Assignment 4

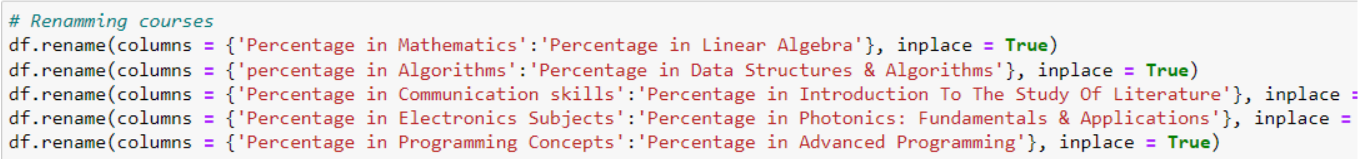
Job Role Suggestion System using ANN

Preprocessing the Data and Experimenting with the values:

As mentioned, I used different modification techniques here I pasted the best ones with the reason why I choose them.

1. Renaming Columns:

As mentioned in point (g) of the handout, first renamed some of the features (Courses/electives) according to the Courses/electives used in assignment 1.

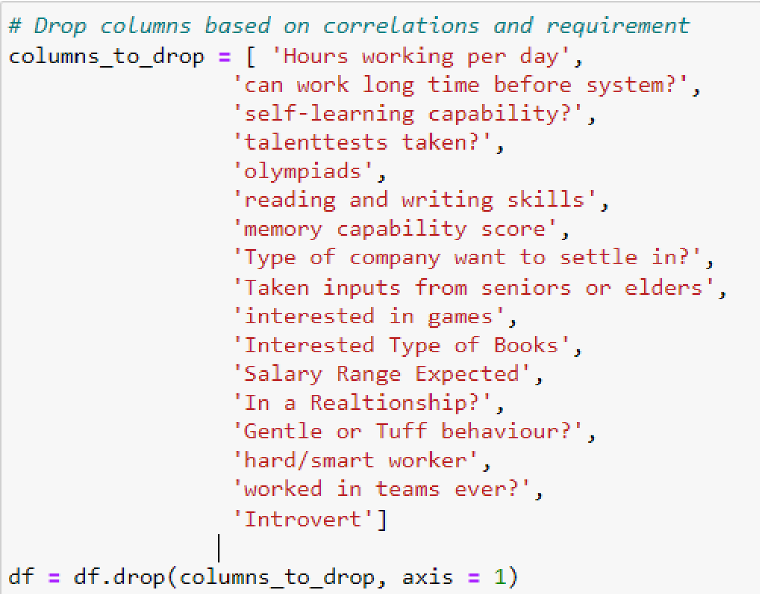


```
# Renamming courses
df.rename(columns = {'Percentage in Mathematics':'Percentage in Linear Algebra'}, inplace = True)
df.rename(columns = {'percentage in Algorithms':'Percentage in Data Structures & Algorithms'}, inplace = True)
df.rename(columns = {'Percentage in Communication skills':'Percentage in Introduction To The Study Of Literature'}, inplace =
df.rename(columns = {'Percentage in Electronics Subjects':'Percentage in Photonics: Fundamentals & Applications'}, inplace =
df.rename(columns = {'Percentage in Programming Concepts':'Percentage in Advanced Programming'}, inplace = True)
```

2. Dropping Features:

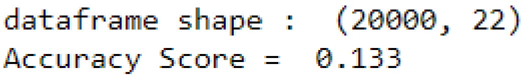
Then, some of the features from the renamed dataset were dropped based on their correlations and the assignment requirement.

- **Reason:** Features were dropped to reduce the computational and spatial cost of the model.

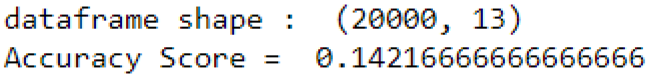


```
# Drop columns based on correlations and requirement
columns_to_drop = [ 'Hours working per day',
                  'can work long time before system?',
                  'self-learning capability?',
                  'talenttests taken?',
                  'olympiads',
                  'reading and writing skills',
                  'memory capability score',
                  'Type of company want to settle in?',
                  'Taken inputs from seniors or elders',
                  'interested in games',
                  'Interested Type of Books',
                  'Salary Range Expected',
                  'In a Realtionship?',
                  'Gentle or Tuff behaviour?',
                  'hard/smart worker',
                  'worked in teams ever?',
                  'Introvert']

df = df.drop(columns_to_drop, axis = 1)
```

- **Experimenting with values:** To find the best columns, I tried different combinations of columns by further removing some of the columns and then ran an MLP classifier on those datasets. Here are two of the results that I got.



```
dataframe shape :  (20000, 22)
Accuracy Score =  0.133
```



```
dataframe shape :  (20000, 13)
Accuracy Score =  0.14216666666666666
```

There was approximately a 1% difference between the accuracy score of the two datasets, not much difference between the Accuracy Scores, so I proceeded with 22 feature spaces.

In NNs, weights help in feature extraction by reducing the weight of respective columns to zero.

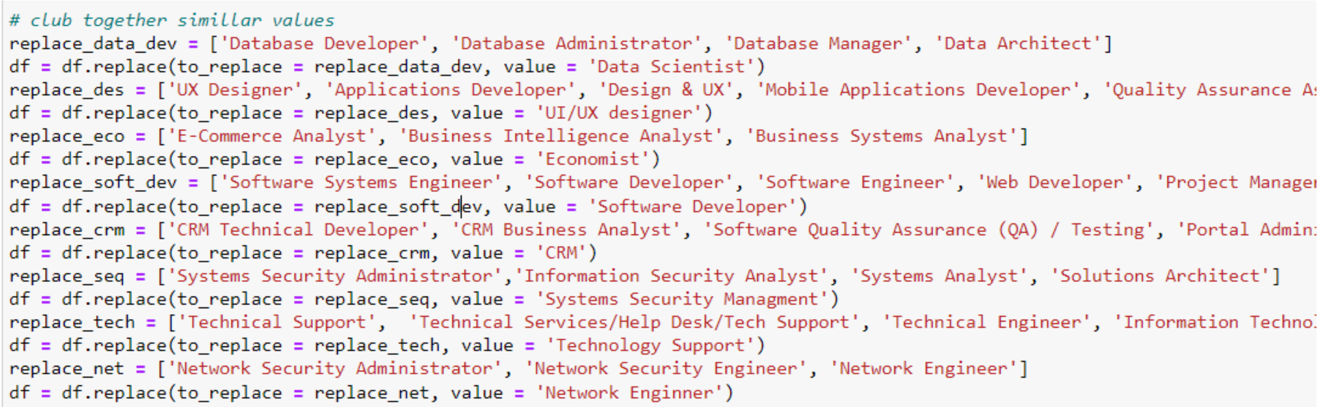
3. Clubbing values:

1. After dropping the features, the size of the dataset was reduced to (20000, 22).

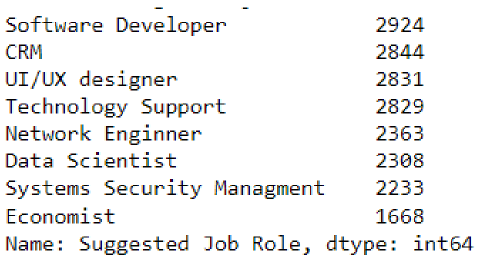
Updated the feature, ' Suggested Job Role,' in the new dataset to club together values using the similar nature of the job. The resultant gives an eight-class classification with Unique values:

['Data Scientist', 'CRM', 'Systems Security Management', 'Economist', 'Software Developer', 'UI/UX designer', 'Technology Support', 'Network Engineer']

- **Reason:** Clubbed together the values based on Job nature as in the original dataset, there were 34 unique Suggested Job Roles. This many classes for a small dataset of 20,000 would have given a bad accuracy score.



```
# club together simillar values
replace_data_dev = ['Database Developer', 'Database Administrator', 'Database Manager', 'Data Architect']
df = df.replace(to_replace = replace_data_dev, value = 'Data Scientist')
replace_des = ['UX Designer', 'Applications Developer', 'Design & UX', 'Mobile Applications Developer', 'Quality Assurance A
df = df.replace(to_replace = replace_des, value = 'UI/UX designer')
replace_eco = ['E-Commerce Analyst', 'Business Intelligence Analyst', 'Business Systems Analyst']
df = df.replace(to_replace = replace_eco, value = 'Economist')
replace_soft_dev = ['Software Systems Engineer', 'Software Developer', 'Software Engineer', 'Web Developer', 'Project Manage
df = df.replace(to_replace = replace_soft_dev, value = 'Software Developer')
replace_crm = ['CRM Technical Developer', 'CRM Business Analyst', 'Software Quality Assurance (QA) / Testing', 'Portal Admin
df = df.replace(to_replace = replace_crm, value = 'CRM')
replace_seq = ['Systems Security Administrator','Information Security Analyst', 'Systems Analyst', 'Solutions Architect']
df = df.replace(to_replace = replace_seq, value = 'Systems Security Managment')
replace_tech = ['Technical Support',  'Technical Services/Help Desk/Tech Support', 'Technical Engineer', 'Information Techno
df = df.replace(to_replace = replace_tech, value = 'Technology Support')
replace_net = ['Network Security Administrator', 'Network Security Engineer', 'Network Engineer']
df = df.replace(to_replace = replace_net, value = 'Network Enginner')
```



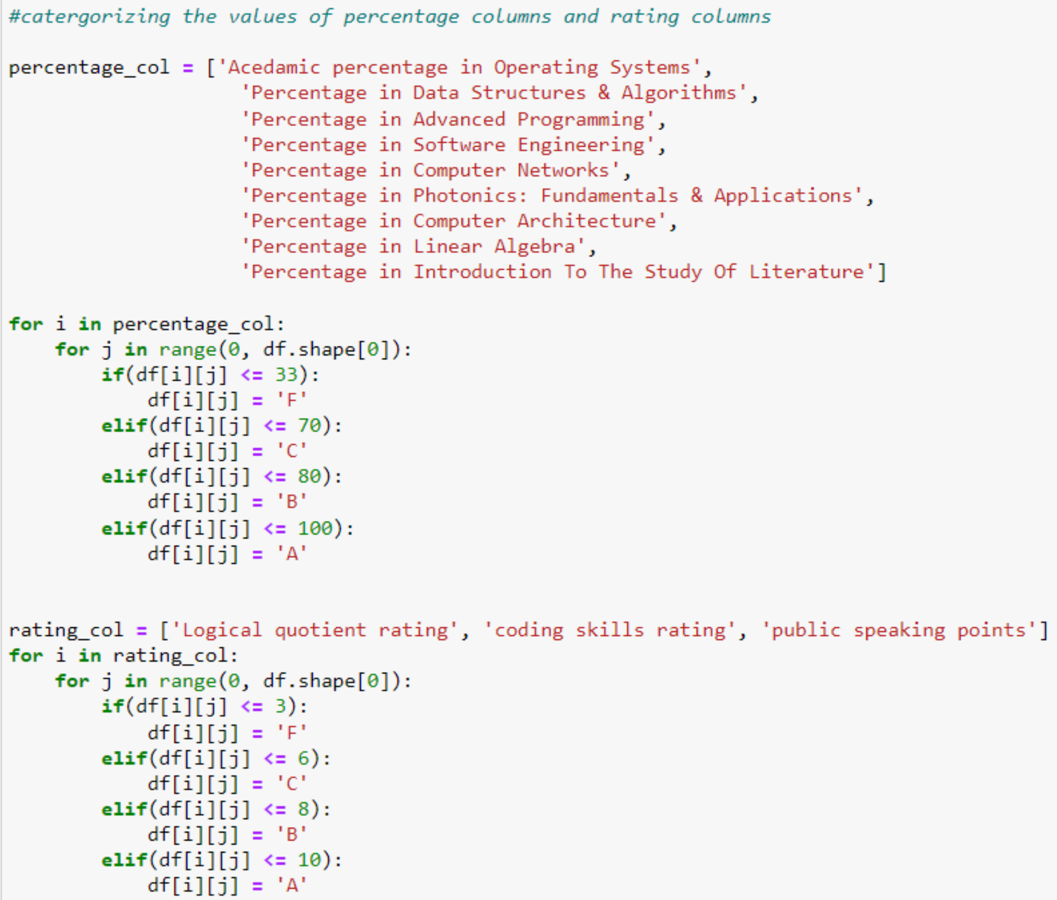
```
Software Developer            2924
CRM                           2844
UI/UX designer                2831
Technology Support            2829
Network Enginner              2363
Data Scientist                2308
Systems Security Managment    2233
Economist                     1668
Name: Suggested Job Role, dtype: int64
```

Unique value count in the 'Suggested Job Role' Column

4. Bucketing values:

Using a custom function, bucketed the value in all the percentage and ranking features to four classes ‘A’, ‘B’, ‘C’ and ‘F’.

- Used bucketing to improve the final model accuracy.

