This jupyter notebook is prepared by Daniel Rodriguez.

1. Load Data and perform basic EDA (4pts total)

1.1 import libraries: numpy, pandas, matplotlib.pyplot, seaborn, sklearn (1pt)

```
# TODO
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn
```

1.2 Upload the dataset to your Google Drive, then using the following code, import the data to a pandas dataframe and show the count of rows and columns (0.5pt)

```
from google.colab import drive
drive.mount('/content/drive')

file_name = '/content/drive/MyDrive/CAP_4611/Assignment2/hr_data_.csv' #you may need to change this line depending on the locatio
with open(file_name, 'r') as file:
    # TODO
    df = pd.read_csv(file_name)
    del df[df.columns[0]] #removes the first index column from the dataframe
# TODO
print('Count of rows and columns',df.shape)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
Count of rows and columns (8955, 14)
```

1.3 Show the top 7 and bottom 7 rows (0.5pt)

```
# TODO
print(df.head(7).append(df.tail(7)))
```

```
      enrollee_id      city  city_development_index  gender  \
0           29725   city_40                   0.776    Male
1             666  city_162                   0.767    Male
2             402   city_46                   0.762    Male
3           27107  city_103                   0.920    Male
4           23853  city_103                   0.920    Male
5           25619   city_61                   0.913    Male
6            6588  city_114                   0.926    Male
8948        33047  city_103                   0.920    Male
8949        13167  city_103                   0.920    Male
8950        21319   city_21                   0.624    Male
8951          251  city_103                   0.920    Male
8952        32313  city_160                   0.920  Female
8953        29754  city_103                   0.920  Female
8954        24576  city_103                   0.920    Male

          relevent_experience enrolled_university education_level  \
0      No relevent experience       no_enrollment        Graduate
1     Has relevent experience       no_enrollment         Masters
2     Has relevent experience       no_enrollment        Graduate
3     Has relevent experience       no_enrollment        Graduate
4     Has relevent experience       no_enrollment        Graduate
```

```
5     Has relevent experience       no_enrollment         Graduate
6     Has relevent experience       no_enrollment         Graduate
8948  Has relevent experience       no_enrollment         Graduate
8949  Has relevent experience       no_enrollment         Graduate
8950   No relevent experience    Full time course         Graduate
8951  Has relevent experience       no_enrollment          Masters
8952  Has relevent experience       no_enrollment         Graduate
8953  Has relevent experience       no_enrollment         Graduate
8954  Has relevent experience       no_enrollment         Graduate

     major_discipline  experience company_size    company_type last_new_job  \
0                STEM        15.0        50-99         Pvt Ltd           >4
1                STEM        21.0        50-99  Funded Startup            4
2                STEM        13.0          <10         Pvt Ltd           >4
3                STEM         7.0        50-99         Pvt Ltd            1
4                STEM         5.0    5000-9999         Pvt Ltd            1
5                STEM        21.0    1000-4999         Pvt Ltd            3
6                STEM        16.0        10/49         Pvt Ltd           >4
8948             STEM        21.0       10000+         Pvt Ltd           >4
8949             STEM         5.0      500-999         Pvt Ltd            1
8950             STEM         1.0      100-500         Pvt Ltd            1
8951             STEM         9.0        50-99         Pvt Ltd            1
8952             STEM        10.0      100-500   Public Sector            3
8953       Humanities         7.0        10/49  Funded Startup            1
8954             STEM        21.0        50-99         Pvt Ltd            4

      training_hours  target
0                 47     0.0
1                  8     0.0
2                 18     1.0
3                 46     1.0
4                108     0.0
5                 23     0.0
6                 18     0.0
8948              18     0.0
```

1.4 Show if any column has null values (0.5pt)

```
# TODO
df.isna().sum().sort_values(ascending=False)
```

```
enrollee_id               0
city                      0
city_development_index    0
gender                    0
relevent_experience       0
enrolled_university       0
education_level           0
major_discipline          0
experience                0
company_size              0
company_type              0
last_new_job              0
training_hours            0
target                    0
dtype: int64
```

