**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**COURSE TITLE**

***Submitted by***

**MUSKAN GUPTA(1BM19CS091)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “LAB COURSE **MACHINE LEARNING**” carried out by **MUSKAN GUPTA (1BM19CS091),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning **- (20CS6PCMAL)** work prescribed for the said degree.

**Saritha A. N** **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | **FIND S** |  |
| **2** | **CANDIDATE ELIMINATION** |  |
| **3** | **DECISION TREE** |  |
| **4** | **NAIVE BAYES** |  |
| **5** | **LINEAR REGRESSION** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

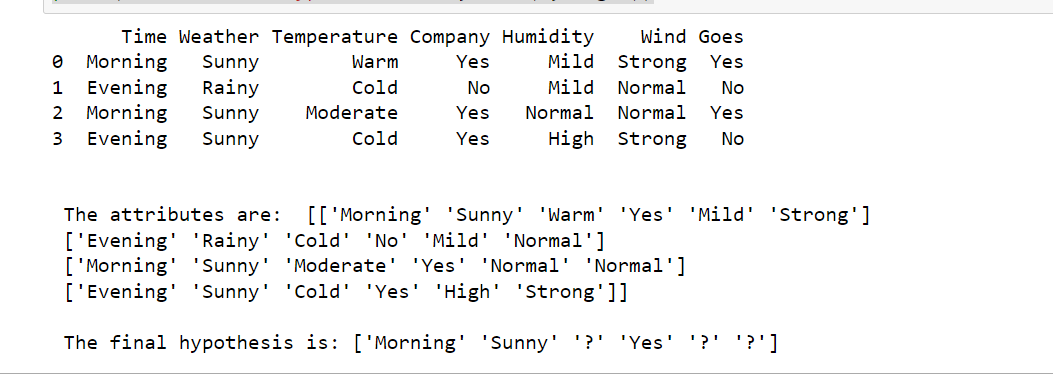
**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Ability to apply the different learning algorithms. |
| CO2 | Ability to analyze the learning techniques for given dataset |
| CO3 | Ability to design a model using machine learning to solve a problem. |
| CO4 | Ability to conduct practical experiments to solve problems using appropriate machine learning  Techniques. |

1.Implement and demonstrate the FIND-S algorithm for finding the

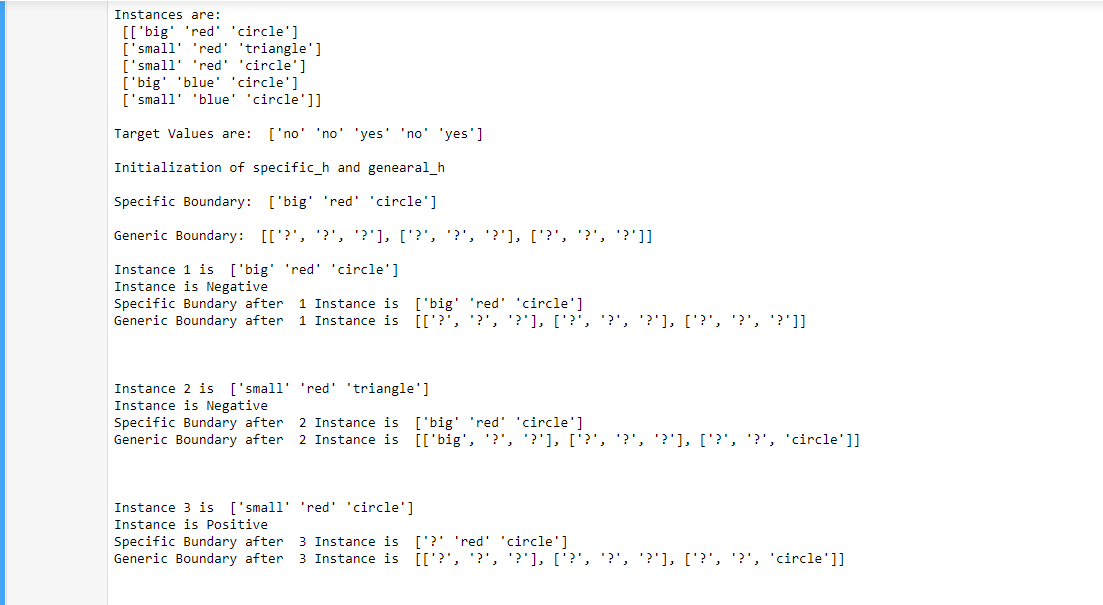
most specific hypothesis based on a given set of training data samples.

