1.65\ngini = 0.041\nsamples = 48\nvalue = [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'X[3] <= 1.55\ngini = 0.444\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'X[2] <= 5.45\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'X[2] <= 4.85\ngini = 0.043\nsamples = 46\nvalue = [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'X[1] <= 3.1\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue = [0, 0, 43]')]
```



10.Evaluate the designed model on the Irish dataset itself with various metrics.

In [12]:
```python
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

# Load the Irish dataset
df = pd.read_csv('iris.csv')

# Split the dataset into features (X) and target (y)
X = df.drop(['Species'], axis=1)
y = df['Species']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the decision tree model
tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

# Predict the test set
y_pred = tree.predict(X_test)

# Evaluate the model using various metrics
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 the metrics
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
```

In [3]:
```
#1. Load the Titanic dataset from Kaggle to working environment.

import pandas as pd
train_data = pd.read_csv("E:/MCA/train.csv")
```

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

Out[4]:

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

In [5]:
```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_data.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 [7]:
```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_data.describe()
```

Out[7]:

	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 [8]:
```
#3. From the above analysis if any attributes are not relevant in accessing the sur
#of the passenger then drop those columns.

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

Out[8]:

	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 [10]:
```
#4.Check for the missing values in the modified dataset and fill the missing
# values with appropriate methods.

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

Out[10]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [24]:
```
#5. Split the modified dataset into 80-20 ratio for training and testing.

from sklearn.model_selection import train_test_split

x = train_data.drop('Survived', axis=1)
y = train_data['Survived']
features = ["Pclass", "Sex", "SibSp", "Parch"]
x = pd.get_dummies(train_data[features])

print(x)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_sta
```

```
     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 [25]:
```
#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(x_train, y_train)
```

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

In [26]:
```
#7 Fit the created model on the test data.

y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

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

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

TUSHAR MATHUR

22MCA1013

MACHINE LEARNING LAB - 5

1.Load the salary dataset working environment.

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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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())
```

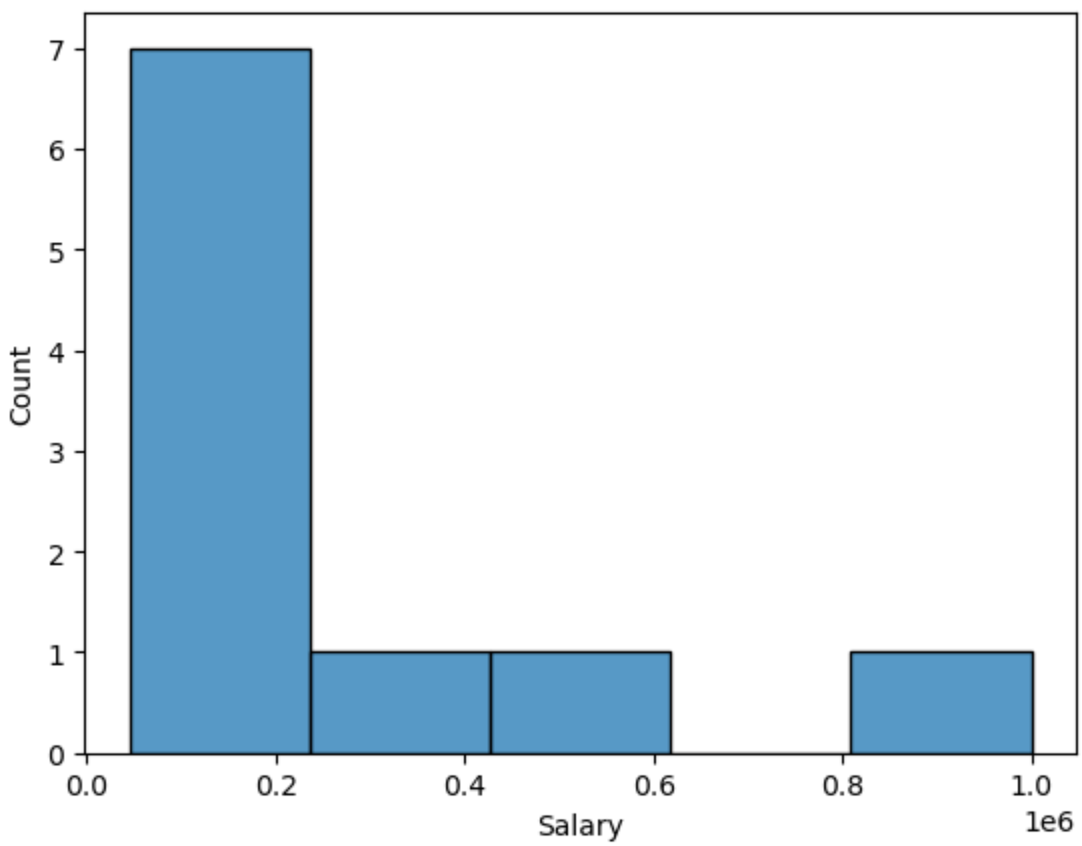
```
299373.88366760087
```

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