1.5 Show/Plot the count of unique **target** labels and discuss its imbalances and possible issues in using it for classification. (1.5pt)

```
# TODO
table = []
for col in df:
  table.append([col,len(df[col].unique())])
table = pd.DataFrame(table)
print(table)
plt.figure(figsize = (25,10))
plt.bar(table[0],table[1])
```

```
                        0     1
0             enrollee_id  8955
1                    city   116
2  city_development_index    91
3                  gender     3
4     relevent_experience     2
5     enrolled_university     3
6         education_level     3
7        major_discipline     6
8              experience    22
9            company_size     8
10           company_type     6
11           last_new_job     6
12         training_hours   241
13                 target     2
<BarContainer object of 14 artists>
```

The main noted issue is found with the ID field as it is unique per row entry. Having a high frequency majority class will cause imbalance issues in classification that can lead to under sampling or over sampling in classification. Reducing the imbalance properly allows for the information to help better predict each class.

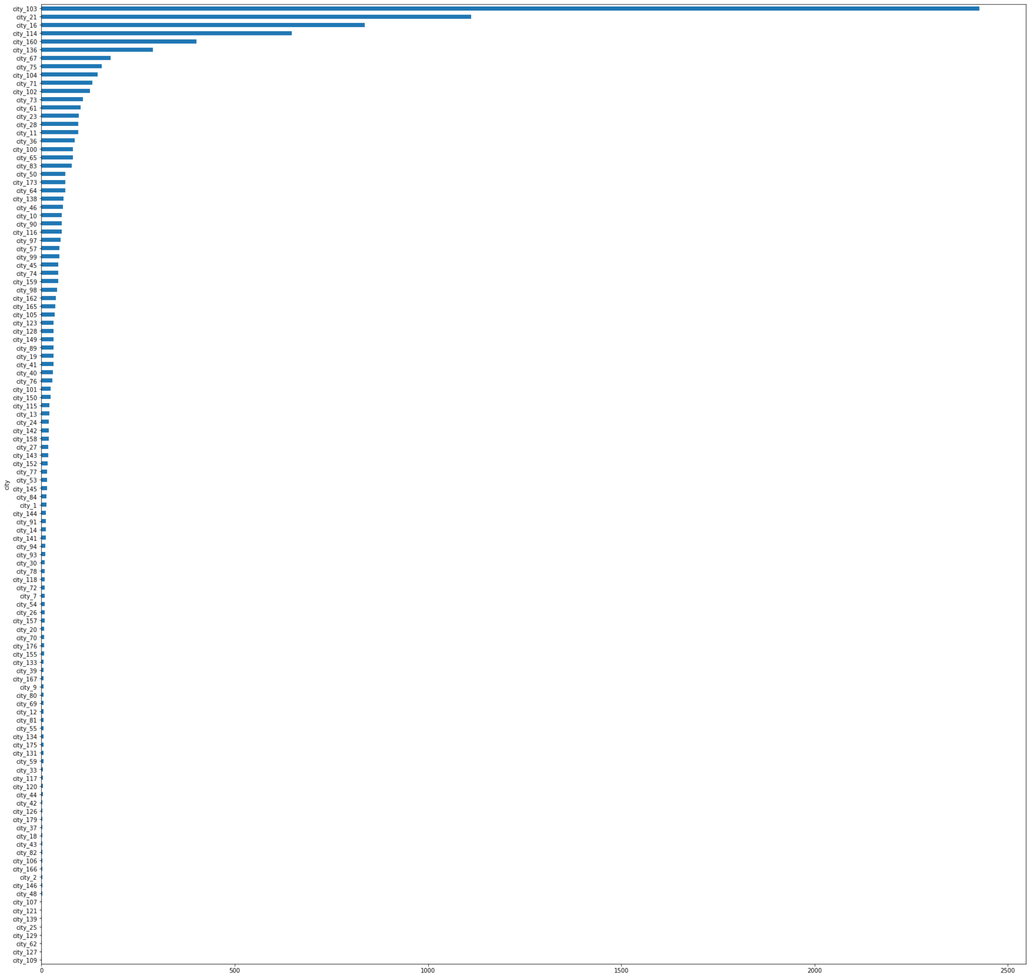
2. Feature Selection and Pre-processing (25 pts total)

