ASSIGNMENT 01: Assignment on Practice of NumPy Library

Create a Numpy array containing the numbers from 1 to 10

In [1]:
```
import numpy as np
arr=np.arange(1,11)
print(arr)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

Convert a given python list to numpy array

In [2]:
```
import numpy as np
my_list=[1,2,3,4,5]
my_array=np.array(my_list)
print(my_array)
```

```
[1 2 3 4 5]
```

Create 50 evenly spaced numbers between 1 and 10

In [3]:
```
import  numpy as np
my_array=np.linspace(1,10,50)
print(my_array)
```

```
[ 1.          1.18367347  1.36734694  1.55102041  1.73469388  1.91836735
  2.10204082  2.28571429  2.46938776  2.65306122  2.83673469  3.02040816
  3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
  4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
  5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
  6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
  7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
  8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
  9.81632653 10.        ]
```

create a 5X5 matrix which contains random samples from standard normal distribution

In [4]:
```
import numpy as np
matrix=np.random.normal(0,1,(5,5))
print(matrix)
```

```
[[-0.13090119 -0.52915085 -1.27446489 -0.56230264 -0.26462096]
 [ 1.21907352  1.18424062 -1.90818347 -0.96528377 -0.73085963]
 [-0.17711287  0.39457746  0.00804115  0.76743507 -1.13530414]
 [ 0.87886451 -1.10269832 -0.70209975 -0.58849543  0.07141402]
 [ 0.91361729  0.24076211  1.56225522 -0.73524895  1.36505296]]
```

Create 20 random integer numbers between 1 to 100 as a numpy array

In [5]:
```
import numpy as np
rndm_int=np.random.randint(1,101,20)
rndm_int
```

Out[5]:
```
array([10, 22, 49,  1, 93,  4, 84, 70, 95, 80, 90, 88, 25, 25,  8, 26, 45,
       81, 64, 69])
```

Given the numpy array 'arr' reverse its elements and find its size

In [6]:
```
import numpy as np
arr=np.array([1,2,3,4,5])
rev_arr=np.flip(arr,0)
size_rev_arr=rev_arr.size
print("Original ",arr)
```

```
print("Reversed ",rev_arr)
print("Original ",size_rev_arr)
```

```
Original  [1 2 3 4 5]
Reversed  [5 4 3 2 1]
Original  5
```

Find the mean,median and standard deviation of the following numpy array

In [7]:
```
import numpy as np
arr=np.array([5,10,15,20,25])
mean=np.mean(arr)
median=np.median(arr)
std_deviation=np.std(arr)
print(arr)
print('Mean',mean)
print('Median',median)
print('Standard Deviation',std_deviation)
```

```
[ 5 10 15 20 25]
Mean 15.0
Median 15.0
Standard Deviation 7.0710678118654755
```

Create 3X3 matrix with all values set to 1

In [8]:
```
import numpy as np
matrix=np.ones((3,3))
print(matrix)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Create 3X3 matrix with all values set to 0

In [9]:
```
import numpy as np
matrix=np.zeros((3,3))
print(matrix)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Given two numpy arrays arr1 and arr2 concatenate them horizontally

In [10]:
```
import numpy as np
arr1=np.array([1,2,3,4,5])
arr2=np.array([6,7,8,9,10])
res=np.concatenate((arr1.reshape(1,-1),arr2.reshape(1,-1)),axis=1)
print(res)
```

```
[[ 1  2  3  4  5  6  7  8  9 10]]
```

Create a numpy array containing all even numbers from 0 to 20

In [11]:
```
import numpy as np
arr1=np.arange(0,21,2)
print(arr1)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20]
```

Element-wise multiplication of two numpy arrays

In [12]:
```
import numpy as np
arr1=np.array([1,2,3,4,5])
arr2=np.array([6,7,8,9,10])
res=arr1*arr2
print(res)
```

```
[ 6 14 24 36 50]
```

Reshape a numpy array into a 2X3 matrix

In [13]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6])
reshape=arr1.reshape(2,3)
print(reshape)
```

```
[[1 2 3]
 [4 5 6]]
```

Find the maximum and minimum values in a numpy array

In [14]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6])
print("maximum ",np.max(arr1))
print("maximum ",np.min(arr1))
```

```
maximum  6
maximum  1
```

Calculate the dot product of two numpy arrays

In [15]:
```
import numpy as np
arr1=np.array([1,2,3,4,5])
arr2=np.array([6,7,8,9,10])
dot_product=np.dot(arr1,arr2)
print(dot_product)
```

```
130
```

Create 2D numpy array with random floating point numbers between 0 to 1

In [16]:
```
import numpy as np
random_arr=np.random.rand(3,4)
print(random_arr)
```

```
[[0.83140146 0.58817119 0.34460686 0.69393307]
 [0.38158844 0.94324199 0.02596882 0.58110464]
 [0.9648661  0.02218178 0.88425532 0.55922529]]
```

Transpose a 2D numpy array

In [17]:
```
import numpy as np
arr1=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print("Transpose matrix is ",np.transpose(arr1))
```

```
Transpose matrix is  [[ 1  6]
 [ 2  7]
 [ 3  8]
 [ 4  9]
 [ 5 10]]
```

Create slice and set value 100

In [18]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9])
slice_arr=arr1[0:6]
slice_arr[:]=100
print(slice_arr)
```

```
[100 100 100 100 100 100]
```

Slice the first 2 rows and the last 2 cols of 2D numpy array (right top corner)

In [19]:
```
import numpy as np
arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
sliced_arr=arr[:2,1:]
print(sliced_arr)
```

```
[[2 3]
 [5 6]]
```

Given a numpy array of numbers how can you filter out all values greater than 5

In [20]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9])
print("Number greater than 5 ",arr1[arr1>=5])
```