```
45000
1000000
```

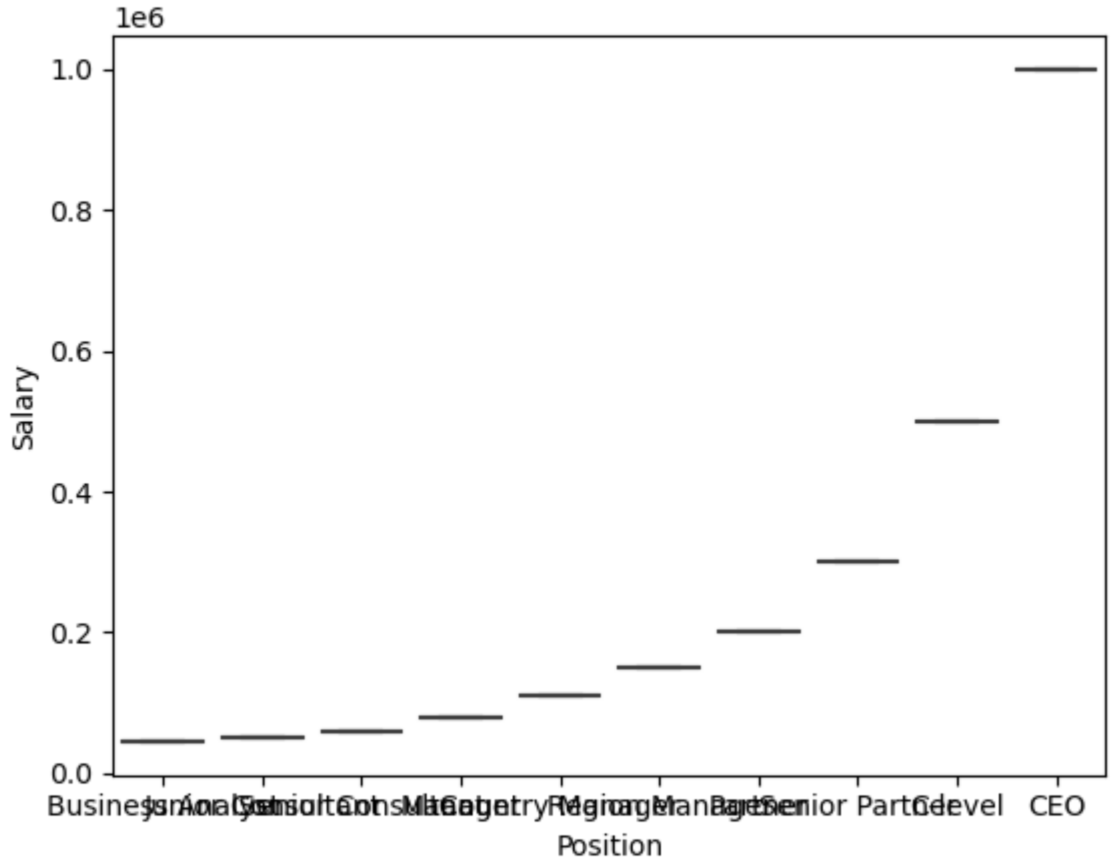
In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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



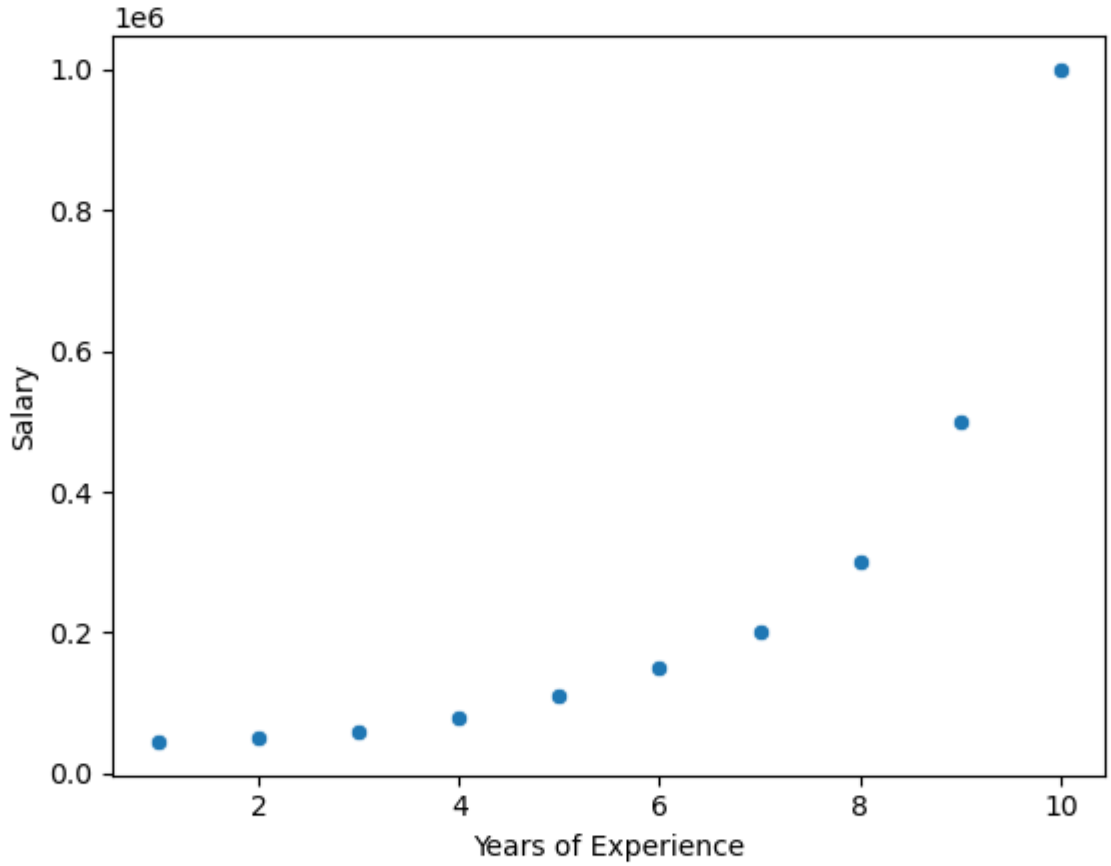
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



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()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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, random_sta


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_)
```