2.1 Preprocessing City (1+1+1+1 = 4pts total)

2.1.1 Plot no. of records per city so that the highest city counts are shown in descending order (1pt)

```
# TODO
plt.figure(figsize = (30,30))
city_count = df.groupby(by=['city']).size().sort_values()
city_count.plot(kind='barh')
```

<AxesSubplot:ylabel='city'>



2.1.2 How many rows belong to the count-wise top 4 cities in total and how many for the remaining? (1pt)

```
# TODO
print('Top 4 cities contain a total of ',city_count.tail(4).sum(),'rows')
print('The rest contain a total of',city_count.iloc[:-4].sum(),'')
```

```
Top 4 cities contain a total of  5021 rows
The rest contain a total of 3934
```

2.1.3 Replace the city name with city_others if the city name is not among the top 4 (1pt)

```
# TODO
df['city'] = df['city'].replace(city_count.iloc[:-4].index,'city_other')
```

2.1.4 Show some sample data that the records have changed correctly. (1pt)

```
# TODO
print(df['city'])
print(df['city'].unique())
```

```
0       city_other
1       city_other
2       city_other
3         city_103
4         city_103
           ...
8950       city_21
8951      city_103
8952    city_other
8953      city_103
8954      city_103
Name: city, Length: 8955, dtype: object
['city_other' 'city_103' 'city_114' 'city_21' 'city_16']
```

2.2. Preprocessing Education Level (1+2+2+1 = 6pts total)

2.2.1. Show the unique values of education level. (1pt)

```
# TODO
print(df['education_level'].unique())
```

```
['Graduate' 'Masters' 'Phd']
```

2.2.2. Write a function named replace_labels() that can replace labels using given {old_label:new_label} dictionary (2pts)

Parameters: (1) dataframe, (2) a column name, (3) a dictionary with {old_label:new_label} mapping.

Returns: a dataframe with specified column values replaced with the

```
# TODO
def replace_labels(frame,col_name,labels):
  return frame.replace({col_name:labels})
```

2.2.3. Using the replace_labels() function you just created, replace education_level column with ordinal values. The mapping can be like "Graduate":0, "Masters":1, "Phd":2 . (2pt)

```
# TODO
mapping = {"Graduate":0,"Masters":1,"Phd":2}
df = replace_labels(df,'education_level',mapping)
```

2.2.4 Show some sample data that the records have changed appropriately (1pt)

```
# TODO
print(df['education_level'])
print(df['education_level'].unique())
```

```
0       0
1       1
2       0
3       0
4       0
       ..
8950    0
8951    1
8952    0
8953    0
8954    0
Name: education_level, Length: 8955, dtype: int64
[0 1 2]
```

2.3. Preprocessing company_size (2+2+1 = 5pts total)

2.3.1 Show the unique values of the company_size column and their counts (2pt)

```
# TODO
print(df['company_size'].unique())
print(df.groupby(by=['company_size']).size().sort_values())
```

```
['50-99' '<10' '5000-9999' '1000-4999' '10/49' '100-500' '10000+'
 '500-999']
company_size
5000-9999     393
500-999       592
<10           840
1000-4999     930
10/49         951
10000+       1449
100-500      1814
50-99        1986
dtype: int64
```

2.3.2 Change the values of the company_size column from 0 to 7 where e0 is <10 and 7 is 10000+. The order of the numbers should be based on the values of the column-like an ordinary variable. (2pt)

(Hint: you can use the replace_labels() function you created before.)

```
# TODO
mapping = {"<10":0,"10/49":1,"50-99":2,"100-500":3,"500-999":4,"1000-4999":5,"5000-9999":6,"10000+":7}
df = replace_labels(df,'company_size',mapping)
```