```
Number greater than 5  [5 6 7 8 9]
```

Using boolean indexing, Extract all even numbers from a numpy array

In [21]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9,10])
print("Even numbers ",arr1[arr1%2==0])
```

```
Even numbers  [ 2  4  6  8 10]
```

2. Assignment on Practice of Pandas Library

Create Dataframe:

In [1]:
```
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

Create DataSeries:

In [2]:
```
import pandas as pd
s = pd.Series([2, 4, 6, 8, 10])
print(s)
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

Creating 2D dataframe

In [3]:
```
import pandas as pd
import numpy as np
data = {
 'w': np.random.randn(5),
 'x': np.random.randn(5),
 'y': np.random.randn(5),
 'Z': np.random.randn(5)
}
index = ['a', 'b', 'c', 'd', 'e']
df = pd.DataFrame(data, index=index)
print(df)
```

```
          w         x         y         Z
a  0.788508  0.359622 -0.357006  0.065455
b  1.372744 -0.704107 -1.374672  0.064281
c -2.175178 -1.512153 -0.448932  0.629077
d  0.330367 -1.090455 -0.932629 -0.294877
e -0.875626 -0.426688  0.759883 -0.263554
```

Printing specific columns

In [5]:
```
selected_columns = df[['y', 'Z']]
print(selected_columns)
```

```
          y         Z
a -0.357006  0.065455
b -1.374672  0.064281
c -0.448932  0.629077
d -0.932629 -0.294877
e  0.759883 -0.263554
```

Creating new column by the help of existing columns

In [7]:
```
df['new'] = df['w'] + df['y']
print(df)
```

```
          w         x         y         Z       new
a  0.788508  0.359622 -0.357006  0.065455  0.431502
b  1.372744 -0.704107 -1.374672  0.064281 -0.001929
c -2.175178 -1.512153 -0.448932  0.629077 -2.624110
d  0.330367 -1.090455 -0.932629 -0.294877 -0.602262
e -0.875626 -0.426688  0.759883 -0.263554 -0.115743
```

Deleting column from the datasets

In [8]:
```
df = df.drop('new', axis=1)
print(df)
```

```
          w         x         y         Z
a  0.788508  0.359622 -0.357006  0.065455
b  1.372744 -0.704107 -1.374672  0.064281
c -2.175178 -1.512153 -0.448932  0.629077
d  0.330367 -1.090455 -0.932629 -0.294877
e -0.875626 -0.426688  0.759883 -0.263554
```

Displaying specific rows

In [10]:
```
row_b = df.loc['b']
print(row_b)
```

```
w    1.372744
x   -0.704107
y   -1.374672
Z    0.064281
Name: b, dtype: float64
```

performing slicing operations

In [11]:
```
slice1 = df.loc[['a', 'b'], ['y', 'Z']]
print(slice1)
slice2 = df.loc[['c', 'd'], ['x', 'y']]
print(slice2)
```

```
          y         Z
a -0.357006  0.065455
b -1.374672  0.064281
          x         y
c -1.512153 -0.448932
d -1.090455 -0.932629
```

Converting the values into binary form

In [12]:
```
threshold = 0
boolean_df = df > threshold
print(boolean_df)
```

```
       w      x      y      Z
a   True   True  False   True
b   True  False  False   True
c  False  False  False   True
d   True  False  False  False
e  False  False   True  False
```

performing some filtering operations

In [13]:
```
filtered_df = df[(df['w'] > 0) & (df['y'] > 1)]
print(filtered_df)
```

```
Empty DataFrame
Columns: [w, x, y, Z]
Index: []
```

Performing concatination operation

In [14]:
```
import pandas as pd
data1 = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df1 = pd.DataFrame(data1)
print(df1)
data2 = {'A': [7, 8, 9], 'B': [10, 11, 12]}
df2 = pd.DataFrame(data2)
print(df2)
result = pd.concat([df1, df2])
print(result)
```

```
   A  B
0  1  4
1  2  5
2  3  6
   A   B
0  7  10
1  8  11
2  9  12
   A   B
0  1   4
1  2   5
2  3   6
0  7  10
1  8  11
2  9  12
```

Merging two data sets

In [15]:
```
import pandas as pd
data1 = {'key': ['A', 'B', 'C'], 'value1': [1, 2, 3]}
df1 = pd.DataFrame(data1)
print(df1)
data2 = {'key': ['B', 'C', 'D'], 'value2': [4, 5, 6]}
df2 = pd.DataFrame(data2)
print(df2)
merged_df = pd.merge(df1, df2, on='key')
print(merged_df)
```

```
  key  value1
0   A       1
1   B       2
2   C       3
  key  value2
0   B       4
1   C       5
2   D       6
  key  value1  value2
0   B       2       4
1   C       3       5
```

Performing join operation

In [16]:
```
import pandas as pd
data1 = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df1 = pd.DataFrame(data1, index=['X', 'Y', 'Z'])
print(df1)
data2 = {'C': [7, 8, 9], 'D': [10, 11, 12]}
df2 = pd.DataFrame(data2, index=['Y', 'Z', 'W'])
print(df2)
joined_df = df1.join(df2, how='inner') # 'inner' will keep only common index values
print(joined_df)
```

```
   A  B
X  1  4
Y  2  5
Z  3  6
   C   D
Y  7  10
Z  8  11
W  9  12
```

```
   A  B  C   D
Y  2  5  7  10
Z  3  6  8  11
```

3. Assignment on Find S algorithm. Let's assume we have a dataset of Enjoysport with sevenn attributes: Sky, Air, Temp, Humidity, Wind, Water, Forecast, EnjoySport. Divide the Dataset into two groups: "Specific Hypothesis" and "Generic Hypothesis" using the Find-S algorithm.

In [1]:
```
#This dataset consists of seven attributes including the output.
#Let's import the required libraries.
import pandas as pd
import numpy as np
```

In [2]:
```
#Let us understand how to read the data of the CSV file(dataset).
dataset=pd.read_csv("ENJOYSPORT.csv")
```

In [3]:
```
dataset
```

Out[3]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

The output of the above code would be the dataset EnjoySport.