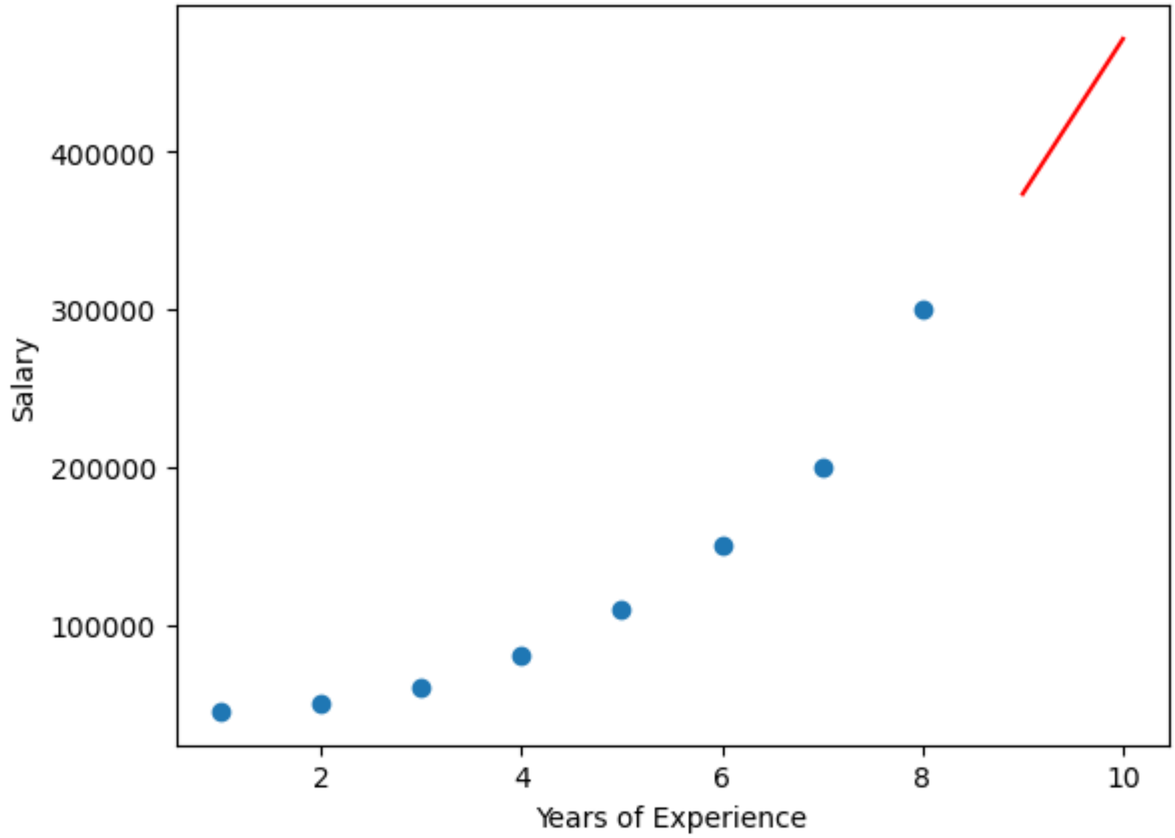
```
Mean Squared Error: 7840057409.334131
R-Squared Score: 0.8451346684575974
Coefficient: [87887.93103448]
Intercept: -240258.62068965525
```

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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js




```
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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

TUSHAR MATHUR

22MCA1013

MACHINE LEARNING LAB (SVM)

In [23]:
```
#Load the breast cancer dataset form sklearn.
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

In [24]:
```
#If require perform data preprocessing.
from sklearn.preprocessing import StandardScaler

x= data.data
y = data.target
scaler = StandardScaler()
X = scaler.fit_transform(x)
```

In [25]:
```
#Split the dataset into training and testing by 90:10 ratios
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_sta
```

In [26]:
```
#Apply linear SVM on the training dataset.
from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X_train, y_train)
```

Out[26]:
```
LinearSVC()
```

In [27]:
```
#Use the above model and fit the test dataset.
y_predicted = svm.predict(X_test)
```