2.3.3 Show the updated unique values to validate they changed appropriately (1pt)

```
# TODO
print(df['company_size'])
print(df['company_size'].unique())
```

```
0       2
1       2
2       0
3       2
4       6
       ..
8950    3
8951    2
8952    3
8953    1
8954    2
Name: company_size, Length: 8955, dtype: int64
[2 0 6 5 1 3 7 4]
```

2.4. Preprocessing last_new_job (1+2+1 = 4pts total)

2.4.1 Show unique values of the last_new_job column (1pt)

```
# TODO
print(df['last_new_job'].unique())
print(df.groupby(by=['last_new_job']).size().sort_values())
```

```
['>4' '4' '1' '3' '2' 'never']
last_new_job
never     373
4         599
3         610
2        1570
>4       1965
1        3838
dtype: int64
```

2.4.2 Convert the values of this column to never->0, 1->1,....>4 -->5 (2pt)

Hint: replace_labels()

```
# TODO
mapping = {"never":0,"1":1,"2":2,"3":3,"4":4,">4":5}
df = replace_labels(df,'last_new_job',mapping)
```

2.4.3 Show the updated values (1pt)

```
# TODO
print(df['last_new_job'])
print(df['last_new_job'].unique())
```

```
0       5
1       4
2       5
3       1
4       1
       ..
8950    1
8951    1
8952    3
8953    1
8954    4
```

```
Name: last_new_job, Length: 8955, dtype: int64
[5 4 1 3 2 0]
```

2.5 Preprocessing other columns (2pt total)

2.5.1 Drop the enrollee_id, any unnamed columns, and any duplicate columns (if you created multiple columns one with original and one with updated, then remove the original one) (2pt)

```
# TODO
df = df.drop(columns=['enrollee_id'])
```

No other columns were created in the process and the unnamed was dropped upon importing the data.

2.6 Feature Scaling (3+1 = 4ps total)

2.6.1 Use sklearn.preprocessing's MinMaxScaler to perform min max scaling to all the numeric columns (3pt)

```
# TODO
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['city_development_index','experience','training_hours']] = scaler.fit_transform(df[['city_development_index','experience','tra
```

```
            city  city_development_index  gender      relevent_experience  \
0     city_other                   0.776    Male   No relevent experience
1     city_other                   0.767    Male  Has relevent experience
2     city_other                   0.762    Male  Has relevent experience
3       city_103                   0.920    Male  Has relevent experience
4       city_103                   0.920    Male  Has relevent experience
...          ...                     ...     ...                      ...
8950     city_21                   0.624    Male   No relevent experience
8951    city_103                   0.920    Male  Has relevent experience
8952  city_other                   0.920  Female  Has relevent experience
8953    city_103                   0.920  Female  Has relevent experience
8954    city_103                   0.920    Male  Has relevent experience

     enrolled_university  education_level major_discipline  experience  \
0          no_enrollment                0             STEM        15.0
1          no_enrollment                1             STEM        21.0
2          no_enrollment                0             STEM        13.0
3          no_enrollment                0             STEM         7.0
4          no_enrollment                0             STEM         5.0
...                  ...              ...              ...         ...
8950    Full time course                0             STEM         1.0
8951       no_enrollment                1             STEM         9.0
8952       no_enrollment                0             STEM        10.0
8953       no_enrollment                0       Humanities         7.0
8954       no_enrollment                0             STEM        21.0

      company_size    company_type  last_new_job  training_hours  target
0                2         Pvt Ltd             5              47     0.0
1                2  Funded Startup             4               8     0.0
2                0         Pvt Ltd             5              18     1.0
3                2         Pvt Ltd             1              46     1.0
4                6         Pvt Ltd             1             108     0.0
...            ...             ...           ...             ...     ...
8950             3         Pvt Ltd             1              52     1.0
8951             2         Pvt Ltd             1              36     1.0
8952             3   Public Sector             3              23     0.0
8953             1  Funded Startup             1              25     0.0
8954             2         Pvt Ltd             4              44     0.0

[8955 rows x 13 columns]
```