Now, the next step is making an array of all attributes by excluding the output column.

In [4]:
```
arr=np.array(dataset)[:,:-1]
```

In [5]:
```
print(arr)
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

The next step is getting only the output values of the dataset.

In [6]:
```
target=np.array(dataset)[:,-1]
```

In [7]:
```
print(target)
```

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

- Instantiate the variable specific_hypothesis by the first positive example.
- Then for every positive example compare it with specific_hypothesis.
- If an attribute does not match, replace it with '?' else continue the process until the last positive example.

In [8]:
```
def train(Attributes,Target):

    for i, val in enumerate(Target):
        if val == 1:
```

```
            specific_hypothesis = Attributes[i].copy()
            break

    for i, val in enumerate(Attributes):
        if Target[i] == 1:
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis
```

The final value in specific_hypothesis is the most specific hypothesis of the dataset.

In [9]:
```
output=train(arr,target)
print("Hypothesis Space for Enjoy Sport Dataset",output)
```

```
Hypothesis Space for Enjoy Sport Dataset ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

This means that if the first four attributes of record are Sunny, Warm, High, Strong respectively then the output of that record is positive(Yes) irrespective of the last two attributes Water and forecast.

In [ ]:
```

```

4. Assignment on candidate elimination algorithm: consider a simplified dataset with two binary attributes ('A' and 'B') and a binary target variable ('Target'). Apply Candidate Elimination algorithm to find the most specific and most general hypotheses that cover all positive and negative examples

In [22]:
```
#import all the necessory libraries
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np   # linear algebra
```

In [23]:
```
#Load the Data set.
#(Enjoysport dataset is taken from Kaggle website as .csv file)
data=pd.read_csv("sample.csv")
data
```

Out[23]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

In [24]:
```
# Separating concept features from Target
concepts=np.array(data.iloc[:,0:-1])

# Isolating target into a separate DataFrame
# copying last column to target array
target=np.array(data.iloc[:,-1])

concepts
```

Out[24]:
```
array([['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'],
       ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'],
       ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'],
       ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']],
      dtype=object)
```

In [25]:
```
target
```

Out[25]:
```
array(['Yes', 'Yes', 'No', 'Yes'], dtype=object)
```

Building the algorithm

In [26]:
```
def learn(concepts, target):
    '''
    learn() function implements the learning method of the Candidate elimination algo
    Arguments:
        concepts - a data frame with all the features
        target - a data frame with corresponding output values
    '''
    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing to the same m
    specific_h = concepts[0].copy()
```

```
    print("\nInitialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h)
    print(general_h)
    # The learning iterations
    for i, h in enumerate(concepts):
        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):
                # For negative hyposthesis change values only  in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("\nSteps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    # find indices where we have empty rows, meaning those that are unchanged
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    # Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

```
Initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4
['Sunny' 'Warm' '?' 'Strong' '?' '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

5. Assignment on simple regression: Build an application where it can predict a salary based on year of experience using Single Variable Linear Regression (Use Salary dataset from Kaggle). Display the coefficient and intercept. Also visualize the results by plotting the graphs on both training and testing dataset.

In [1]:
```
#import all the necessory libraries
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as mlt
import numpy as np
dataset=pd.read_csv('Salary_dataset.csv')
#x=dataset.iloc[:,:-1].values

#assign the values of x and y for the model building.

x=dataset['YearsExperience'].values
x.reshape(30,1)
y=dataset['Salary'].values
y.reshape(30,1)
```

Out[2]:
```
array([[ 39344.],
       [ 46206.],
       [ 37732.],
       [ 43526.],
       [ 39892.],
       [ 56643.],
       [ 60151.],
       [ 54446.],
       [ 64446.],
       [ 57190.],
       [ 63219.],
       [ 55795.],
       [ 56958.],
       [ 57082.],
       [ 61112.],
       [ 67939.],
       [ 66030.],
       [ 83089.],
       [ 81364.],
       [ 93941.],
       [ 91739.],
       [ 98274.],
       [101303.],
       [113813.],
       [109432.],
       [105583.],
       [116970.],
       [112636.],
       [122392.],
       [121873.]])
```

In [39]:
```
#Draw the scattered graph of the given values x and y
mlt.scatter(x,y,color='red')
mlt.title('Salary vs Expirience')
mlt.xlabel('Experience')
mlt.ylabel('salary')
mlt.show()
```

In [25]:
```
#Reshape the x and values to set equal values of x and y
x=x.reshape(30,1)
y=y.reshape(30,1)
x.shape
```

Out[25]:
```
(30, 1)
```

Divide the complete dataset into training and testing data

In [26]:
```
# divide the dataset in some amount of training and testing data
from sklearn.model_selection import train_test_split

# random_state => seed value used by random number generator
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)
```

Implement Classifier based on Simple Linear Regression

In [27]:
```
from sklearn.linear_model import LinearRegression
```

In [28]:
```
#model implementation
model=LinearRegression()
```

In [31]:
```
#fit the implemented model with respect to x_train and y_train values
model.fit(x_train,y_train)
```

Out[31]:
```
LinearRegression()
```

In [32]:
```
#predict the model output with the help of predict function
y_predict=model.predict(x_test)
```