In [28]:
```
#Display  the  accuracy  and  confusion  matrix  of  evaluated  model  on  test dat
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_predicted)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9473684210526315
```

In [53]:
```
#Apply SVM on the training dataset using polynomial kernel.
from sklearn.svm import SVC

svm = SVC(kernel='poly', degree=3)
svm.fit(X_train, y_train)
```

Out[53]:
```
SVC(kernel='poly')
```

In [54]:
```
#Use the above model and fit the test dataset.
y_predicted_poly = svm.predict(X_test)
```

In [55]:
```
#Display  the  accuracy  and  confusion  matrix  of  evaluated  model  on  test dat

accuracy = accuracy_score(y_test, y_predicted_poly)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8771929824561403
```

In [56]:
```
#Apply SVM on the training dataset using RBF kernel.

svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)
```

Out[56]:
```
SVC()
```

In [57]:
```
#Use the above model and fit the test dataset.
y_predicted_rbf = svm.predict(X_test)
```

In [58]:
```
#Display  the  accuracy  and  confusion  matrix  of  evaluated  model  on  test dat

accuracy = accuracy_score(y_test, y_predicted_rbf)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9649122807017544
```

TUSHAR MATHUR

22MCA1013

ML_LAB_KNN

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("irisDataset.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), d
```

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 d
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 [15]:
```
#7. Design a method for finding the nearest neighbours of a given data point using
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 [16]:
```
#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 [17]:
```
#9.Choose any dataset from Kaggle or UCI repository suitable for regression and app
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/compressi
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_sta
y_pred = []
for i, x in enumerate(X_test):
    y_pred.append(predict_knn_regression(X_train, y_train, x, k=5))
```

In [18]:
```
#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
```

TUSHAR MATHUR

22MCA1013

PYTHON LAB - 8

In [5]:
```
# 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[5]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	emba
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 [6]:

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

```
C:\Users\TR\AppData\Local\Temp\ipykernel_10600\2925064758.py:3: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
user_guide/indexing.html#returning-a-view-versus-a-copy
  data['sex'] = data['sex'].map({'male':0 ,'female':1})
```

Out[6]:

	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 [7]:

```
#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=1),
```

In [8]:

```
#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 [9]:

```
#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 [10]:

```
#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 [11]:

```
#6.Design a method for calculating the distance between data points for the given d
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 [14]:

```
#7. Design a method for finding the nearest neighbours of a given data point using
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 [15]:

```
#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)
```

22MCA1013

TUSHAR MATHUR

MACHINE LEARNING LAB

RANDOM FOREST

1. Load the dataset (iris.csv).

In [24]:
```
import pandas as pd
iris_df = pd.read_csv("IrisDataset.csv")
```

In [25]:
```
print("22MCA1013")
iris_df.head()
```

22MCA1013

Out[25]:

	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 [26]:
```
churn_df = pd.read_csv("telecom_customer_churn.csv")
```

In [27]:
```
print("22MCA1013")
churn_df.head()
```

22MCA1013

Out[27]:

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Numl Referr
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 [28]:
```
iris_df = iris_df.drop(['Id'],axis=1)
churn_df = churn_df.drop(['Customer ID'], axis=1)
print("22MCA1013")
```

```
22MCA1013
```

4. If require perform data preprocessing.

In [6]:
```
# There is no need for data preprocessing in this case.
```

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

In [10]:
```
pip install numpy==1.21.3
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=No
ne)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connectio
n.HTTPSConnection object at 0x0000021F9665D700>: Failed to establish a new connecti
on: [Errno 11001] getaddrinfo failed')': /simple/numpy/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=No
ne)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connectio
n.HTTPSConnection object at 0x0000021F9665DA30>: Failed to establish a new connecti
on: [Errno 11001] getaddrinfo failed')': /simple/numpy/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=No
ne)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connectio
n.HTTPSConnection object at 0x0000021F9665DD30>: Failed to establish a new connecti
on: [Errno 11001] getaddrinfo failed')': /simple/numpy/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=No
ne)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connectio
n.HTTPSConnection object at 0x0000021F9665DEE0>: Failed to establish a new connecti
on: [Errno 11001] getaddrinfo failed')': /simple/numpy/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=No
ne)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connectio
n.HTTPSConnection object at 0x0000021F9665DFA0>: Failed to establish a new connecti
on: [Errno 11001] getaddrinfo failed')': /simple/numpy/
ERROR: Could not find a version that satisfies the requirement numpy==1.21.3 (from
versions: none)
ERROR: No matching distribution found for numpy==1.21.3
WARNING: There was an error checking the latest version of pip.
```

In [43]:

```
print("22MCA1013")
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)
```

```
22MCA1013
```

In [44]:

```
print("22MCA1013")
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_churn
```

```
22MCA1013
```

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

In [45]:
```
# Logistic Regression
print("22MCA1013")
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)
```

```
22MCA1013
```

Out[45]:
```
RandomForestClassifier()
```

7. Build Voting ensemble classifier on the training dataset.

In [46]:
```
print("22MCA1013")
from sklearn.ensemble import VotingClassifier