2.6.2 Show some of the scaled records. (1pt)

```
# TODO
print(df[['city_development_index','experience','training_hours']])
```

```
        city_development_index  experience  training_hours
0                     0.654691    0.714286        0.137313
1                     0.636727    1.000000        0.020896
2                     0.626747    0.619048        0.050746
3                     0.942116    0.333333        0.134328
4                     0.942116    0.238095        0.319403
...                        ...         ...             ...
8950                  0.351297    0.047619        0.152239
8951                  0.942116    0.428571        0.104478
8952                  0.942116    0.476190        0.065672
8953                  0.942116    0.333333        0.071642
8954                  0.942116    1.000000        0.128358

[8955 rows x 3 columns]
```

3. X/Y and Training/Test Split with stratified sampling (15pts in total)

3.1 Using a lot of features with categorical values is not memory-efficient. Use a LabelEncoder() to convert all the categorical columns to numeric labels. (This task is similar to previous assignment A1) (2pt)

```
# TODO
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['city'] = le.fit_transform(df['city'])
df['gender'] = le.fit_transform(df['gender'])
df['relevent_experience'] = le.fit_transform(df['relevent_experience'])
df['enrolled_university'] = le.fit_transform(df['enrolled_university'])
df['major_discipline'] = le.fit_transform(df['major_discipline'])
df['company_type'] = le.fit_transform(df['company_type'])
```

3.2 Copy all the features into X and the target to Y (2pt)

```
# TODO
x = df.loc[:, df.columns != 'target']
y = df['target']
```

3.3 Show the ratio of 1 and 0 in Y. (1pt)

```
# TODO
print(y.value_counts().to_frame('count').join(y.value_counts(normalize=True).to_frame('%')))
```

```
     count         %
0.0   7472  0.834394
1.0   1483  0.165606
```

3.4 Use sklearn's train_test_split() to split the data set into 70% training and 30% test sets. Set random_state to 42. We want to have the same ratio of 0 and 1 in the test set, use the stratify parameter to Y to ensure this. Then show the ratio of 1 and 0 in both train and test target. (4pt)

```
# TODO
from sklearn.model_selection import train_test_split as TTS;
TS = 0.3 #for tuning
print("\nTest Size = ", TS, "\n")
x_train, x_test, y_train, y_test = TTS(x,y, test_size=0.25, random_state=42,stratify=y)

print(y_train.value_counts().to_frame('count').join(y_train.value_counts(normalize=True).to_frame('%')))
print(y_test.value_counts().to_frame('count').join(y_test.value_counts(normalize=True).to_frame('%')))
```

```
Test Size =  0.3

     count         %
0.0   5604  0.834425
1.0   1112  0.165575
     count         %
0.0   1868  0.834301
1.0    371  0.165699
```

3.5 Rebalancing (4+2 = 6pts)

3.5.1 Use imblearn's SMOTENC to balance the x_train

When our training set have class imbalance, we often perform over-sampling to generate synthetic data that can help in training. SMOTE is a library by imblearn for this purpose. The usage is fairly straightforward. See documentation here and a brief explanation with example here

```
# TODO
```

3.5.2 Did that change the ratio in label? Confirm by printing the ratio in resampled labels.

```
# TODO
```

4. Decision Tree (20pts total)

4.1 Initialize a decision tree model using sklearns DecisionTreeClassifier. Use the unbalanced training set. Set a consistent value for random_state parameter so that your result is reproducible. (1pt)

```
# TODO
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf = clf.fit(x_train,y_train)
```