Find the mean squared error of the model

In [34]:
```
from sklearn.metrics import mean_squared_error
```

In [35]:
```
mean_squared_error(y_test,y_predict)
```

Out[35]:
```
23370078.80083297
```

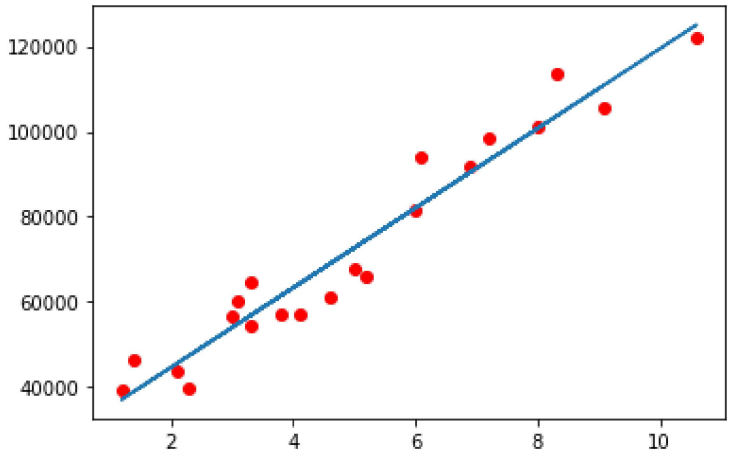
Plotting the Best-fit Linear Regression Graph

- Formula for the Linear Regression : Salary = B0 + B1*(Experience)
- B0 = intercept => salary when experience is 0, B1 = slope => increase in salary with unit increase in salary

In [38]:
```
mlt.scatter(x_train, y_train, color='red')
```

```
mlt.plot(x_train, model.predict(x_train))
```

Out[38]: [<matplotlib.lines.Line2D at 0x2b510601910>]



6.Assignment on multi-regression: Build an application where it can predict price of a house using multiple variable Linear Regression (Use USA_Housing dataset from Kaggle). Display all the coefficients.

Predicting Housing Prices for regions in the USA.

The data contains the following columns:

- Avg. Area Income': Avg. Income of residents of the city house is located in.
- Avg. Area House Age': Avg Age of Houses in same city.
- Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city.
- Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city.
- Area Population': Population of city house is located in.
- Price': Price that the house sold at.
- Address': Address for the house.

In [2]:
```
# importing librarires
import pandas as pd    # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as mlt
import numpy as np    # linear algebra


# loading csv data to dataframe
dataset=pd.read_csv('USA_housing.csv')
```

In [5]:
```
# checking out the Data
dataset.head()
```

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [6]:
```
#checking columns and total records
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
```

```
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

Generating descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN value.

In [7]:
```
dataset.describe()
```

Out[7]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

In [8]:
```
#Getting all Coulmn names
dataset.columns
```

Out[8]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

In [10]:
```
# Columns as Features
x=dataset[['Avg. Area Income','Avg. Area House Age','Avg. Area Number of Rooms','Avg.
           'Area Population']]
```

In [11]:
```
# Price is my Target Variable, what we trying to predict
y=dataset['Price']
```

Training the Model

Now that we've explored the data a bit, let's go ahead and split the data into training and testing sets.

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=101)
```

In [14]:
```
#importing the Linear Regression Algorithm
from sklearn.linear_model import LinearRegression
```

In [15]:
```
#creating LinearRegression Object
lm=LinearRegression()
```

In [16]:
```
#Training the Data Model
lm.fit(x_train,y_train)
```

Out[16]:
```
LinearRegression()
```

Predictions from our Model

Let's grab predictions off our test set and see how well it did!

In [17]:
```
predictions=lm.predict(x_test)
```

Calculating the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error

- MAE is the easiest to understand, because it's the average error.
- MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are loss functions, because we want to minimize them.

In [18]:
```
from sklearn import metrics
```

In [19]:
```
print('MAE:',metrics.mean_absolute_error(y_test,predictions))
```

```
MAE: 82288.22251914947
```

In [20]:
```
print('MSE:',metrics.mean_squared_error(y_test,predictions))
```

```
MSE: 10460958907.209057
```

In [21]:
```
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

```
RMSE: 102278.82922290935
```

7. Assignment on binary classification using Decision Tree Classifier: Build an application to decide on whether to play the tennis using Decision Tree. Use Tennis data from Kaggle. Do the required data processing. Display Accuracy score, Classification report and Confusion matrix.

In [1]:
```
#numpy and pandas initialization
import pandas as pd
import numpy as np


#Loading the PlayTennis data
dataset=pd.read_csv('play_tennis.csv')
```

In [2]:
```
#display the dataset values
dataset.head()
```

Out[2]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes

Preparing the Data (Data Slicing)

In [3]:
```
#machine learning algorithms can only learn from numbers (int, float, doubles .. )
#so let us encode it to int
```

In [4]:
```
outlook=dataset['outlook'].str.get_dummies(' ')
temp=dataset['temp'].str.get_dummies(' ')
humidity=dataset['humidity'].str.get_dummies(' ')
wind=dataset['wind'].str.get_dummies(' ')
play=dataset['play'].str.get_dummies(' ')
```

In [5]:
```
#drop the present dataset columns
dataset.drop(['day','outlook','temp','humidity','wind','play'],axis=1)
```

Out[5]:

0
1
2
3
4
5
6
7
8
9

10

11

12

13

In [6]:
```
#concat the new column values to the dataset back
dataset=pd.concat([outlook,temp,humidity,wind,play],axis=1)
```

In [7]:
```
dataset
```

Out[7]:

	Overcast	Rain	Sunny	Cool	Hot	Mild	High	Normal	Strong	Weak	No	Yes
0	0	0	1	0	1	0	1	0	0	1	1	0
1	0	0	1	0	1	0	1	0	1	0	1	0
2	1	0	0	0	1	0	1	0	0	1	0	1
3	0	1	0	0	0	1	1	0	0	1	0	1
4	0	1	0	1	0	0	0	1	0	1	0	1
5	0	1	0	1	0	0	0	1	1	0	1	0
6	1	0	0	1	0	0	0	1	1	0	0	1
7	0	0	1	0	0	1	1	0	0	1	1	0
8	0	0	1	1	0	0	0	1	0	1	0	1
9	0	1	0	0	0	1	0	1	0	1	0	1
10	0	0	1	0	0	1	0	1	1	0	0	1
11	1	0	0	0	0	1	1	0	1	0	0	1
12	1	0	0	0	1	0	0	1	0	1	0	1
13	0	1	0	0	0	1	1	0	1	0	1	0

In [8]:
```
#To divide our data into attribute set and Label:

x=dataset.drop(['No','Yes'],axis=1)                    #contains the attribute
```

In [9]:
```
x
```

Out[9]:

	Overcast	Rain	Sunny	Cool	Hot	Mild	High	Normal	Strong	Weak
0	0	0	1	0	1	0	1	0	0	1
1	0	0	1	0	1	0	1	0	1	0
2	1	0	0	0	1	0	1	0	0	1
3	0	1	0	0	0	1	1	0	0	1
4	0	1	0	1	0	0	0	1	0	1
5	0	1	0	1	0	0	0	1	1	0
6	1	0	0	1	0	0	0	1	1	0
7	0	0	1	0	0	1	1	0	0	1
8	0	0	1	1	0	0	0	1	0	1
9	0	1	0	0	0	1	0	1	0	1
10	0	0	1	0	0	1	0	1	1	0
11	1	0	0	0	0	1	1	0	1	0
12	1	0	0	0	1	0	0	1	0	1
13	0	1	0	0	0	1	1	0	1	0

In [10]:
```
y=dataset['Yes']     #contains the label
```

In [11]:
```
y
```

Out[11]:
```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: Yes, dtype: int64
```

In [12]:
```
#To divide our data into training and test sets:
from sklearn.model_selection import train_test_split
```

In [13]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.4,random_state=0)
```

Training and Making Predictions

In [14]:
```
from sklearn.tree import DecisionTreeClassifier    # import the classifier
```

In [15]:
```
# perform training
model= DecisionTreeClassifier()      # create a classifier object
```

In [16]:
```
model.fit(x_train,y_train)     # fit the classifier with X and Y data
```

Out[16]:
```
DecisionTreeClassifier()
```

In [17]:
```
#Predict the response for test dataset
y_predict=model.predict(x_test)
```

In [18]:
```
#Gemerate the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.33      1.00      0.50         1
           1       1.00      0.60      0.75         5

    accuracy                           0.67         6
   macro avg       0.67      0.80      0.62         6
weighted avg       0.89      0.67      0.71         6
```

In [19]:
```
#Find the confusion matrrix
from sklearn.metrics import confusion_matrix
```

In [20]:
```
print(confusion_matrix(y_test,y_predict))
```

```
[[1 0]
 [2 3]]
```

In [21]:
```
# Model Accuracy, how often is the classifier correct?
from sklearn.metrics import accuracy_score
```

In [22]:
```
accuracy_score(y_test,y_predict)
```

Out[22]:
```
0.6666666666666666
```

8. Assignment on binary classification using Perceptron: Implement Perceptron model. Use this model to classify a patient that she is having cancer or not. Use Breast cancer dataset from sklearn library. Display Accuracy score, Classification report and Confusion matrix.

In [1]:
```
#import all the necessory libraries.
import sklearn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
import numpy as np
```

In [2]:
```
# Load the Breast Cancer dataset
data = load_breast_cancer()
```

In [3]:
```
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

In [4]:
```
#creating the dataset with the help of above attributes
df=pd.DataFrame(data.data,columns=data.feature_names)
```

In [5]:
```
#Describe the dataset
df.describe()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

8 rows × 30 columns

In [6]:
```
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split

train, test, train_labels, test_labels = train_test_split(features, labels,test_size
```

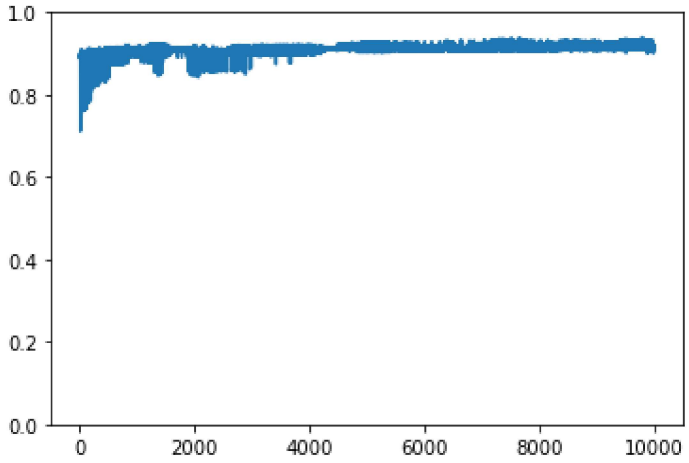
In [7]:
```
# Create a Perceptron model
class Perceptron:
    def __init__(self):
        self.w=None
        self.b=None
```

```
    def model(self,x):
        return 1 if (np.dot(self.w,x)>=self.b) else 0

    # Make predictions
    def predict(self,X):
        Y=[]
        for x in X:
            result=self.model(x)
            Y.append(result)
        return np.array(Y)

    # Train the model
    def fit(self,X,Y,epochs=1,lr=1):
        self.w=np.ones(X.shape[1])
        self.b=0

        accuracy={}
        max_accuracy=0

        wt_matrix=[]
        for i in range(epochs):
            for x,y in zip(X,Y):
                y_pred=self.model(x)
                if y==1 and y_pred==0:
                    self.w=self.w+lr*x
                    self.b=self.b-lr*1
                elif y==0 and y_pred==1:
                    self.w=self.w-lr*x
                    self.b=self.b+lr*1

            wt_matrix.append(self.w)

            accuracy[i]=accuracy_score(self.predict(X),Y)
            if(accuracy[i]>max_accuracy):
                max_accuracy=accuracy[i]
                chkpw=self.w
                chkpb=self.b
        self.w=chkpw
        self.b=chkpb
        print(max_accuracy)
        plt.plot(accuracy.values())
        plt.ylim([0,1])
        plt.show()

        return np.array(wt_matrix)
```