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

```
22MCA1013
```

Out[46]:
```
VotingClassifier(estimators=[('lr', LogisticRegression(max_iter=1000)),
                             ('dt', DecisionTreeClassifier()),
                             ('rf', RandomForestClassifier())])
```

8. Build Bagging ensemble classifier on the training dataset.

In [47]:
```
import warnings
# Ignore the FutureWarning for the base_estimator parameter
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [48]:
```
print("22MCA1013")
from sklearn.ensemble import BaggingClassifier

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

```
22MCA1013
```

Out[48]:
```
BaggingClassifier(base_estimator=LogisticRegression(max_iter=1000),
                  n_estimators=100)
```

9. Build Boosting ensemble classifier on the training dataset.

In [49]:
```
print("22MCA1013")
from sklearn.ensemble import AdaBoostClassifier

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

```
22MCA1013
```

Out[49]:
```
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 [50]:
```
print("22MCA1013")
logistic_regression.score(X_test, y_test)
```

```
22MCA1013
```

Out[50]:
```
0.9666666666666667
```

In [51]:
```
print("22MCA1013")
decision_tree.score(X_test, y_test)
```

```
22MCA1013
```

Out[51]:
```
0.9666666666666667
```

In [52]:
```
print("22MCA1013")
random_forest.score(X_test, y_test)
```

```
22MCA1013
```

Out[52]:
```
0.9333333333333333
```

In [53]:
```
print("22MCA1013")
voting_classifier.score(X_test, y_test)
```

```
22MCA1013
```

Out[53]:
```
0.9666666666666667
```

In [54]:
```
print("22MCA1013")
bagging_classifier.score(X_test, y_test)
```

```
22MCA1013
```

Out[54]:
```
0.9666666666666667
```

In [55]:
```
print("22MCA1013")
adaboost_classifier.score(X_test, y_test)
```

```
22MCA1013
```

Out[55]:
```
1.0
```

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

In [56]:
```
print("22MCA1013")
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

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

```
22MCA1013
```

Out[56]:
```
0.9666666666666667
```

TUSHAR MATHUR

22MCA1013

MACHINE LEARNING LAB 10

In [4]:
```
#1.Load the dataset (pima-indians-diabetes).
import pandas as pd
data = pd.read_csv("pima-indians-diabetes.csv")
print(data)
```

```
      6  148  72  35    0  33.6  0.627  50  1
0     1   85  66  29    0  26.6  0.351  31  0
1     8  183  64   0    0  23.3  0.672  32  1
2     1   89  66  23   94  28.1  0.167  21  0
3     0  137  40  35  168  43.1  2.288  33  1
4     5  116  74   0    0  25.6  0.201  30  0
..   ..  ...  ..  ..  ...   ...    ...  .. ..
762  10  101  76  48  180  32.9  0.171  63  0
763   2  122  70  27    0  36.8  0.340  27  0
764   5  121  72  23  112  26.2  0.245  30  0
765   1  126  60   0    0  30.1  0.349  47  1
766   1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]
```

In [5]:
```
#2.check if there are missing values are present in the dataset.
data.isnull()
```

Out[5]:

	6	148	72	35	0	33.6	0.627	50	1
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
762	False	False	False	False	False	False	False	False	False
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False

767 rows × 9 columns

In [6]:
```
#3. Perform data preprocessing.
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("pima-indians-diabetes.csv",names=names)
print(data)
```

```
     preg  plas  pres  skin  test  mass   pedi  age  class
0       6   148    72    35     0  33.6  0.627   50      1
1       1    85    66    29     0  26.6  0.351   31      0
2       8   183    64     0     0  23.3  0.672   32      1
3       1    89    66    23    94  28.1  0.167   21      0
4       0   137    40    35   168  43.1  2.288   33      1
..    ...   ...   ...   ...   ...   ...    ...  ...    ...
763    10   101    76    48   180  32.9  0.171   63      0
764     2   122    70    27     0  36.8  0.340   27      0
765     5   121    72    23   112  26.2  0.245   30      0
766     1   126    60     0     0  30.1  0.349   47      1
767     1    93    70    31     0  30.4  0.315   23      0

[768 rows x 9 columns]
```

In [7]:
```
#4.Split dataset into test and train (20:80).
from sklearn.model_selection import train_test_split

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, random_s
```

In [8]:
```
#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, random_s
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)
```

```
[0 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 1 0 0 1 0 1 0 0 0 0 0
 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 0
 0 0 0 0 0 1 1 0 0 1 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 1 0
 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0
 1 0 1 0 0 0]
C:\Users\TR\AppData\Local\Temp\ipykernel_7348\2771513193.py:12: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples,), for example using ravel().
  classifier.fit(x_train, y_train)
```

In [10]:
```
#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("pima-indians-diabetes.csv",names=names)
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, random_s
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 0 1 0 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 1 0 0 0 0 1 1 1 1 0 0 0 0
 1 0 1 0 0 0]
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [11]:
```
#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("pima-indians-diabetes.csv",names=names)
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, random_s
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)
```

```
[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 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 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]
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [12]:
```
#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']
data=pd.read_csv("pima-indians-diabetes.csv",names=names)
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, random_s
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)
```