4.2 Use grid search to find out the best combination of values for the parameters: criterion, max_depth, min_samples_split, max_features. Then print the best performing parameters. (4pt)

```
# TODO
from sklearn.model_selection import GridSearchCV
params = {'criterion': ['gini','entropy'],'max_depth': list(range(2,100)), 'min_samples_split': [2, 3, 4],'max_features':['auto',
grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), params, verbose=1, cv=3)
grid_search_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 324 candidates, totalling 972 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=42),
```

```
                    param_grid={'criterion': ['gini', 'entropy'],
                                'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                                              14, 15, 16, 17, 18, 19],
                                'max_features': ['auto', 'sqrt', 'log2'],
                                'min_samples_split': [2, 3, 4]},
                    verbose=1)
```

4.3 Add the best performing parameter set to the already-initialized Decision Tree model. Then fit it on the train dataset. (2pt)

```
# TODO
print(grid_search_cv.best_estimator_)
clf = DecisionTreeClassifier(max_depth=3, max_features='auto', random_state=42)
clf = clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=3, max_features='auto', random_state=42)
```

4.4 Import the accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score from scikitlearn's metrics package. Evaluate your Decision Tree on the Test dataset and print all the metrics. (3pt)

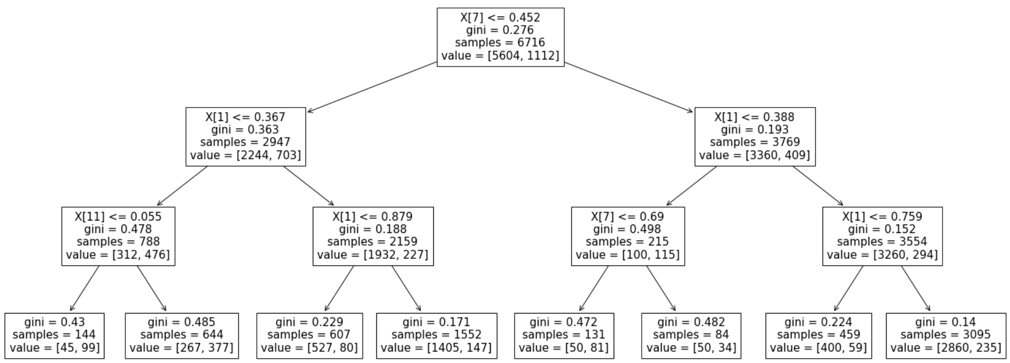
```
# TODO
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score

clf_pred = clf.predict(x_test)

print('accuracy_score = ',accuracy_score(y_test,clf_pred))
print('precision_score = ',precision_score(y_test,clf_pred))
print('recall_score = ',recall_score(y_test,clf_pred))
print('confusion_matrix = ',confusion_matrix(y_test,clf_pred))
print('f1_score = ',f1_score(y_test,clf_pred))
print('roc_auc_score = ',roc_auc_score(y_test,clf_pred))
```

```
accuracy_score =  0.8588655649843681
precision_score =  0.5816023738872403
recall_score =  0.5283018867924528
confusion_matrix =  [[1727  141]
 [ 175  196]]
f1_score =  0.5536723163841808
roc_auc_score =  0.7264100440386246
```