In [8]:
```
#Calling the function Perceptron for creating object
per=Perceptron()
```

In [12]:
```
wt_matrics=per.fit(train,train_labels,10000,0.5)
```

```
0.9396325459317585
```



In [13]:
```
y_predict=per.predict(test)
```

In [14]:
```
# Calculate classification report
from sklearn.metrics import classification_report
print(classification_report(test_labels, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93        67
           1       0.97      0.96      0.96       121

    accuracy                           0.95       188
   macro avg       0.95      0.95      0.95       188
weighted avg       0.95      0.95      0.95       188
```

Evaluate accuracy

In [15]:
```
from sklearn.metrics import accuracy_score
print('Accuracy score : ',accuracy_score(test_labels, y_predict))
```

```
Accuracy score :  0.9521276595744681
```

In [16]:
```
# Calculate confusion matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(test_labels, y_predict))
```

```
[[ 63   4]
 [  5 116]]
```

9. Assignment on Multi classification using Multilayer Perceptron (MLP): Buid an application to classify a given flower into its specie using MLP. Use Iris dataset from Kaggle. Display Accuracy score, Classification report and Confusion matrix.

In [1]:
```
# Importing Dependencies
import sklearn
import pandas as pd     # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np      # linear algebra
```

In [2]:
```
#load the Iris flower dataset, which is in the "../input/" directory
dataset=pd.read_csv('iris.csv')      # the iris dataset is now a Pandas DataFrame
```

In [3]:
```
# first five observations
dataset.head()
```

Out[3]:

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

In [4]:
```
# Number of observations and missing values.
# There are 150 observations and no nan value
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [5]:
```
#Recreate the dataset with the help of existing attributes of the dataset
x=dataset[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
y = dataset[['Species']].replace(['Iris-setosa','Iris-versicolor','Iris-virginica'],[

df = pd.concat([x, y], axis=1)
df.sample(n=5)
```

Out[5]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
72	6.3	2.5	4.9	1.5	1
134	6.1	2.6	5.6	1.4	2
17	5.1	3.5	1.4	0.3	0

60	5.0	2.0	3.5	1.0	1
136	6.3	3.4	5.6	2.4	2

In [6]:
```
# Split data into a training set and a testing set.
# train_test_split shuffle the data before the split (shuffle=True by default)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=

#Standardising the training data
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)

X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

In [7]:
```
#Building the model with the help of Multi Layered Classifier
from sklearn.neural_network import MLPClassifier
```

In [8]:
```
clf = MLPClassifier(hidden_layer_sizes=(10, 10,10), max_iter=1000)       # create a cl
```

In [9]:
```
clf.fit(X_train, y_train.values.ravel())      # fit the classifier with X and Y data
```

Out[9]:
```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

In [10]:
```
# Now get predictions from the model and create a confusion matrix and a classificati
y_pred = clf.predict(X_test)
```

In [11]:
```
# Model Accuracy, how often is the classifier correct?
from sklearn.metrics import accuracy_score
print('Accuracy Score : ',accuracy_score(y_test, y_pred))
```

```
Accuracy Score :  0.9555555555555556
```

In [12]:
```
#Generate the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.94      0.94      0.94        18
           2       0.92      0.92      0.92        13

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

In [13]:
```
#Find the confusion matrrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
```

10. Assignment on regression using KNN: Build an application where it can predict a salary based on year of experience using KNN (Use Salary dataset from Kaggle).

In [1]:
```
# Importing Dependencies
import pandas as pd     # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as mlt
import numpy as np      # linear algebra
dataset=pd.read_csv('Salary_dataset.csv')
```

In [2]:
```
dataset.describe()
```

Out[2]:

	Unnamed: 0	YearsExperience	Salary
count	30.000000	30.000000	30.000000
mean	14.500000	5.413333	76004.000000
std	8.803408	2.837888	27414.429785
min	0.000000	1.200000	37732.000000
25%	7.250000	3.300000	56721.750000
50%	14.500000	4.800000	65238.000000
75%	21.750000	7.800000	100545.750000
max	29.000000	10.600000	122392.000000

In [3]:
```
# Number of observations and missing values.
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       30 non-null     int64  
 1   YearsExperience  30 non-null     float64
 2   Salary           30 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 848.0 bytes
```

In [4]:
```
# Split data into a training set and a testing set.
from sklearn.model_selection import train_test_split
train,test=train_test_split(dataset,test_size=0.3)
```

In [5]:
```
x_train=train['YearsExperience']

y_train=train['Salary']
```

In [6]:
```
#Graphical representation of Testing data
viz_test=mlt
viz_test.scatter(x_train,y_train,color='red')
viz_test.title('Salary Vs Experience')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [7]:
```
#Reset the traing and testing datas of the dataset
x_train=train.drop('Salary',axis=1)
x_test=test.drop('Salary',axis=1)
y_test=test['Salary']
```

In [8]:
```
x_test
```

Out[8]:

	Unnamed: 0	YearsExperience
13	13	4.2
27	27	9.7
4	4	2.3
24	24	8.8
0	0	1.2
7	7	3.3
18	18	6.0
21	21	7.2
14	14	4.6

In [9]:
```
y_test
```

Out[9]:
```
13     57082.0
27    112636.0
4      39892.0
24    109432.0
0      39344.0
7      54446.0
18     81364.0
21     98274.0
14     61112.0
Name: Salary, dtype: float64
```

In [10]:
```
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [11]:
```
model=neighbors.KNeighborsRegressor(n_neighbors=3)
model.fit(x_train,y_train)
```

Out[11]:
```
KNeighborsRegressor(n_neighbors=3)
```

In [12]:
```
#Predict Test Set Results
pred=model.predict(x_test)
print(pred)
```

```
[ 60230.66666667 114981.66666667  45967.         106899.66666667
  42488.          60413.33333333  81020.         102285.
  63642.33333333]
```