```
[0 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 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 1 0 0 0 0 1 0 0 1 0 0 0 0
 1 0 1 0 0 0]
C:\Users\TR\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:98: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:133: DataCo
nversionWarning: 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)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Conv
ergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [18]:

```
#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("C:/Users/student/Downloads/archive (2)/pima-indians-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, random_s
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)
```

```
[0 1 0 1 1 0 0 1 1 0 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]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Please ch
ange the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

In [13]:
```
#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']
data=pd.read_csv("pima-indians-diabetes.csv",names=names)
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, random_s
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)
```

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

In [15]:
```
#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, random_s
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("pima-indians-diabetes.csv",names=names)
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, random_s
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("pima-indians-diabetes.csv",names=names)
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, random_s
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 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 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 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 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 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 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 0 0 0 1 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 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0
 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 0 0 0 0 1 1 0 0 0 1 0
 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 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 1 0 0 0 0 0 0
 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0
 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 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 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1
 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1
 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 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
 1 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 1
 0 0 0 1 0 0 0 0 0 1 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 1 0 0 0 1 0 1 0 1 0 1 0 1 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 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 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
 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 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 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 1 0 0 1 0 0 0]
Logistic Regression
[1 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 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 1 0 0 1 0 0 1 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 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0
 0 1 0 0 1 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 1 0 1 1 0 1 0 0 0 0 0 0 1 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 1 1 0 0 1 0 0 0 1 1 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 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
 1 1 1 0 1 0 0 0 0 1 0 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 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 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]
```

```
C:\Users\TR\AppData\Local\Temp\ipykernel_7348\1429264599.py:11: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples,), for example using ravel().
  classifier.fit(x_test, y_test)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [16]:

```
#10.Evaluate  the  designed  models  from  step-5  to  step-8  with  appropriate cl
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("pima-indians-diabetes.csv",names=names)
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, random_s

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)


#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)
```

```
Random Forest Accuracy:  0.7337662337662337
Random Forest Precision:  0.6052631578947368
Random Forest Recall:  0.46938775510204084
Random Forest F1 score:  0.5287356321839081
Random Forest Classification
              precision    recall  f1-score   support
```

```
           0       0.78      0.86      0.81       105
           1       0.61      0.47      0.53        49

    accuracy                           0.73       154
   macro avg       0.69      0.66      0.67       154
weighted avg       0.72      0.73      0.72       154

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

C:\Users\TR\AppData\Local\Temp\ipykernel_7348\565755772.py:20: DataConversionWarnin
g: A column-vector y was passed when a 1d array was expected. Please change the sha
pe of y to (n_samples,), for example using ravel().
  classifier.fit(x_train, y_train)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\TR\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:

In [1]:
```
#Name: Tushar Mathur
#Reg: 22MCA1013
```

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]
```

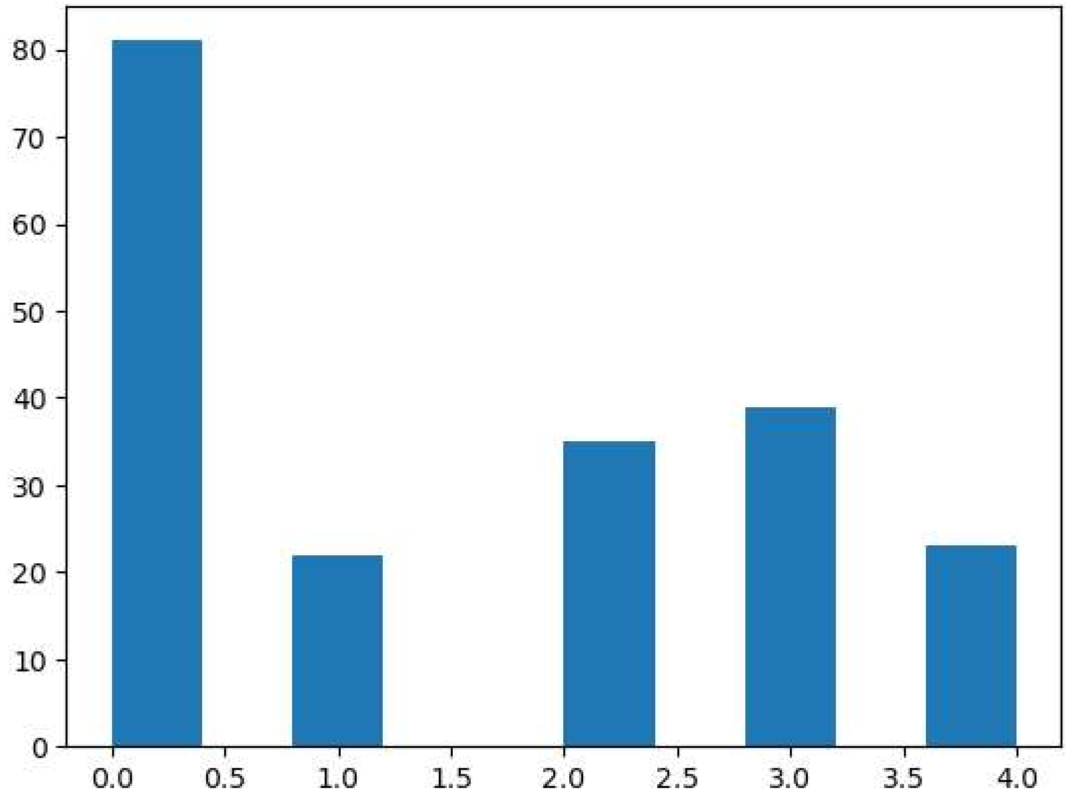
In [3]:
```
#2. If require perform data preprocessing.
data.isnull().sum()
```