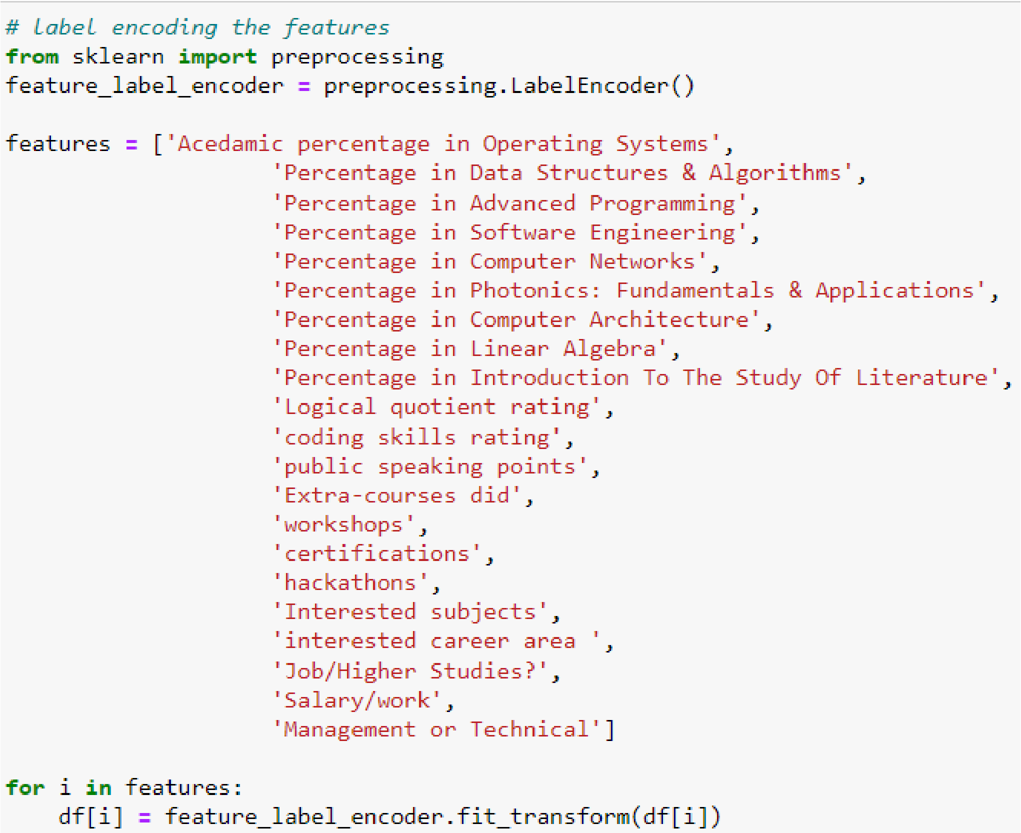


```
#catergorizing the values of percentage columns and rating columns

percentage_col = ['Acedamic percentage in Operating Systems',
                  'Percentage in Data Structures & Algorithms',
                  'Percentage in Advanced Programming',
                  'Percentage in Software Engineering',
                  'Percentage in Computer Networks',
                  'Percentage in Photonics: Fundamentals & Applications',
                  'Percentage in Computer Architecture',
                  'Percentage in Linear Algebra',
                  'Percentage in Introduction To The Study Of Literature']

for i in percentage_col:
    for j in range(0, df.shape[0]):
        if(df[i][j] <= 33):
            df[i][j] = 'F'
        elif(df[i][j] <= 70):
            df[i][j] = 'C'
        elif(df[i][j] <= 80):
            df[i][j] = 'B'
        elif(df[i][j] <= 100):
            df[i][j] = 'A'


rating_col = ['Logical quotient rating', 'coding skills rating', 'public speaking points']
for i in rating_col:
    for j in range(0, df.shape[0]):
        if(df[i][j] <= 3):
            df[i][j] = 'F'
        elif(df[i][j] <= 6):
            df[i][j] = 'C'
        elif(df[i][j] <= 8):
            df[i][j] = 'B'
        elif(df[i][j] <= 10):
            df[i][j] = 'A'
```

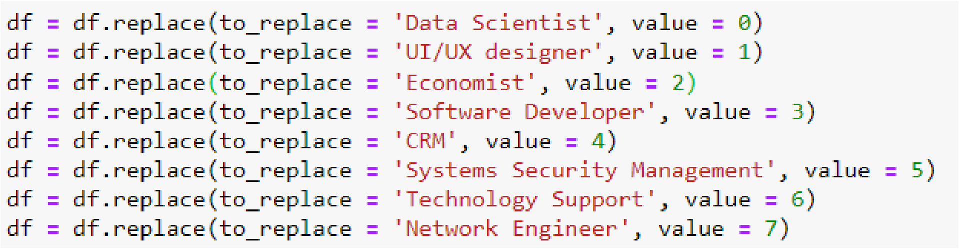
5. Label Encoding:

After that, label encoded all the columns with a string value or an int value.

- Here, I used label encoding instead of one hot encoding as one hot encoding will increase the features count, and that will cost more computational power.
- Also, the difference between label-encoded dataset accuracy and one hot-encoded dataset accuracy was marginal.
- Giving more priority to computational power and choosing to proceed with the label-encoded dataset



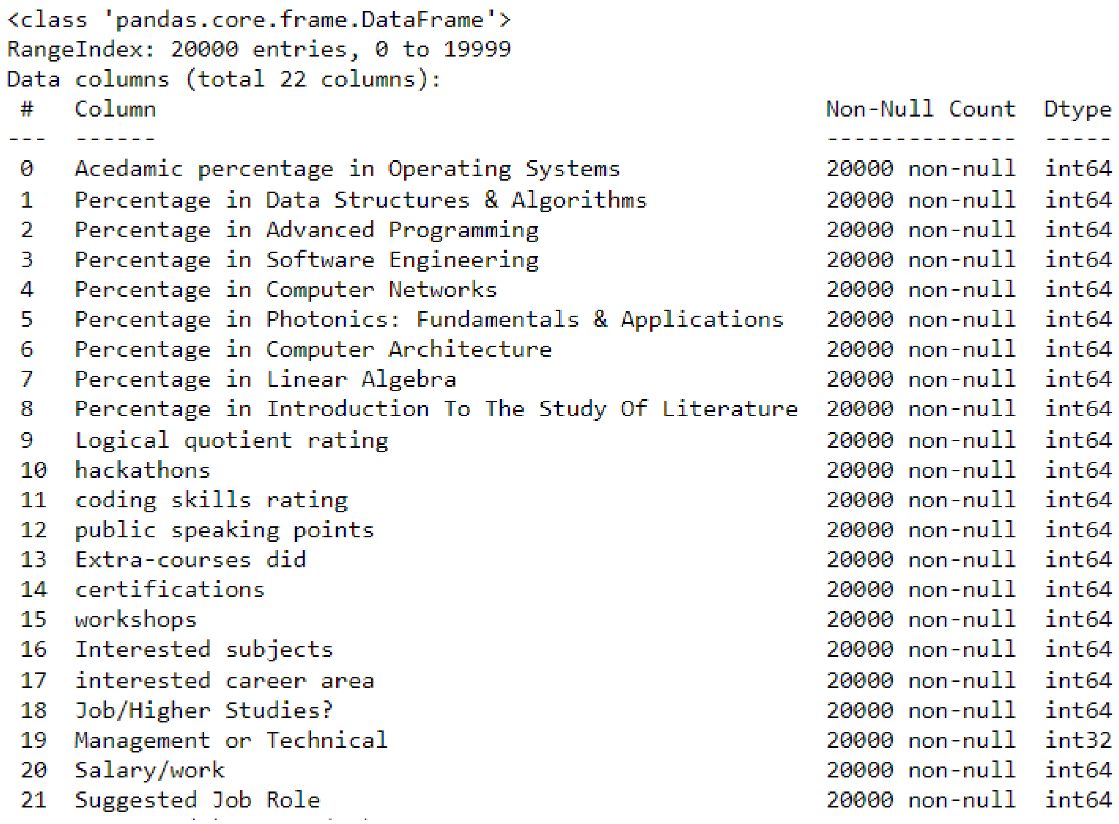
```
# label encoding the features
from sklearn import preprocessing
feature_label_encoder = preprocessing.LabelEncoder()

features = ['Acedamic percentage in Operating Systems',
                    'Percentage in Data Structures & Algorithms',
                    'Percentage in Advanced Programming',
                    'Percentage in Software Engineering',
                    'Percentage in Computer Networks',
                    'Percentage in Photonics: Fundamentals & Applications',
                    'Percentage in Computer Architecture',
                    'Percentage in Linear Algebra',
                    'Percentage in Introduction To The Study Of Literature',
                    'Logical quotient rating',
                    'coding skills rating',
                    'public speaking points',
                    'Extra-courses did',
                    'workshops',
                    'certifications',
                    'hackathons',
                    'Interested subjects',
                    'interested career area ',
                    'Job/Higher Studies?',
                    'Salary/work',
                    'Management or Technical']

for i in features:
    df[i] = feature_label_encoder.fit_transform(df[i])
```



```
df = df.replace(to_replace = 'Data Scientist', value = 0)
df = df.replace(to_replace = 'UI/UX designer', value = 1)
df = df.replace(to_replace = 'Economist', value = 2)
df = df.replace(to_replace = 'Software Developer', value = 3)
df = df.replace(to_replace = 'CRM', value = 4)
df = df.replace(to_replace = 'Systems Security Management', value = 5)
df = df.replace(to_replace = 'Technology Support', value = 6)
df = df.replace(to_replace = 'Network Engineer', value = 7)
```