Evaluate Model Performance :

In [13]:
```
error=sqrt(mean_squared_error(y_test,pred))
print('Error : ',error)
```

```
Error :  3754.0709442312077
```

11. Assignment on Classification using KNN: Buid an application to classify a given flower into its specie using KNN (Use Iris dataset from sklearn library)

In [1]:
```
# Import libraries
import pandas as pd
from sklearn.datasets import load_iris
import numpy as np
```

In [2]:
```
# import iris dataset
irisData = load_iris()
```

In [3]:
```
#assigning iris data to the variable x
X = irisData.data

#assigning iris specieses to y
y = irisData.target

#Integer represent the species : 0-setosa, 1-versicolor, 2-virginica
y
```

Out[3]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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])
```

In [4]:
```
# split the data into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)
```

In [6]:
```
#Importing KNeighborsClassifier from Scikit-learn.
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=7)

## Train the classifier on the training set
model.fit(X_train, y_train)

#Predicting the testing data using predict() function.
pred=model.predict(X_test)
```

In [7]:
```
#Checking the predicting output with real output which is stored in y_test.
y_test
```

Out[7]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [8]:
```
# Calculate the accuracy of the classifier
from sklearn.metrics import accuracy_score
print('Accuracy Score : ',accuracy_score(y_test, pred))
```

```
Accuracy Score :  0.9666666666666667
```

In [9]:
```
#Classification Report: It consists of precision, recall and F1 score.
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00        10
              1       1.00      0.89      0.94         9
              2       0.92      1.00      0.96        11

    accuracy                            0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [10]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))
```

```
[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
```

12. Assignment on Naïve Bayes classifier: Using Naïve Bayes classifier, build an application to classify a given text. Use text data from sklearn (Text classification)

In [1]:
```
#Import all the necessory libraries
import numpy as np   # linear algebra
import matplotlib.pyplot as plt   # for data visualization purposes
import seaborn as sns; sns.set()   # for statistical data visualization
```

In [2]:
```
#load the dataset from the scikit-learn
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups()
data.target_names
```

Out[2]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [3]:
```
#make categorical differentiations
categories = ['talk.religion.misc', 'soc.religion.christian',
              'sci.space', 'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
```

In [4]:
```
#Display the categorires of the dataset
print(train.data[5])
```

```
From: dmcgee@uluhe.soest.hawaii.edu (Don McGee)
Subject: Federal Hearing
Originator: dmcgee@uluhe
Organization: School of Ocean and Earth Science and Technology
Distribution: usa
Lines: 10


Fact or rumor....?  Madalyn Murray O'Hare an atheist who eliminated the
use of the bible reading and prayer in public schools 15 years ago is now
going to appear before the FCC with a petition to stop the reading of the
Gospel on the airways of America.  And she is also campaigning to remove
Christmas programs, songs, etc from the public schools.  If it is true
then mail to Federal Communications Commission 1919 H Street Washington DC
20054 expressing your opposition to her request.  Reference Petition number
```

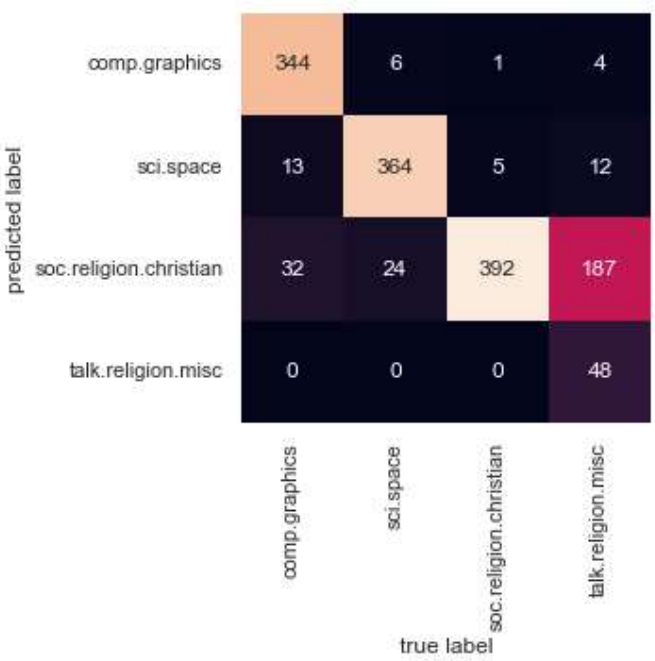
2493.

Build the model

In [5]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

# create a classifier object
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

In [6]:
```
model.fit(train.data, train.target)     # fit the classifier

#Predict the response for test dataset
labels = model.predict(test.data)
```

In [7]:
```
#Scemantically represent the confusion matrix
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names, yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



In [8]:
```
#display the content of cinfussion matrix
print(mat)
```

```
[[344  13  32   0]
 [  6 364  24   0]
 [  1   5 392   0]
 [  4  12 187  48]]
```

In [9]:
```
#Predict the categories of the texts
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

In [10]:
```
predict_category('sending a payload to the ISS')
```

Out[10]:
```
'sci.space'
```

In [11]:
```
predict_category('discussing islam vs atheism')
```

Out[11]:
```
'soc.religion.christian'
```

In [12]:
```
predict_category('determining the screen resolution')
```

Out[12]:
```
'comp.graphics'
```

In [ ]:
```

```

13. Assignment on Dimensionality Reduction using PCA.

In [2]:
```
import pandas as pd

# load dataset into Pandas DataFrame
df = pd.read_csv("iris.csv")
df.head()
```

Out[2]:

	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

In [11]:
```
from sklearn.preprocessing import StandardScaler
features = ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
# Separating out the features
x = df.loc[:,features].values
# Separating out the target
y = df.loc[:,['Species']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state
```

In [12]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2', 'principa
```