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)
```

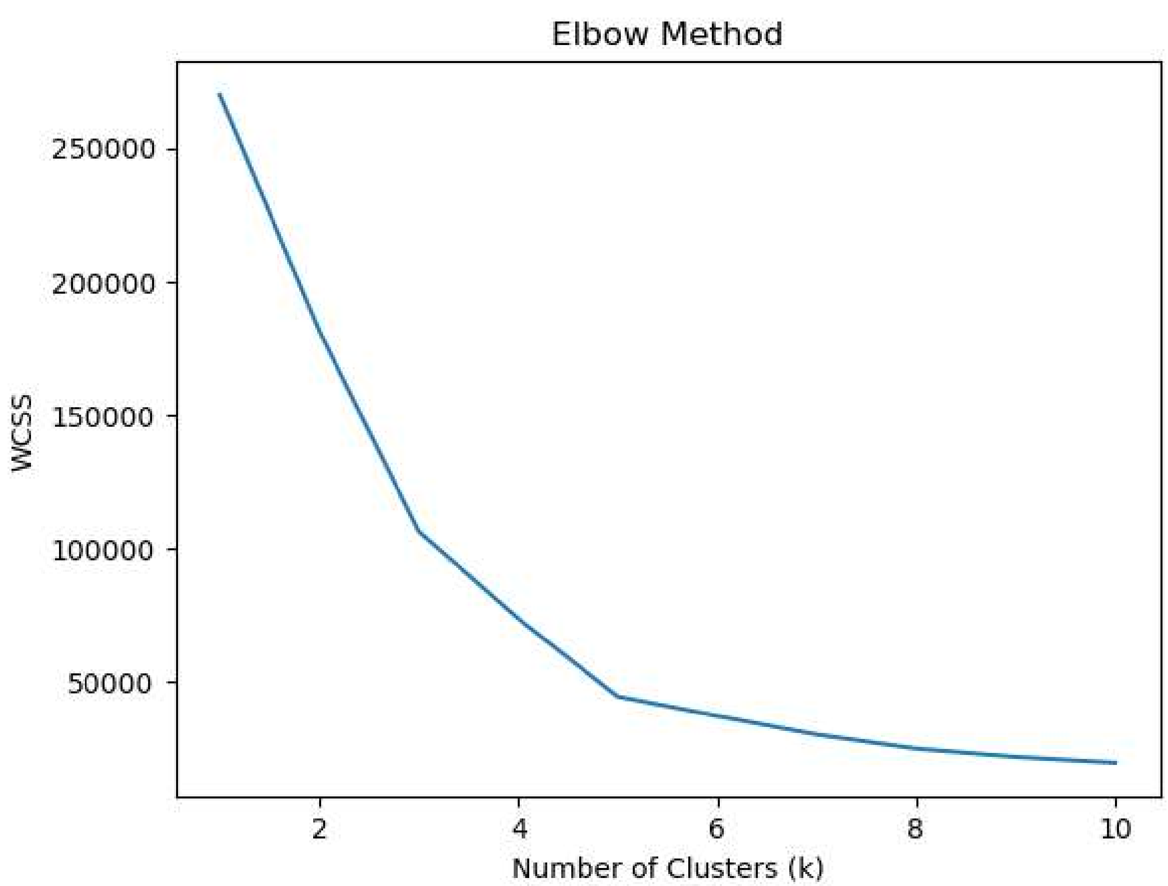
```
Cluster Labels:
[4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
 1 4 1 4 1 4 0 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 3 2 3 0 3 2 3 2 3 0 3 2 3 2 3 2 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()
```

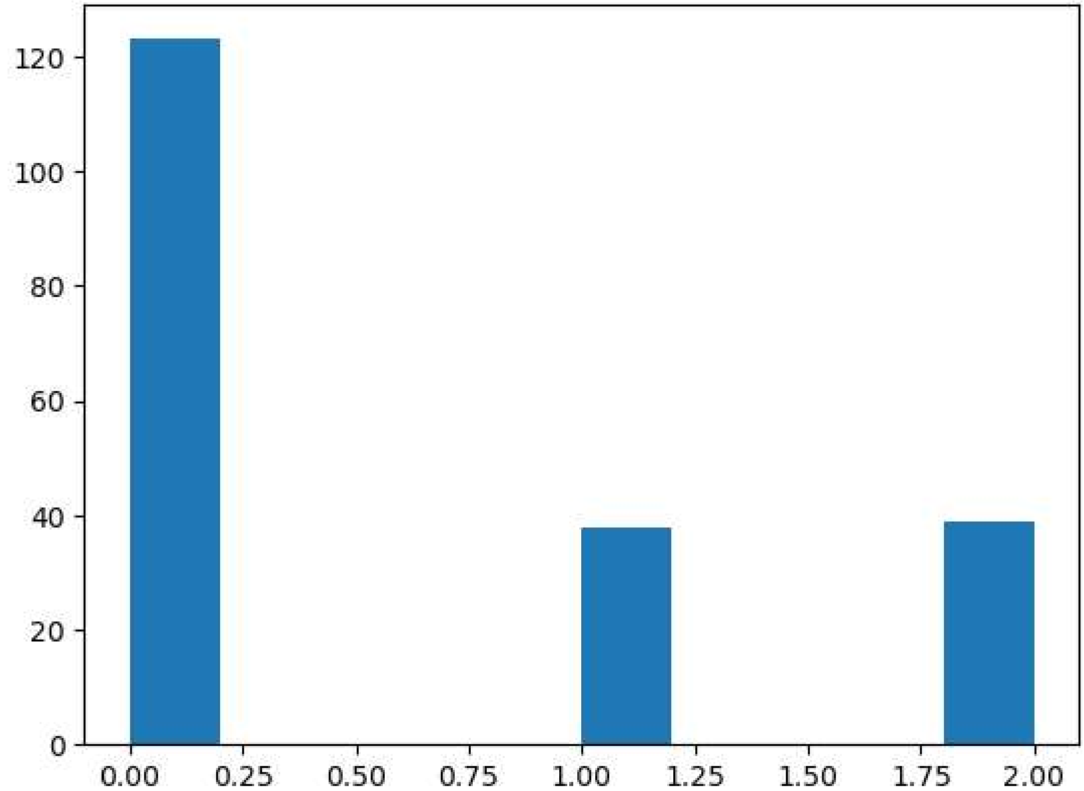
```
C:\Users\vipul\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: U
serWarning: KMeans is known to have a memory leak on Windows with MKL, when t
here are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