4.5 Plot the tree using scikitlearn's tree package. You may need to define a large figure size using matplotlib to have an intelligible figure. (2pt)

```
# TODO
from sklearn.tree import plot_tree
plt.figure(figsize = (25,10))
plot_tree(clf)
```

```
[Text(0.5, 0.875, 'X[7] <= 0.452\ngini = 0.276\nsamples = 6716\nvalue = [5604,
1112]'),
 Text(0.25, 0.625, 'X[1] <= 0.367\ngini = 0.363\nsamples = 2947\nvalue = [2244,
703]'),
 Text(0.125, 0.375, 'X[11] <= 0.055\ngini = 0.478\nsamples = 788\nvalue = [312,
476]'),
 Text(0.0625, 0.125, 'gini = 0.43\nsamples = 144\nvalue = [45, 99]'),
 Text(0.1875, 0.125, 'gini = 0.485\nsamples = 644\nvalue = [267, 377]'),
 Text(0.375, 0.375, 'X[1] <= 0.879\ngini = 0.188\nsamples = 2159\nvalue = [1932,
227]'),
 Text(0.3125, 0.125, 'gini = 0.229\nsamples = 607\nvalue = [527, 80]'),
 Text(0.4375, 0.125, 'gini = 0.171\nsamples = 1552\nvalue = [1405, 147]'),
 Text(0.75, 0.625, 'X[1] <= 0.388\ngini = 0.193\nsamples = 3769\nvalue = [3360,
409]'),
 Text(0.625, 0.375, 'X[7] <= 0.69\ngini = 0.498\nsamples = 215\nvalue = [100,
115]'),
 Text(0.5625, 0.125, 'gini = 0.472\nsamples = 131\nvalue = [50, 81]'),
 Text(0.6875, 0.125, 'gini = 0.482\nsamples = 84\nvalue = [50, 34]'),
 Text(0.875, 0.375, 'X[1] <= 0.759\ngini = 0.152\nsamples = 3554\nvalue = [3260,
294]'),
 Text(0.8125, 0.125, 'gini = 0.224\nsamples = 459\nvalue = [400, 59]'),
 Text(0.9375, 0.125, 'gini = 0.14\nsamples = 3095\nvalue = [2860, 235]')]
```



4.6 Initialize a new Decision Tree model, then use the best set of parameters from Step 4.3 to train it on the balanced train set that you prepared in Step 3.5.1. (3pt)

```
# TODO
```

4.7 Print the evaluation scores (accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score) from the training on balanced dataset. (3pt)

```
# TODO
```

4.8 Discuss any difference between evaluation results from the unbalanced train set and balanced train set. (2pt)

'#TODO'

5. Random Forest Classifier (12pts total)

5.1 Use grid search to find best combinations of the following Random Forest parameters: n_estimators, max_depth, min_samples_split and min_samples_leaf. Use your own choice of scoring, criterion, number of folds for cross-validation for the model initialization. Remember the grid search can take a while to finish. (4pt)

```
# TODO
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc = rfc.fit(x_train,y_train)

params = {'n_estimators': [100, 200, 300, 1000] ,'max_depth': [80, 90, 100, 110], 'min_samples_split': [8, 10, 12],'min_samples_le
grid_search_cv = GridSearchCV(RandomForestClassifier(random_state=42), params, verbose=1, cv=3)
grid_search_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [80, 90, 100, 110],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [8, 10, 12],
                         'n_estimators': [100, 200, 300, 1000]},
             verbose=1)
```

5.2 Print the best combination of parameters and use it to train a Random Forest classifier model. (3pt)

```
# TODO
print(grid_search_cv.best_estimator_)
rfc = RandomForestClassifier(max_depth=80, min_samples_leaf=5, min_samples_split=8,n_estimators=1000, random_state=42)
rfc = rfc.fit(x_train,y_train)
```

```
RandomForestClassifier(max_depth=80, min_samples_leaf=5, min_samples_split=8,
                       n_estimators=1000, random_state=42)
```

5.3 Evaluate using the same metrics as before (accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score) (5pt)

```
# TODO
rfc_pred = rfc.predict(x_test)

print('accuracy_score = ',accuracy_score(y_test,rfc_pred))
print('precision_score = ',precision_score(y_test,rfc_pred))
print('recall_score = ',recall_score(y_test,rfc_pred))
print('confusion_matrix = ',confusion_matrix(y_test,rfc_pred))
print('f1_score = ',f1_score(y_test,rfc_pred))
print('roc_auc_score = ',roc_auc_score(y_test,rfc_pred))
```

```
accuracy_score =  0.8575256811076374
precision_score =  0.5866666666666667
recall_score =  0.4743935309973046
confusion_matrix =  [[1744  124]
 [ 195  176]]
f1_score =  0.5245901639344263
roc_auc_score =  0.704006187340194
```