In [14]:
```
finalDf = pd.concat([principalDf, df[['Species']]], axis = 1)
```

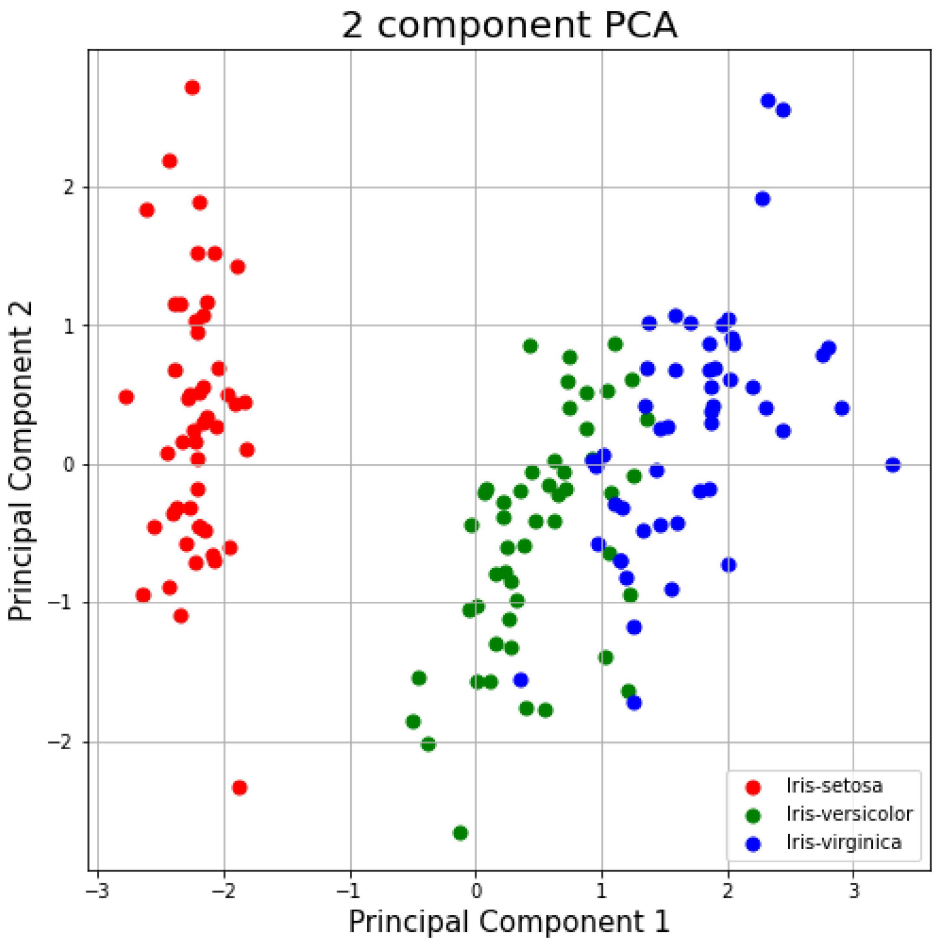
In [15]:
```
finalDf.head()
```

Out[15]:

	principal component 1	principal component 2	principal component 3	Species
0	-2.264542	0.505704	-0.121943	Iris-setosa
1	-2.086426	-0.655405	-0.227251	Iris-setosa
2	-2.367950	-0.318477	0.051480	Iris-setosa
3	-2.304197	-0.575368	0.098860	Iris-setosa
4	-2.388777	0.674767	0.021428	Iris-setosa

In [17]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Species'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
```

```
                , finalDf.loc[indicesToKeep, 'principal component 2']
                , c = color
                , s = 50)
ax.legend(targets)
ax.grid()
```



In [18]:
```
pca.explained_variance_ratio_
```

Out[18]:
```
array([0.72770452, 0.23030523, 0.03683832])
```

In [19]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [20]:
```
#using original data
model = RandomForestClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
C:\Users\danes\AppData\Local\Temp\ipykernel_7964\1694027823.py:3: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please change the shap
e of y to (n_samples,), for example using ravel().
  model.fit(X_train, y_train)
```

In [21]:
```
predictions
```

Out[21]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
```

```
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor'], dtype=object)
```

In [22]:
```
#Display the accuracy score
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

Out[22]:
```
0.9555555555555556
```

In [24]:
```
# Separating out the features
x = finalDf.drop(["Species"], axis = 1)
x = StandardScaler().fit_transform(x)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state
```

In [25]:
```
predictions
```

Out[25]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor'], dtype=object)
```

In [26]:
```
#Accuracy score of the altered dataset
accuracy_score(y_test, predictions)
```

Out[26]:
```
0.9555555555555556
```

In [ ]:
```

```

14. Assignment on Image Processing: Build an application to recognise a Digit from an image using MLP (Use Digit image Dataset from sklearn)

Import Necessary Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.metrics import accuracy_score
```

Load the digits dataset from sklearn and split it into training and testing sets.

In [2]:
```
# Load digits dataset
digits = datasets.load_digits()

# Flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, digits.target, test_size=0.
```

In [3]:
```
def plot_multi(i):
    '''Plots 16 digits, starting with digit i'''
    nplots = 16
    fig = plt.figure(figsize=(15,15))
    for j in range(nplots):
        plt.subplot(4,4,j+1)
        plt.imshow(digits.images[i+j], cmap='binary')
        plt.title(digits.target[i+j])
        plt.axis('off')
    plt.show()
```

In [4]:
```
plot_multi(0)
```

Create an MLP model and train it using the training set.

In [5]:
```
# Create MLP model
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500)

# Train the model
mlp.fit(X_train, y_train)
```

Out[5]:
```
MLPClassifier(max_iter=500)
```

Use the trained model to make predictions on the test set.

In [6]:
```
# Make predictions on the test set
y_pred = mlp.predict(X_test)
```

Evaluate the performance of the model using accuracy or other metrics.

In [7]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[7]:
```
0.9722222222222222
```