classes and their labels

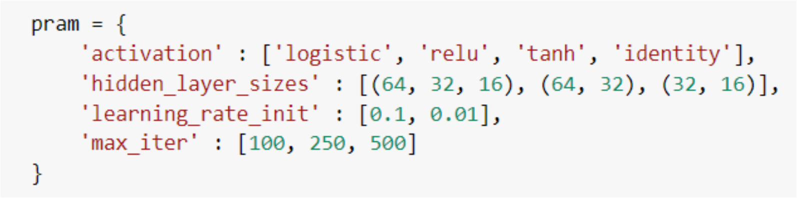
Screenshot of the final dataset:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 22 columns):
 #   Column                                         Non-Null Count  Dtype
---  ------                                         --------------  -----
 0   Acedamic percentage in Operating Systems       20000 non-null  int64
 1   Percentage in Data Structures & Algorithms     20000 non-null  int64
 2   Percentage in Advanced Programming             20000 non-null  int64
 3   Percentage in Software Engineering             20000 non-null  int64
 4   Percentage in Computer Networks                20000 non-null  int64
 5   Percentage in Photonics: Fundamentals & Applications 20000 non-null  int64
 6   Percentage in Computer Architecture            20000 non-null  int64
 7   Percentage in Linear Algebra                   20000 non-null  int64
 8   Percentage in Introduction To The Study Of Literature 20000 non-null  int64
 9   Logical quotient rating                        20000 non-null  int64
 10  hackathons                                     20000 non-null  int64
 11  coding skills rating                           20000 non-null  int64
 12  public speaking points                         20000 non-null  int64
 13  Extra-courses did                              20000 non-null  int64
 14  certifications                                 20000 non-null  int64
 15  workshops                                      20000 non-null  int64
 16  Interested subjects                            20000 non-null  int64
 17  interested career area                         20000 non-null  int64
 18  Job/Higher Studies?                            20000 non-null  int64
 19  Management or Technical                        20000 non-null  int32
 20  Salary/work                                    20000 non-null  int64
 21  Suggested Job Role                             20000 non-null  int64
```

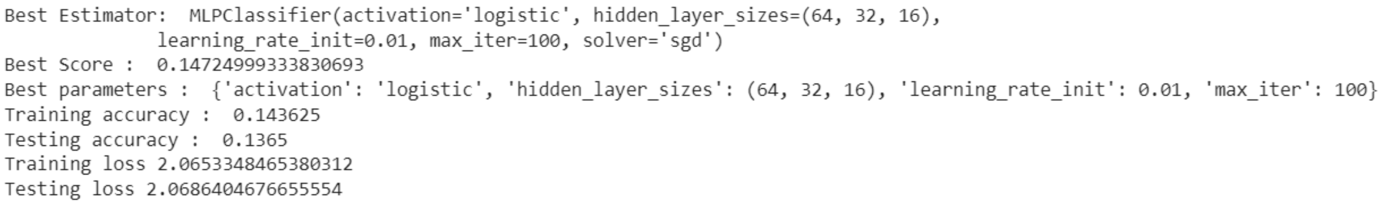
6. Experimenting with values:

For point (f), I ran the grid search algorithm to find the best combination of the parameters for the MLP classifier.



```
pram = {
    'activation' : ['logistic', 'relu', 'tanh', 'identity'],
    'hidden_layer_sizes' : [(64, 32, 16), (64, 32), (32, 16)],
    'learning_rate_init' : [0.1, 0.01],
    'max_iter' : [100, 250, 500]
}
```

Result of grid search:

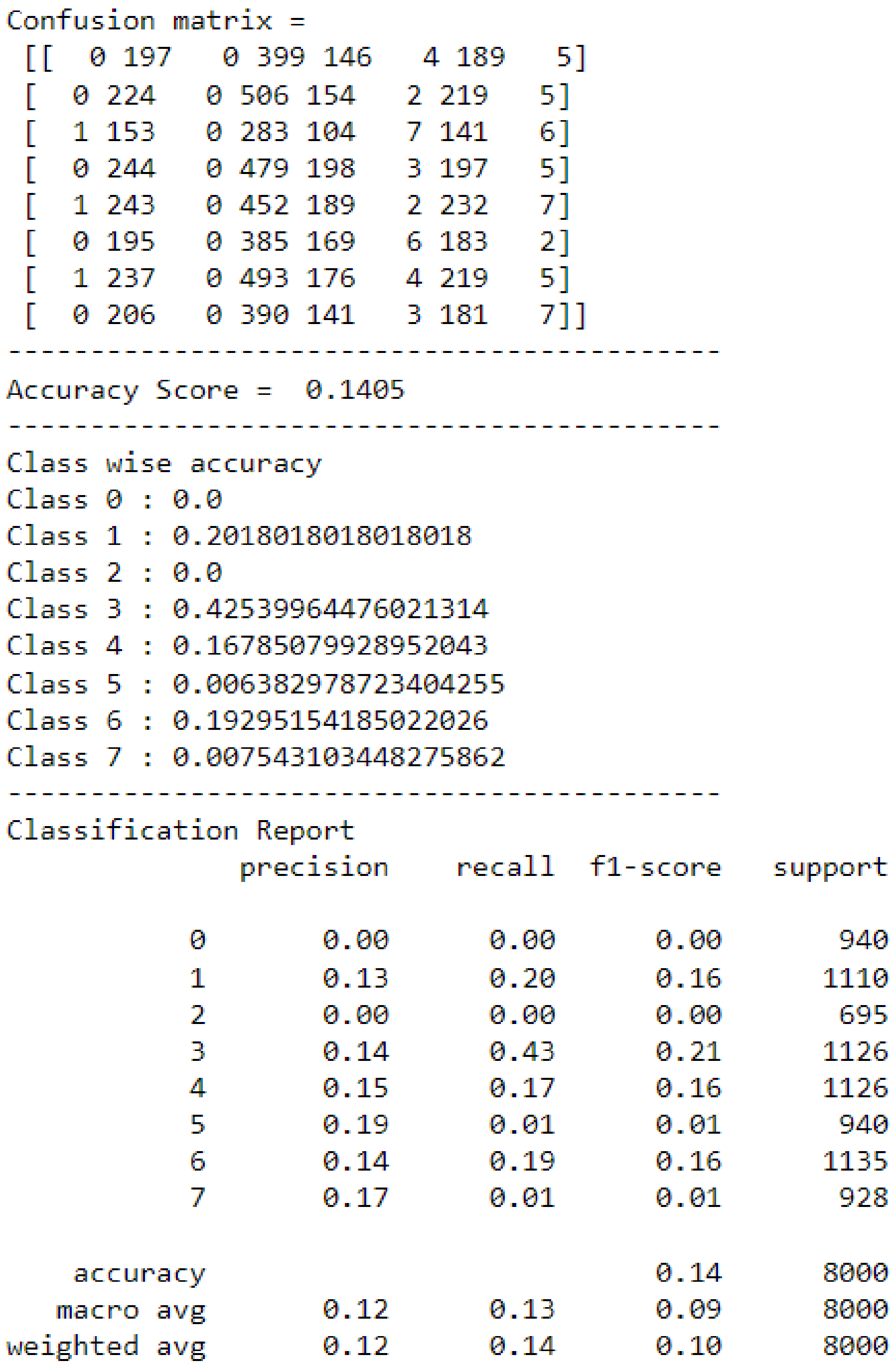