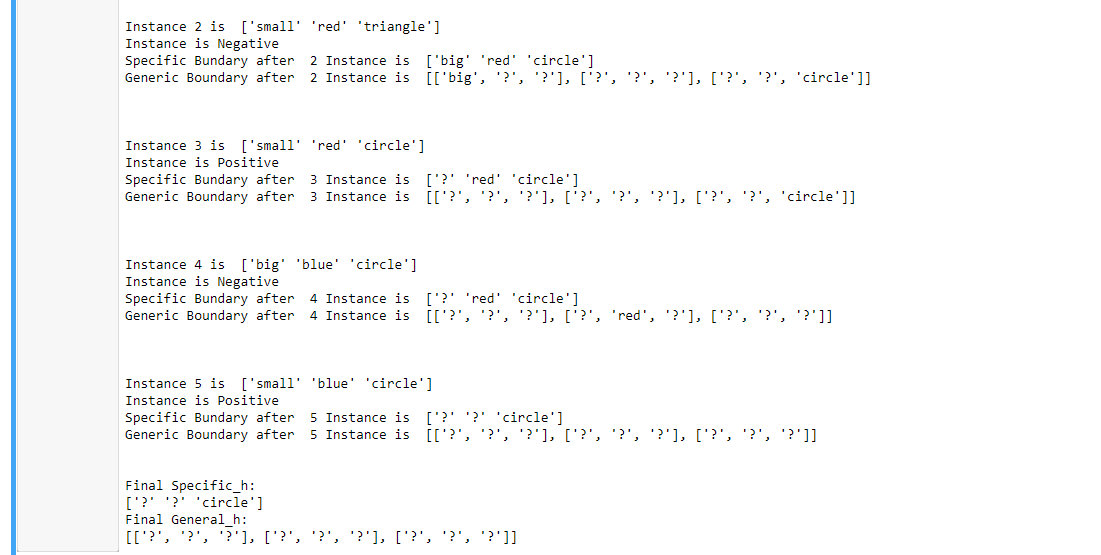
|  |
| --- |
| import pandas as pd |
|  | import numpy as np |
|  |  |
|  |  |
|  | data = pd.read\_csv("data - Sheet1.csv") |
|  | print(data,"\n") |
|  |  |
|  | d = np.array(data)[:,:-1] |
|  | print("\n The attributes are: ",d) |
|  |  |
|  | target = np.array(data)[:,-1] |
|  |  |
|  | def findS(c,t): |
|  | for i, val in enumerate(t): |
|  | if val == "Yes": |
|  | specific\_hypothesis = c[i].copy() |
|  | break |
|  |  |
|  | for i, val in enumerate(c): |
|  | if t[i] == "Yes": |
|  | for x in range(len(specific\_hypothesis)): |
|  | if val[x] != specific\_hypothesis[x]: |
|  | specific\_hypothesis[x] = '?' |
|  | else: |
|  | pass |
|  |  |
|  | return specific\_hypothesis |
|  |  |
|  | print("\n The final hypothesis is:",findS(d,target)) |



2.For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

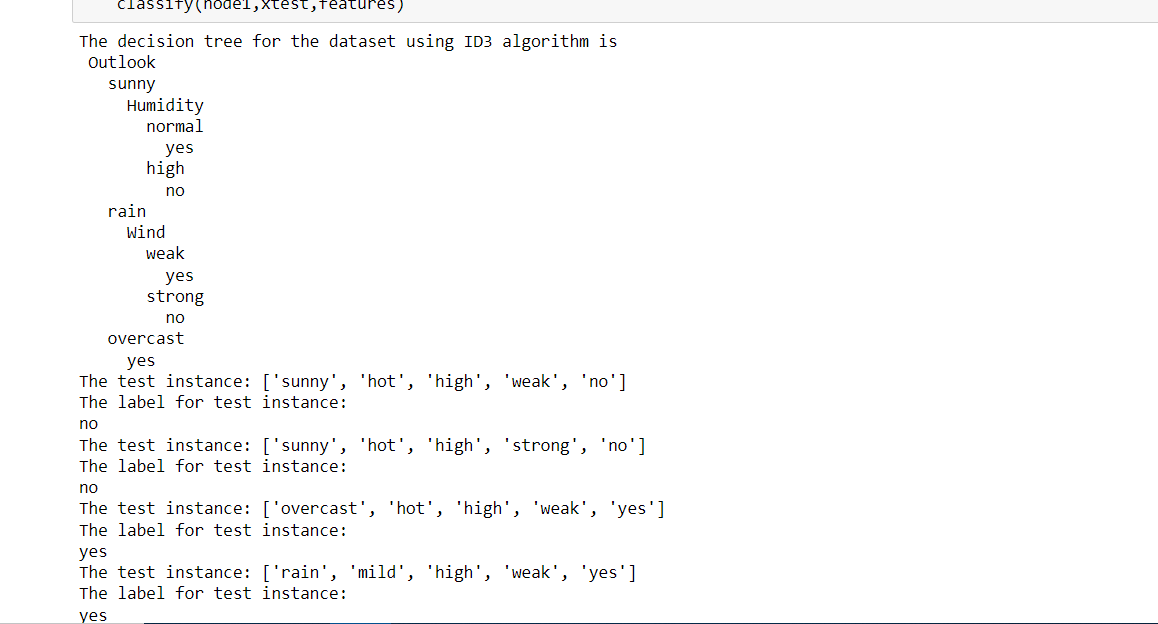
|  |
| --- |
| import numpy as np |
|  | import pandas as pd |
|  | data = pd.read\_csv('shape.csv') |
|  | concepts = np.array(data.iloc[:,0:-1]) |
|  | print("\nInstances are:\n",concepts) |
|  | target = np.array(data.iloc[:,-1]) |
|  | print("\nTarget Values are: ",target) |
|  | def learn(concepts, target): |
|  | specific\_h = concepts[0].copy() |
|  | print("\nInitialization of specific\_h and genearal\_h") |
|  | print("\nSpecific Boundary: ", specific\_h) |
|  | general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))] |
|  | print("\nGeneric Boundary: ",general\_h) |
|  |  |
|  | for i, h in enumerate(concepts): |
|  | print("\nInstance", i+1 , "is ", h) |
|  | if target[i] == "yes": |
|  | print("Instance is Positive ") |
|  | for x in range(len(specific\_h)): |
|  | if h[x]!= specific\_h[x]: |
|  | specific\_h[x] ='?' |
|  | general\_h[x][x] ='?' |
|  |  |
|  | if target[i] == "no": |
|  | print("Instance is Negative ") |
|  | for x in range(len(specific\_h)): |
|  | if h[x]!= specific\_h[x]: |
|  | general\_h[x][x] = specific\_h[x] |
|  | else: |
|  | general\_h[x][x] = '?' |
|  |  |
|  | print("Specific Bundary after ", i+1, "Instance is ", specific\_h) |
|  | print("Generic Boundary after ", i+1, "Instance is ", general\_h) |
|  | print("\n") |
|  |  |
|  | indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']] |
|  | for i in indices: |
|  | general\_h.remove(['?', '?', '?', '?', '?', '?']) |
|  | return specific\_h, general\_h |
|  | s\_final, g\_final = learn(concepts, target) |
|  |  |
|  | print("Final Specific\_h: ", s\_final, sep="\n") |
|  | print("Final General\_h: ", g\_final, sep="\n") |

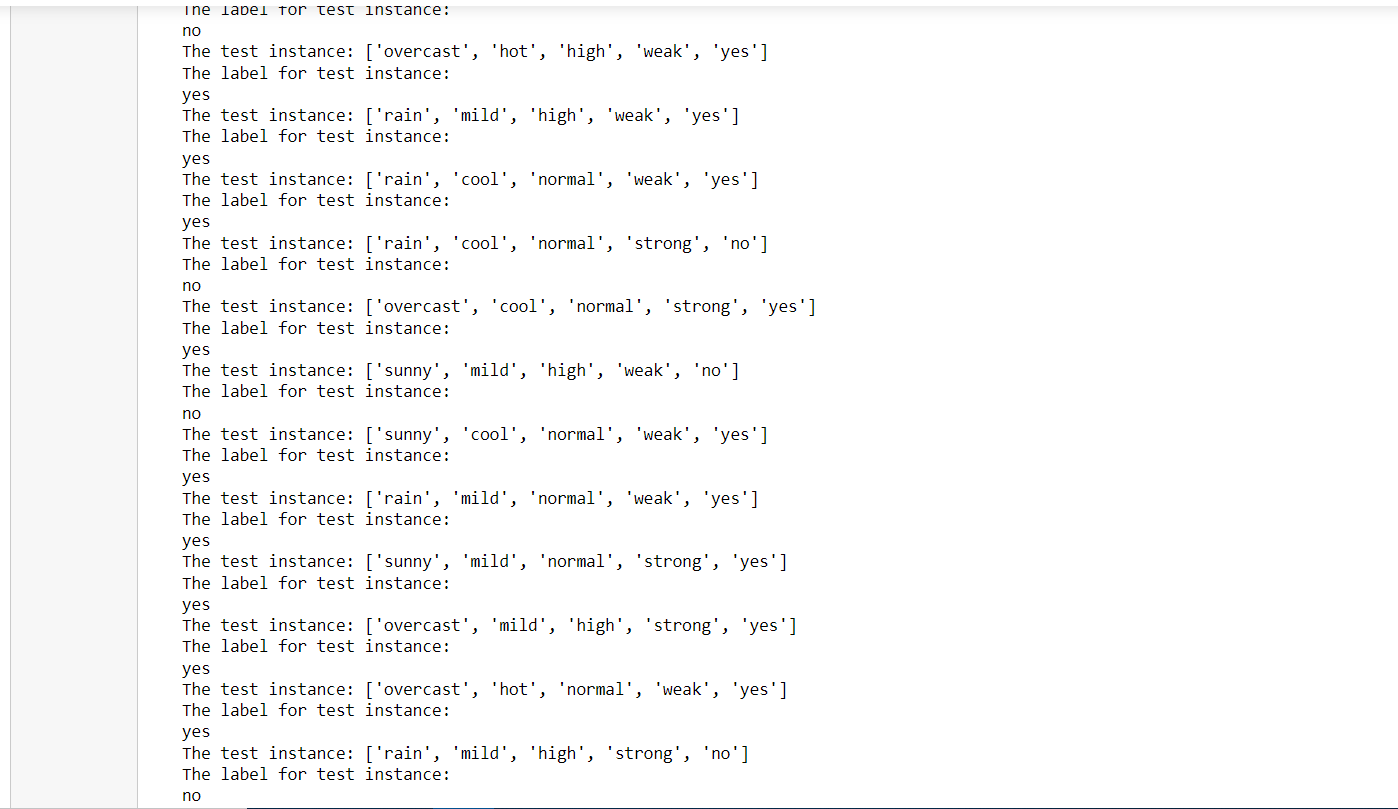




3.Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

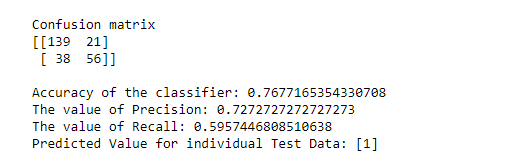
|  |
| --- |
| import math |
|  | import csv |
|  | def load\_csv(filename): |
|  | lines=csv.reader(open(“data.csv”,"r")) |
|  | dataset = list(lines) |
|  | headers = dataset.pop(0) |
|  | return dataset,headers |
|  |  |
|  | class Node: |
|  | def \_\_init\_\_(self,attribute): |
|  | self.attribute=attribute |
|  | self.children=[] |
|  | self.answer="" |
|  |  |
|  | def subtables(data,col,delete): |
|  | dic={} |
|  | coldata=[row[col] for row in data] |
|  | attr=list(set(coldata)) |
|  |  |
|  | counts=[0]\*len(attr) |
|  | r=len(data) |
|  | c=len(data[0]) |
|  | for x in range(len(attr)): |
|  | for y in range(r): |
|  | if data[y][col]==attr[x]: |
|  | counts[x]+=1 |
|  |  |
|  | for x in range(len(attr)): |
|  | dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])] |
|  | pos=0 |
|  | for y in range(r): |
|  | if data[y][col]==attr[x]: |
|  | if delete: |
|  | del data[y][col] |
|  | dic[attr[x]][pos]=data[y] |
|  | pos+=1 |
|  | return attr,dic |
|  |  |
|  | def entropy(S): |
|  | attr=list(set(S)) |
|  | if len(attr)==1: |
|  | return 0 |
|  |  |
|  | counts=[0,0] |
|  | for i in range(2): |
|  | counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)\*1.0) |
|  |  |
|  | sums=0 |
|  | for cnt in counts: |
|  | sums+=-1\*cnt\*math.log(cnt,2) |
|  | return sums |
|  |  |
|  | def compute\_gain(data,col): |
|  | attr,dic = subtables(data,col,delete=False) |
|  |  |
|  | total\_size=len(data) |
|  | entropies=[0]\*len(attr) |
|  | ratio=[0]\*len(attr) |
|  |  |
|  | total\_entropy=entropy([row[-1] for row in data]) |
|  | for x in range(len(attr)): |
|  | ratio[x]=len(dic[attr[x]])/(total\_size\*1.0) |
|  | entropies[x]=entropy([row[-1] for row in dic[attr[x]]]) |
|  | total\_entropy-=ratio[x]\*entropies[x] |
|  | return total\_entropy |
|  |  |
|  | def build\_tree(data,features): |
|  | lastcol=[row[-1] for row in data] |
|  | if(len(set(lastcol)))==1: |
|  | node=Node("") |
|  | node.answer=lastcol[0] |
|  | return node |
|  |  |
|  | n=len(data[0])-1 |
|  | gains=[0]\*n |
|  | for col in range(n): |
|  | gains[col]=compute\_gain(data,col) |
|  | split=gains.index(max(gains)) |
|  | node=Node(features[split]) |
|  | fea = features[:split]+features[split+1:] |
|  |  |
|  |  |
|  | attr,dic=subtables(data,split,delete=True) |
|  |  |
|  | for x in range(len(attr)): |
|  | child=build\_tree(dic[attr[x]],fea) |
|  | node.children.append((attr[x],child)) |
|  | return node |
|  |  |
|  | def print\_tree(node,level): |
|  | if node.answer!="": |
|  | print(" "\*level,node.answer) |
|  | return |
|  |  |
|  | print(" "\*level,node.attribute) |
|  | for value,n in node.children: |
|  | print(" "\*(level+1),value) |
|  | print\_tree(n,level+2) |
|  |  |
|  |  |
|  | def classify(node,x\_test,features): |
|  | if node.answer!="": |
|  | print(node.answer) |
|  | return |
|  | pos=features.index(node.attribute) |
|  | for value, n in node.children: |
|  | if x\_test[pos]==value: |
|  | classify(n,x\_test,features) |
|  |  |
|  | '''Main program''' |
|  | dataset,features=load\_csv("id3.csv") |
|  | node1=build\_tree(dataset,features) |
|  |  |
|  | print("The decision tree for the dataset using ID3 algorithm is") |
|  | print\_tree(node1,0) |
|  | testdata,features=load\_csv("test.csv") |
|  |  |
|  | for xtest in testdata: |
|  | print("The test instance:",xtest) |
|  | print("The label for test instance:") |
|  | classify(node1,xtest,features) |

****

****

4.Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

|  |
| --- |
| import numpy as np |
|  | import pandas as pd |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn.naive\_bayes import GaussianNB |
|  | from sklearn import metrics |
|  |  |
|  | df = pd.read\_csv("pima\_indian.csv") |
|  | feature\_col\_names = ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness', 'insulin', 'bmi', 'diab\_pred', 'age'] |
|  | predicted\_class\_names = ['diabetes'] |
|  | X = df[feature\_col\_names].values |
|  | y = df[predicted\_class\_names].values |
|  | print(df.head) |
|  | xtrain,xtest,ytrain,ytest=train\_test\_split(X,y,test\_size=0.33) |
|  | print ('\nThe total number of Training Data:',ytrain.shape) |
|  | print ('The total number of Test Data:',ytest.shape) |
|  | clf = GaussianNB().fit(xtrain,ytrain.ravel()) |
|  | predicted = clf.predict(xtest) |
|  | predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]]) |
|  | print('\nConfusion matrix') |
|  | print(metrics.confusion\_matrix(ytest,predicted)) |
|  | print('\nAccuracy of the classifier:',metrics.accuracy\_score(ytest,predicted)) |
|  | print('The value of Precision:', metrics.precision\_score(ytest,predicted)) |
|  | print('The value of Recall:', metrics.recall\_score(ytest,predicted)) |
|  | print("Predicted Value for individual Test Data:", predictTestData) |



5.Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

|  |
| --- |
| import numpy as np |
|  | import matplotlib.pyplot as plt |
|  | import pandas as pd |
|  | dataset = pd.read\_csv('salary.csv') |
|  | X = dataset.iloc[:, :-1].values |
|  | y = dataset.iloc[:, 1].values |
|  | from sklearn.model\_selection import train\_test\_split |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0) |
|  | # Fitting Simple Linear Regression to the Training set |
|  | from sklearn.linear\_model import LinearRegression |
|  | regressor = LinearRegression() |
|  | regressor.fit(X\_train, y\_train) |
|  | LinearRegression() |
|  | # Predicting the Test set results |
|  | y\_pred = regressor.predict(X\_test) |
|  | # Visualizing the Training set results |
|  | viz\_train = plt |
|  | viz\_train.scatter(X\_train, y\_train, color='red') |
|  | viz\_train.plot(X\_train, regressor.predict(X\_train), color='blue') |
|  | viz\_train.title('Salary VS Experience (Training set)') |
|  | viz\_train.xlabel('Year of Experience') |
|  | viz\_train.ylabel('Salary') |
|  | viz\_train.show() |
|  |  |
|  | # Visualizing the Test set results |
|  | viz\_test = plt |
|  | viz\_test.scatter(X\_test, y\_test, color='red') |
|  | viz\_test.plot(X\_train, regressor.predict(X\_train), color='blue') |
|  | viz\_test.title('Salary VS Experience (Test set)') |
|  | viz\_test.xlabel('Year of Experience') |
|  | viz\_test.ylabel('Salary') |
|  | viz\_test.show() |