6. Boosting Classifier (20 pts total)

6.1 AdaBoost Classifier (10 pts total)

6.1.1 Perform a grid search for best values for parameters={n_estimators, learning_rate} of an AdaBoostClassifier and the given training set. (4pt)

```
# TODO
```

```
# TODO
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ada = ada.fit(x_train,y_train)

params = {'n_estimators': [10, 50, 100, 500] ,'learning_rate':[0.0001, 0.001, 0.01, 0.1, 1.0]}
grid_search_cv = GridSearchCV(AdaBoostClassifier(random_state=42), params, verbose=1, cv=3)
grid_search_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
GridSearchCV(cv=3, estimator=AdaBoostClassifier(random_state=42),
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
                         'n_estimators': [10, 50, 100, 500]},
             verbose=1)
```

6.1.2 Train an AdaboostClassifier using the best parameter set you found in step 6.1.1 (3pt)

```
# TODO
print(grid_search_cv.best_estimator_)
ada = AdaBoostClassifier(learning_rate=0.1, random_state=42)
ada = ada.fit(x_train,y_train)
```

```
AdaBoostClassifier(learning_rate=0.1, random_state=42)
```

6.1.3 Evaluate using the same metrics as before (accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score) (3pt)

```
# TODO
ada_pred = ada.predict(x_test)

print('accuracy_score = ',accuracy_score(y_test,ada_pred))
print('precision_score = ',precision_score(y_test,ada_pred))
print('recall_score = ',recall_score(y_test,ada_pred))
print('confusion_matrix = ',confusion_matrix(y_test,ada_pred))
print('f1_score = ',f1_score(y_test,ada_pred))
print('roc_auc_score = ',roc_auc_score(y_test,ada_pred))
```

```
accuracy_score =  0.8610987047789191
precision_score =  0.5842696629213483
recall_score =  0.5606469002695418
confusion_matrix =  [[1720  148]
 [ 163  208]]
f1_score =  0.5722145804676754
roc_auc_score =  0.7407088891069336
```

6.2 Gradient Boosting Classifier (10 pts total)

6.2.1 Perform a grid search for best values for parameters={n_estimators, max_depth, learning_rate} of a GradientBoostingClassifier and the given training set. (4pt)

```
# TODO
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb = gb.fit(x_train,y_train)

params = {'n_estimators': [10, 50, 100, 500] ,'max_depth':[3,5,8],'learning_rate':[0.0001, 0.001, 0.01, 0.1, 1.0]}
grid_search_cv = GridSearchCV(GradientBoostingClassifier(random_state=42), params, verbose=1, cv=3)
grid_search_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=42),
```

```
                    param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
                                'max_depth': [3, 5, 8],
                                'n_estimators': [10, 50, 100, 500]},
                    verbose=1)
```

6.2.2 Train a GradientBoostingClassifier using the best parameter set you found in step 6.2.1 (3pt)

```
# TODO
print(grid_search_cv.best_estimator_)
gb = GradientBoostingClassifier(learning_rate=0.01, n_estimators=500,random_state=42)
gb = gb.fit(x_train,y_train)
```

```
     GradientBoostingClassifier(learning_rate=0.01, n_estimators=500,
                                random_state=42)
```

6.2.3 Evaluate using the same metrics as before (accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score) (3pt)

```
# TODO
gb_pred = gb.predict(x_test)

print('accuracy_score = ',accuracy_score(y_test,gb_pred))
print('precision_score = ',precision_score(y_test,gb_pred))
print('recall_score = ',recall_score(y_test,gb_pred))
print('confusion_matrix = ',confusion_matrix(y_test,gb_pred))
print('f1_score = ',f1_score(y_test,gb_pred))
print('roc_auc_score = ',roc_auc_score(y_test,gb_pred))
```

```
     accuracy_score =  0.8619919606967397
     precision_score =  0.5933734939759037
     recall_score =  0.5309973045822103
     confusion_matrix =  [[1733  135]
      [ 174  197]]
     f1_score =  0.5604551920341394
     roc_auc_score =  0.7293637486508482
```

7. Summary Discussion (4 pts)

Which model yields the highest precision?

Which model yields the lowest recall?

Which model yields the higest True Positive (TP)?

Which model yields the best performance overall?