```
Best Estimator:  MLPClassifier(activation='logistic', hidden_layer_sizes=(64, 32, 16),
              learning_rate_init=0.01, max_iter=100, solver='sgd')
Best Score :  0.14724999333830693
Best parameters :  {'activation': 'logistic', 'hidden_layer_sizes': (64, 32, 16), 'learning_rate_init': 0.01, 'max_iter': 100}
Training accuracy :  0.143625
Testing accuracy :  0.1365
Training loss 2.0653348465380312
Testing loss 2.0686404676655554
```

Activation Function: ‘logistic’
Hidden layer sizes = (64, 32, 16)
learning rate = 0.01
Max Iterations = 100

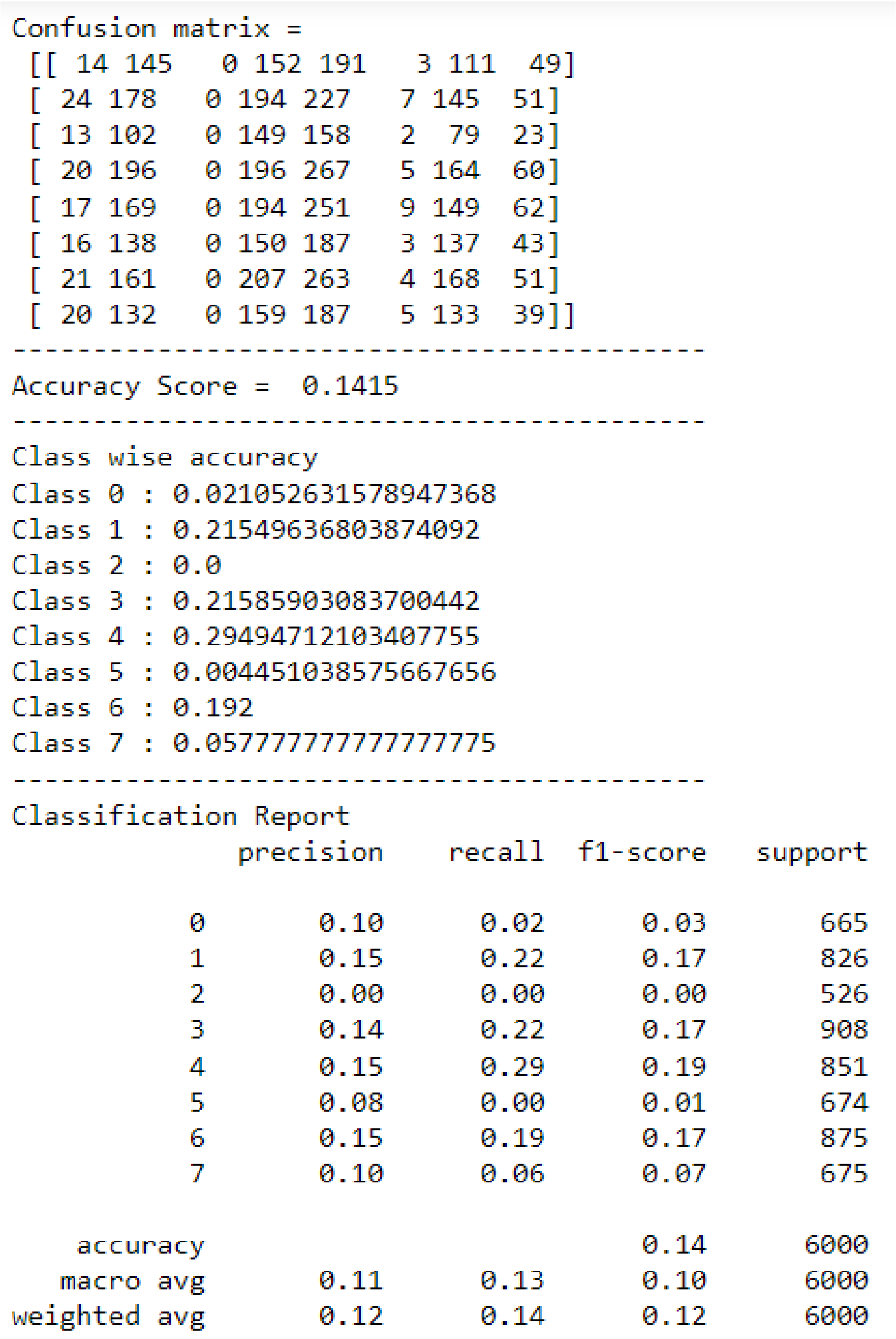
Results

60:40 split



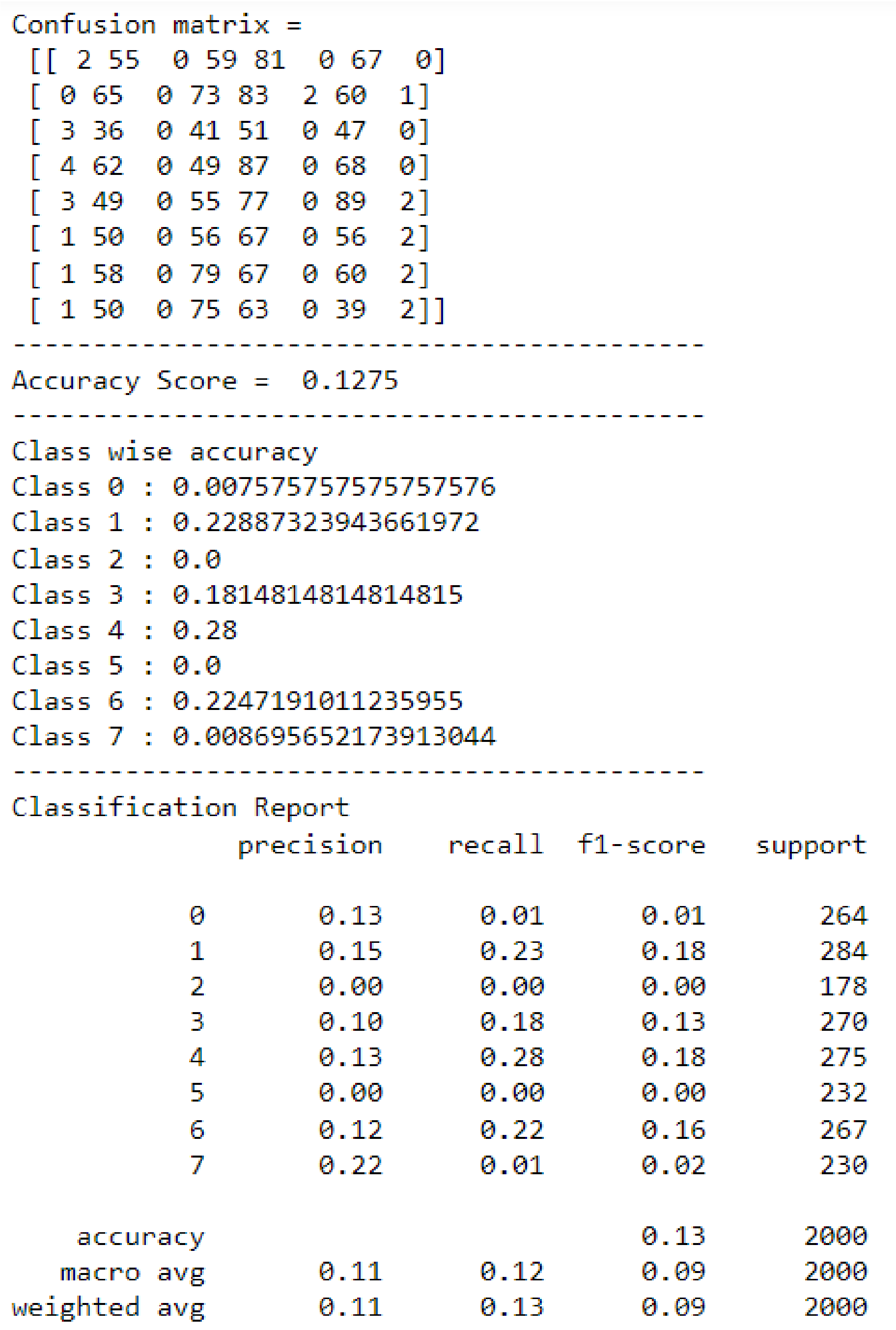
```
Confusion matrix =
 [[  0 197   0 399 146   4 189   5]
 [  0 224   0 506 154   2 219   5]
 [  1 153   0 283 104   7 141   6]
 [  0 244   0 479 198   3 197   5]
 [  1 243   0 452 189   2 232   7]
 [  0 195   0 385 169   6 183   2]
 [  1 237   0 493 176   4 219   5]
 [  0 206   0 390 141   3 181   7]]
-------------------------------------------
Accuracy Score =  0.1405
-------------------------------------------
Class wise accuracy
Class 0 : 0.0
Class 1 : 0.2018018018018018
Class 2 : 0.0
Class 3 : 0.42539964476021314
Class 4 : 0.16785079928952043
Class 5 : 0.006382978723404255
Class 6 : 0.19295154185022026
Class 7 : 0.007543103448275862
-------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       940
           1       0.13      0.20      0.16      1110
           2       0.00      0.00      0.00       695
           3       0.14      0.43      0.21      1126
           4       0.15      0.17      0.16      1126
           5       0.19      0.01      0.01       940
           6       0.14      0.19      0.16      1135
           7       0.17      0.01      0.01       928

    accuracy                           0.14      8000
   macro avg       0.12      0.13      0.09      8000
weighted avg       0.12      0.14      0.10      8000
```

70:30 split



```
Confusion matrix =
 [[ 14 145   0 152 191   3 111  49]
 [ 24 178   0 194 227   7 145  51]
 [ 13 102   0 149 158   2  79  23]
 [ 20 196   0 196 267   5 164  60]
 [ 17 169   0 194 251   9 149  62]
 [ 16 138   0 150 187   3 137  43]
 [ 21 161   0 207 263   4 168  51]
 [ 20 132   0 159 187   5 133  39]]
-------------------------------------------
Accuracy Score =  0.1415
-------------------------------------------
Class wise accuracy
Class 0 : 0.021052631578947368
Class 1 : 0.21549636803874092
Class 2 : 0.0
Class 3 : 0.21585903083700442
Class 4 : 0.29494712103407755
Class 5 : 0.004451038575667656
Class 6 : 0.192
Class 7 : 0.057777777777777775
-------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.10      0.02      0.03       665
           1       0.15      0.22      0.17       826
           2       0.00      0.00      0.00       526
           3       0.14      0.22      0.17       908
           4       0.15      0.29      0.19       851
           5       0.08      0.00      0.01       674
           6       0.15      0.19      0.17       875
           7       0.10      0.06      0.07       675

    accuracy                           0.14      6000
   macro avg       0.11      0.13      0.10      6000
weighted avg       0.12      0.14      0.12      6000
```

90:10 split



```
Confusion matrix =
 [[ 2 55  0 59 81  0 67  0]
 [ 0 65  0 73 83  2 60  1]
 [ 3 36  0 41 51  0 47  0]
 [ 4 62  0 49 87  0 68  0]
 [ 3 49  0 55 77  0 89  2]
 [ 1 50  0 56 67  0 56  2]
 [ 1 58  0 79 67  0 60  2]
 [ 1 50  0 75 63  0 39  2]]
-------------------------------------------
Accuracy Score =  0.1275
-------------------------------------------
Class wise accuracy
Class 0 : 0.007575757575757576
Class 1 : 0.22887323943661972
Class 2 : 0.0
Class 3 : 0.18148148148148815
Class 4 : 0.28
Class 5 : 0.0
Class 6 : 0.2247191011235955
Class 7 : 0.008695652173913044
-------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.13      0.01      0.01       264
           1       0.15      0.23      0.18       284
           2       0.00      0.00      0.00       178
           3       0.10      0.18      0.13       270
           4       0.13      0.28      0.18       275
           5       0.00      0.00      0.00       232
           6       0.12      0.22      0.16       267
           7       0.22      0.01      0.02       230

    accuracy                           0.13      2000
   macro avg       0.11      0.12      0.09      2000
weighted avg       0.11      0.13      0.09      2000
```

Analysis of the obtained results:

Final model parameters

Split: 70:30
Activation Function: ‘logistic’
Hidden layer sizes : (64, 32, 16)
learning rate: 0.01
Max Iterations: 100

Comparing Accuracies:

Training Testing Split	Accuracy Score	Best Accuracy Class (Accuracy)	Worst Accuracy Class (Accuracy)
60:40	0.1405	class 3 (0.42)	class 0 and 1 (0.0)
70:30	0.1415	class 4 (0.29)	class 2 (0.0)
90:10	0.1275	class 4 (0.28)	class 2 and 5 (0.0)

Best Performance on the 70:30 split with a marginal difference.

Best prediction on class 4: CRM in two out of three models.

The worst performance on class 2: Economist in all three cases.

Listing of the program