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 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 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 1 2 1
 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()
```



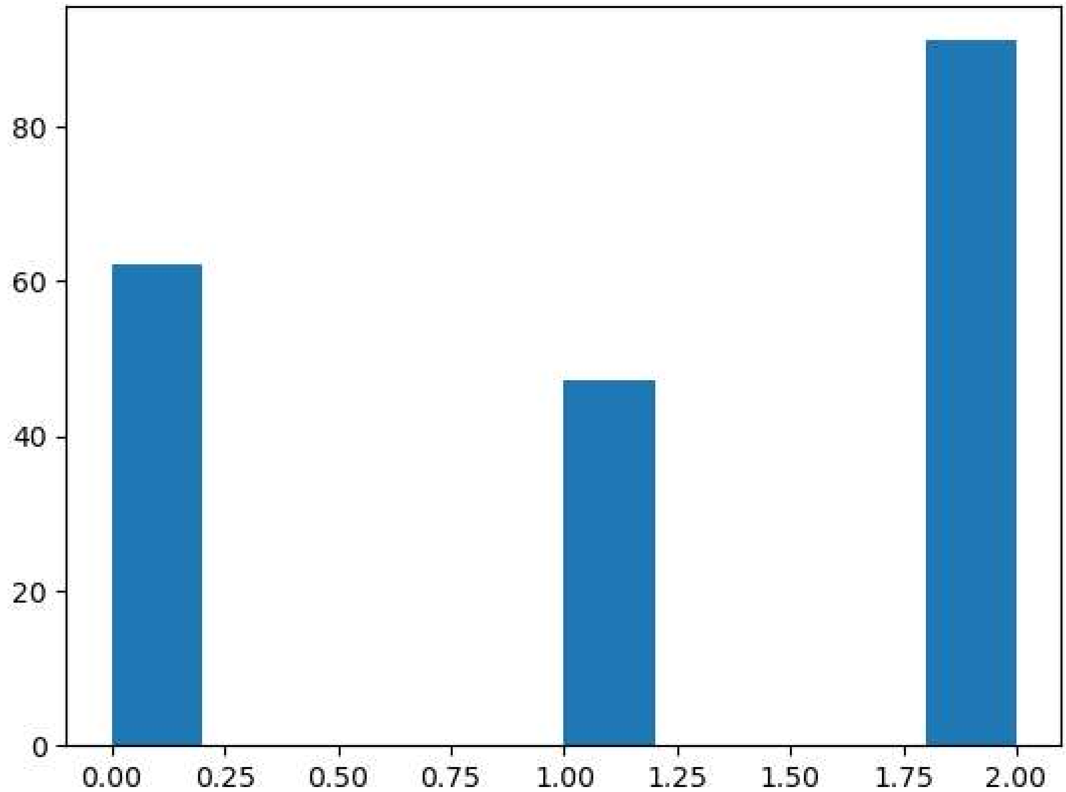
In [9]:
```
#8. Which attributes are strongly correlated with Spending Score?
import pandas as pd

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 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 2 0 1 0 2 0 1 0 1 0 2 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 2 0 1 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 [ ]: