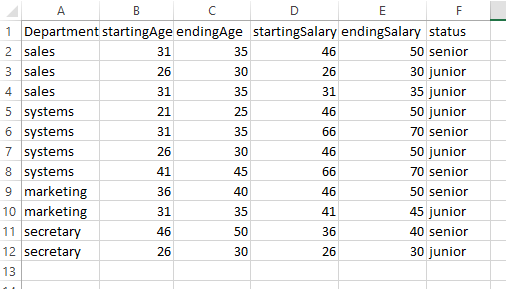
**Dataset (.csv file):**



**Explanation:**

* The first step is to **convert the .csv file** which contains the dataset into a data structure which in my case is a **List.** The **load\_file( )** function performs this task
* The next step is to **covert the string values in the dataset** to integer values and float values to make it easy for calculation. Two columns were string: Department and Status. For this purpose the functions: **column\_str\_to\_int( )** and **str\_to\_int( )** functions are used. 4 numeric columns were converted to float values using function **column\_str\_to\_float( )**
* Next, the data is split according to classes. Status was the column which was used as a class. A dictionary is made which stores the rows in the form of a list according to class. First I access the status column of each row in dataset. Then check if the class is already in the dictionary. The row is appended with its class in the dictionary in the form of a list. The function **split\_data\_by\_class( )** is used for this purpose.
* Since the data was in numerical form, Gaussian distribution was used. For this purpose, mean and standard deviation was calculated for each column in each class and stored in a dictionary for later use. **Mean( )** function and **standard\_deviation( )** function is used to calculate mean and standard deviation. The function **data\_calculation( )** assembles mean, standard deviation and length for all columns and stores in dictionary and function **class\_calculation( )** calls data\_calculation( ) for each class. This step provides a complete statistical summary
* The next function is the **class\_probability( )** which calls **gaussian\_probability( )** for each class and calculate probability for finding the status of the new testing data. The formula used to calculate probability is:

**P(senior| [marketing, startingAge, endingAge, startingSalary, endingSalary**]) = **P(Department|senior) \* P(startingAge|senior) \* P(endingAge| senior) \* P(startingSalary| senior) \* P(endingSalary| senior) \*P(senior)**

**And**

**P(Department|senior) =**

**Code:**

|  |
| --- |
| from math import sqrt  from math import exp  from math import pi  from csv import reader  #loading file containing dataset  def load\_file(filename):  dataset=list() #for storing rows in dataset  with open(filename,'r') as file:  data=reader(file)  for row in data:  if not row:  continue  dataset.append(row) #converting dataset in file to list of ordered collectiom  return dataset  #converting string-to-float  def column\_str\_to\_float(dataset,column):  for row in dataset:  row[column]=float(row[column].strip())    #converting class column to int Value  def column\_str\_to\_int(dataset,column):  class\_values=[row[column] for row in dataset]  unique=set(class\_values)  lookup = dict()  #making a mapping function for classes  print('CLASSES:')  for i, value in enumerate(unique):  lookup[value] = i  print('%s => %d' % (value, i))  print('----------------------------')  #changing string class value to integer value  for row in dataset:  row[column] = lookup[row[column]]  return lookup  def str\_to\_int(dataset,column):  class\_values=[row[column] for row in dataset]  unique=set(class\_values)  lookup = dict()  #making a mapping function for classes  print('DEPARTMENTS:')  for i, value in enumerate(unique):  lookup[value] = i  print('%s => %d' % (value, i))  print('----------------------------')  #changing string class value to integer value  for row in dataset:  row[column] = lookup[row[column]]  return lookup    #splitting data set by class  def split\_data\_by\_class(dataset):  separated = dict()  for i in range(len(dataset)):  vector = dataset[i]  class\_value = vector[-1]  #making each record of seperated dictionary a list  if (class\_value not in separated):  separated[class\_value] = list()  separated[class\_value].append(vector)#appending row with its class list  return separated  #calculating mean for dataset\  def mean(values):  avg=sum(values)/float(len(values))  return avg  #calculating standard deviation  def standard\_deviation(values):  avg=mean(values)  variance=sum([(x-avg)\*\*2 for x in values])/float(len(values)-1)  stdev=sqrt(variance)  return stdev  def data\_calculation(dataset):  summaries = [(mean(column), standard\_deviation(column), len(column)) for column in zip(\*dataset)]  del(summaries[-1])  return summaries  #splitting data set by classes and performing calculations  def class\_calculation(dataset):  #1. Splitting the data by the classes which were converted to integer values  #2. Calculating mean and standard deviation for each column of each class\_calculation  #3. storing values of each row in a dictionary summary according to each class  split=split\_data\_by\_class(dataset)  summary=dict()  for value,rows in split.items():  summary[value]=data\_calculation(rows)  return summary  def gaussian\_probability(x,mean,std):  e=exp(-((x-mean)\*\*2/(2\*std\*\*2)))  d=(1/(sqrt(2\*pi)\*std))  ans=d\*e  return ans  def class\_probability(summaries,newrow):  #getting total rows of the records for calculating probability of class  #classvalues=names of classes in integer  #classsummaries= summary of statistics for each class  totalrows=sum([summaries[label][0][2] for label in summaries])  total\_probability=dict()  for class\_value,class\_summaries in summaries.items():  total\_probability[class\_value]=summaries[class\_value][0][2]/float(totalrows)  for i in range(len(class\_summaries)):  mean,stdev,\_=class\_summaries[i]  total\_probability[class\_value]\*=gaussian\_probability(newrow[i],mean,stdev)  return total\_probability    def findclass(summaries, row):  probabilities = class\_probability(summaries, row)    best\_label=None  best\_prob=-1  for class\_value, probability in probabilities.items():  #finding class with highest probability  if best\_label is None or probability > best\_prob:  best\_prob = probability  best\_label = class\_value    return best\_label  filename='mydataset.csv'  #reading file  dataset=load\_file(filename)  #converting string column values to floating values  for i in range (1,len(dataset[0])-1):#till columns length  column\_str\_to\_float(dataset,i)  #converting class column to int Value  department=dict()  classes=column\_str\_to\_int(dataset, len(dataset[0])-1)  department=str\_to\_int(dataset,0)  #fitting Model  summary=class\_calculation(dataset)  row=['sales',31,35,66,70]  s=row[0]  for x in department:  if x==s:  row[0]=department[x]  # predict the label  predicted\_status = findclass(summary, row)  print("RESULTS:")  for y in classes:  if classes[y]==predicted\_status:  myclass=y  print('Data=%s, Predicted: %s' % (row, myclass)) |

**Output:**

|  |
| --- |
| **Testing data:**  [marketing, 31, 35, 46, 50] |

|  |
| --- |
| **Testing data:**  [sales, 31, 35, 66, 70] |