```
#!/usr/bin/env python
# coding: utf-8

# In[59]:


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt


# In[60]:


# Read Data Frame
df = pd.read_csv("roo_data.csv")


# In[61]:


# Renamming courses
df.rename(columns = {'Percentage in Mathematics':'Percentage in Linear Algebra'}, inplace = True)
df.rename(columns = {'percentage in Algorithms':'Percentage in Data Structures & Algorithms'}, inplace = True)
df.rename(columns = {'Percentage in Communication skills':'Percentage in Introduction To The Study Of Literature'},
inplace = True)
df.rename(columns = {'Percentage in Electronics Subjects':'Percentage in Photonics: Fundamentals & Applications'},
inplace = True)
df.rename(columns = {'Percentage in Programming Concepts':'Percentage in Advanced Programming'}, inplace = True)


# In[62]:


print("Info: \n")
df.info()


# In[63]:


# Print Dataframe
df.head()


# # Preprocessing the Dataframe

# In[64]:
```

```
# Plot heatmap for correlations
import seaborn as sns
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True, cmap=plt.cm.Reds)   # heatmap to show the correaltion between columns
plt.show()


# In[65]:


# list all the coulmns
list(df.columns)


# In[66]:


# Drop columns based on correlations and requirement
columns_to_drop = [ 'Hours working per day',
                'can work long time before system?',
                'self-learning capability?',
                'talenttests taken?',
                'olympiads',
                'reading and writing skills',
                'memory capability score',
                'Type of company want to settle in?',
                'Taken inputs from seniors or elders',
                'interested in games',
                'Interested Type of Books',
                'Salary Range Expected',
                'In a Realtionship?',
                'Gentle or Tuff behaviour?',
                'hard/smart worker',
                'worked in teams ever?',
                'Introvert']

df = df.drop(columns_to_drop, axis = 1)
df.shape


# In[67]:


# See the unique values in Job Role column
df['Suggested Job Role'].unique()


# In[68]:


# club together simillar values
replace_data_dev = ['Database Developer', 'Database Administrator', 'Database Manager', 'Data Architect']
df = df.replace(to_replace = replace_data_dev, value = 'Data Scientist')
replace_des = ['UX Designer', 'Applications Developer', 'Design & UX', 'Mobile Applications Developer', 'Quality Assurance
Associate']
```

```
df = df.replace(to_replace = replace_des, value = 'UI/UX designer')
replace_eco = ['E-Commerce Analyst', 'Business Intelligence Analyst', 'Business Systems Analyst']
df = df.replace(to_replace = replace_eco, value = 'Economist')
replace_soft_dev = ['Software Systems Engineer', 'Software Developer', 'Software Engineer', 'Web Developer', 'Project Manager']
df = df.replace(to_replace = replace_soft_dev, value = 'Software Developer')
replace_crm = ['CRM Technical Developer', 'CRM Business Analyst', 'Software Quality Assurance (QA) / Testing', 'Portal Administrator', 'Programmer Analyst']
df = df.replace(to_replace = replace_crm, value = 'CRM')
replace_seq = ['Systems Security Administrator','Information Security Analyst', 'Systems Analyst', 'Solutions Architect']
df = df.replace(to_replace = replace_seq, value = 'Systems Security Management')
replace_tech = ['Technical Support',  'Technical Services/Help Desk/Tech Support', 'Technical Engineer', 'Information Technology Manager', 'Information Technology Auditor']
df = df.replace(to_replace = replace_tech, value = 'Technology Support')
replace_net = ['Network Security Administrator', 'Network Security Engineer', 'Network Engineer']
df = df.replace(to_replace = replace_net, value = 'Network Engineer')


# In[69]:


print("Unique values : ", end='')
print(df['Suggested Job Role'].unique())
print(df['Suggested Job Role'].value_counts())


# In[70]:


df = df.replace(to_replace = 'Data Scientist', value = 0)
df = df.replace(to_replace = 'UI/UX designer', value = 1)
df = df.replace(to_replace = 'Economist', value = 2)
df = df.replace(to_replace = 'Software Developer', value = 3)
df = df.replace(to_replace = 'CRM', value = 4)
df = df.replace(to_replace = 'Systems Security Management', value = 5)
df = df.replace(to_replace = 'Technology Support', value = 6)
df = df.replace(to_replace = 'Network Engineer', value = 7)


# In[71]:


list(df.columns)


# In[72]:


#catergorizing the values of percentage columns and rating columns

percentage_col = ['Acedamic percentage in Operating Systems',
                  'Percentage in Data Structures & Algorithms',
                  'Percentage in Advanced Programming',
                  'Percentage in Software Engineering',
                  'Percentage in Computer Networks',
                 'Percentage in Photonics: Fundamentals & Applications',
                 'Percentage in Computer Architecture',
                 'Percentage in Linear Algebra',
                 'Percentage in Introduction To The Study Of Literature']

df1_without_class = df.copy(deep=True)

for i in percentage_col:
    for j in range(0, df.shape[0]):
        if(df[i][j] <= 33):
            df[i][j] = 'F'
        elif(df[i][j] <= 70):
            df[i][j] = 'C'
        elif(df[i][j] <= 80):
            df[i][j] = 'B'
        elif(df[i][j] <= 100):
            df[i][j] = 'A'


rating_col = ['Logical quotient rating', 'coding skills rating', 'public speaking points']
for i in rating_col:
    for j in range(0, df.shape[0]):
        if(df[i][j] <= 3):
            df[i][j] = 'F'
        elif(df[i][j] <= 6):
            df[i][j] = 'C'
        elif(df[i][j] <= 8):
            df[i][j] = 'B'
        elif(df[i][j] <= 10):
            df[i][j] = 'A'


# In[73]:


# label encoding the features
from sklearn import preprocessing
feature_label_encoder = preprocessing.LabelEncoder()

features = ['Acedamic percentage in Operating Systems',
                 'Percentage in Data Structures & Algorithms',
                 'Percentage in Advanced Programming',
                 'Percentage in Software Engineering',
                 'Percentage in Computer Networks',
                 'Percentage in Photonics: Fundamentals & Applications',
                 'Percentage in Computer Architecture',
                 'Percentage in Linear Algebra',
                 'Percentage in Introduction To The Study Of Literature',
                 'Logical quotient rating',
                 'coding skills rating',
                 'public speaking points',
                 'Extra-courses did',
                 'workshops',
                 'certifications',
                 'hackathons',
                 'Interested subjects',
                 'interested career area ',
                 'Job/Higher Studies?',
                 'Salary/work',
                 'Management or Technical']

for i in features:
    df[i] = feature_label_encoder.fit_transform(df[i])


# In[74]:


for i in list(df.columns):
    print(i)
    print(df[i].value_counts())
    print("------------------------------------------")


# In[75]:


print("Info: \n")
df.info()


# # Experimenting With the values

# Experimenting with the column values to find best combination of features -
# Result: The change in the accuracy was marginal. The NN itself choose the features by setting the values of the weights
to zero.

# In[76]:


print("dataframe shape : ",df.shape)
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.3, shuffle = True)

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 250, hidden_layer_sizes = (256, 128, 64, 32),
early_stopping = False, validation_fraction=0.001)
model.fit(train_x.values, train_y.values.ravel())
pred_y = model.predict(test_x.values)

print("Accuracy Score = ", accuracy_score(test_y.values, pred_y))


# In[ ]:


columns_to_drop = ['Logical quotient rating', 'hackathons', 'coding skills rating', 'public speaking points',
          'certifications', 'workshops', 'Job/Higher Studies?', 'Management or Technical', 'Salary/work']

df_1 = df.drop(columns_to_drop, axis = 1)
print("dataframe shape : ",df_1.shape)

from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

train_x, test_x, train_y, test_y = train_test_split(df_1.drop('Suggested Job Role', axis = 1), df_1['Suggested Job Role'],
test_size = 0.3, shuffle = True)

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 250, hidden_layer_sizes = (256, 128, 64, 32),
early_stopping = False, validation_fraction=0.001)
model.fit(train_x.values, train_y.values.ravel())
pred_y = model.predict(test_x.values)

print("Accuracy Score = ", accuracy_score(test_y.values, pred_y))


# Experimenting with the values of the columns -
# Result: The change in the accuracy was marginal. The NN itself choose the features by setting the values of the weights
to zero.

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.3, shuffle = True)

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 250, hidden_layer_sizes = (256, 128, 64, 32),
early_stopping = False, validation_fraction=0.001)
model.fit(train_x.values, train_y.values.ravel())
pred_y = model.predict(test_x.values)
print('Label Encoded')
print("Accuracy Score = ", accuracy_score(test_y.values, pred_y))


# In[ ]:


# from sklearn.model_selection import train_test_split
# from sklearn.neural_network import MLPClassifier
# from sklearn.metrics import accuracy_score

# train_x, test_x, train_y, test_y = train_test_split(df1_without_class.drop('Suggested Job Role', axis = 1),
df1_without_class['Suggested Job Role'], test_size = 0.3, shuffle = True)

# model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 250, hidden_layer_sizes = (256, 128, 64, 32),
early_stopping = False, validation_fraction=0.001)
# model.fit(train_x.values, train_y.values.ravel())
# pred_y = model.predict(test_x.values)

# print('Without Label Encoding')
# print("Accuracy Score = ", accuracy_score(test_y.values, pred_y))


# Performed Grid search to find the best combinations of the parameters

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn.model_selection import GridSearchCV

# Using 80:20 split
train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.2, shuffle = True)
model1 = MLPClassifier()


pram = {
    'activation' : ['logistic', 'relu', 'tanh', 'identity'],
    'solver' : ['sgd','adam'],
    'hidden_layer_sizes' : [(256, 128, 64, 32, 16),(256, 128, 64, 32),(200, 150, 100, 50),(500, 400, 300, 200)],
    'learning_rate_init' : [0.1, 0.01, 0.001],
    'max_iter' : [100, 250, 500, 750, 1000]
}

grid_m = GridSearchCV(model1, pram, n_jobs= -1, cv = 3)
grid_m.fit(train_x, train_y)

y_pred = grid_m.predict(test_x)

print("Best Estimator: ",grid_m.best_estimator_)
print("Best Score : ", grid_m.best_score_)
print("Best parameters : ",grid_m.best_params_)

y_pred_train = grid_m.predict(train_x)
y_pred_test = grid_m.predict(test_x)
acc_train = accuracy_score(train_y,y_pred_train)
acc_test = accuracy_score(test_y,y_pred_test)

y_prob_train = grid_m.predict_proba(train_x)
y_prob_test = grid_m.predict_proba(test_x)
loss_train = log_loss(train_y,y_prob_train)
loss_test = log_loss(test_y,y_prob_test)

print("Training accuracy : ", acc_train)
print("Testing accuracy : ", acc_test)
print("Training loss", loss_train)
print("Testing loss", loss_test)
```

```
# # 60-40 Split

# In[ ]:


# Train Test split
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.4, shuffle = True)


# In[ ]:


from sklearn.neural_network import MLPClassifier

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 100, hidden_layer_sizes = (64, 32, 16), early_stopping =
False, validation_fraction=0.001, verbose = 1)
model.fit(train_x.values, train_y.values.ravel())
pred_y = model.predict(test_x.values)


# In[ ]:


from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

confusionmatrix_60 = confusion_matrix(test_y, pred_y)
print("Confusion matrix = \n",confusionmatrix_60)
print("------------------------------------------")
print("Accuracy Score = ", accuracy_score(test_y, pred_y))
print("------------------------------------------")
print("Class wise accuracy")
temp = confusionmatrix_60.diagonal()/confusionmatrix_60.sum(axis=1)
i = 0
for j in temp:
    print("Class "+str(i)+" : "+str(j))
    i = i+1
print("------------------------------------------")
print("Classification Report")
print(classification_report(test_y, pred_y))


# # 70-30 Split

# In[ ]:


# Train Test split
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.3, shuffle = True)
```

```
# In[ ]:


from sklearn.neural_network import MLPClassifier

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 100, hidden_layer_sizes = (64, 32, 16), early_stopping =
False, validation_fraction=0.001, verbose = 1)
model.fit(train_x.values, train_y.values.ravel())
pred_y = model.predict(test_x.values)


# In[ ]:


from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

confusionmatrix_70 = confusion_matrix(test_y, pred_y)
print("Confusion matrix = \n",confusionmatrix_70)
print("------------------------------------------")
print("Accuracy Score = ", accuracy_score(test_y, pred_y))
print("------------------------------------------")
print("Class wise accuracy")
temp = confusionmatrix_70.diagonal()/confusionmatrix_70.sum(axis=1)
i = 0
for j in temp:
    print("Class "+str(i)+" : "+str(j))
    i = i+1
print("------------------------------------------")
print("Classification Report")
print(classification_report(test_y, pred_y))


# # 90-10 Split

# In[ ]:


# Train Test split
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('Suggested Job Role', axis = 1), df['Suggested Job Role'], test_size
= 0.1, shuffle = True)


# In[ ]:


from sklearn.neural_network import MLPClassifier

model = MLPClassifier(activation='tanh', solver = 'sgd',max_iter = 100, hidden_layer_sizes = (64, 32, 16), early_stopping =
False, validation_fraction=0.001, verbose = 1)
model.fit(train_x.values, train_y.values.ravel())
```

```
pred_y = model.predict(test_x.values)


# In[ ]:


from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

confusionmatrix_90 = confusion_matrix(test_y, pred_y)
print("Confusion matrix = \n",confusionmatrix_90)
print("------------------------------------------")
print("Accuracy Score = ", accuracy_score(test_y, pred_y))
print("------------------------------------------")

print("Class wise accuracy")
temp = confusionmatrix_90.diagonal()/confusionmatrix_90.sum(axis=1)
i = 0
for j in temp:
    print("Class "+str(i)+" : "+str(j))
    i = i+1
print("------------------------------------------")
print("Classification Report")
print(classification_report(test_y, pred_y))


# In[ ]:
```