```
#This jupyter notebook is prepared by “Marco Padlan”.
```

```
#import libraries: pandas, numpy, matplotlib (set %matplotlib inline), matplotlib’s py
import pandas as pd
import numpy as np
import missingno as mns
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.stats as st
from sklearn import ensemble, tree, linear_model
import sklearn
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import LinearSVC
from imblearn.over_sampling import SMOTE, ADASYN
from numpy import mean
from numpy import std
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold, train_test_split, GridSea
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, make_scorer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
```

New Section

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr.
```

```
data = pd.read_csv('/content/drive/MyDrive/hrdata2 (1).csv')
```

```
data.shape
```

```
(8955, 15)
```

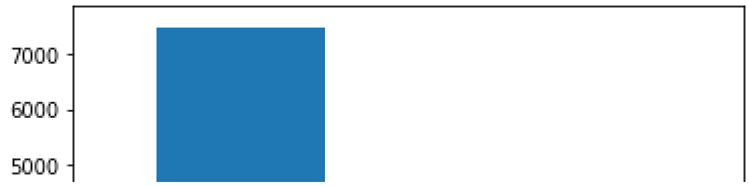
```
data.head()
```

	Unnamed: 0	enrollee_id	city	city_development_index	gender	relevent_experi
0	1	29725	city_40	0.776	Male	No relevent exper
1	4	666	city_162	0.767	Male	Has relevent exper
2	7	402	city_46	0.762	Male	Has relevent exper
3	8	27107	city_103	0.920	Male	Has relevent exper
4	11	23853	city_103	0.920	Male	Has relevent exper

```
nulls= data.isnull().sum().to_frame("nulls")
```

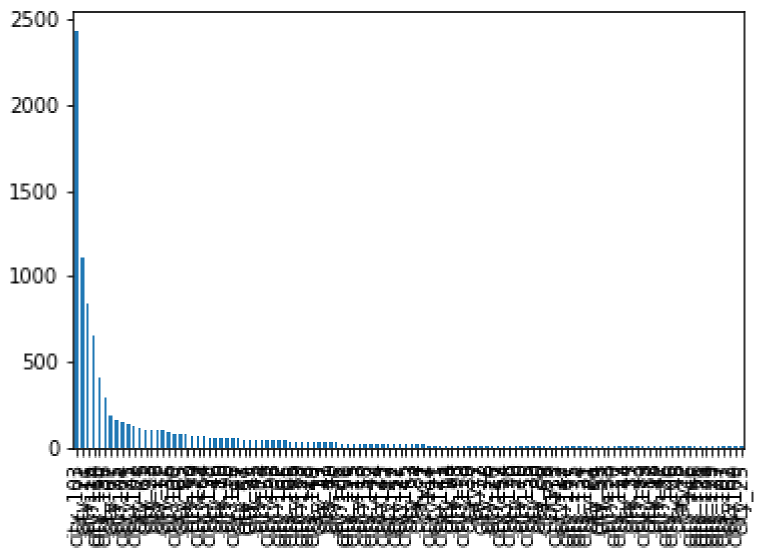
```
#Plot the count of target and discuss its imbalances and probably issues and solutions
data["target"].value_counts().plot(kind='bar')
plt.show
```

```
<function matplotlib.pyplot.show>
```




```
#2  Feature Selection and Pre-processing
##Plot #of records per city so that the highest city counts are shown in descending o1
data["city"].value_counts().plot(kind='bar')
plt.show
```

```
<function matplotlib.pyplot.show>
```




```
#How many rows belong to the count-wise top 4 cities in total and how many for the rem
##(The plot you have generated in 2.i.i should help you to identify those cities)
result = data[ data.city == 'city_103'].shape[0] + data[data.city == 'city_21'].shape|
print(result)
print(data["city"].shape[0]-result)
```

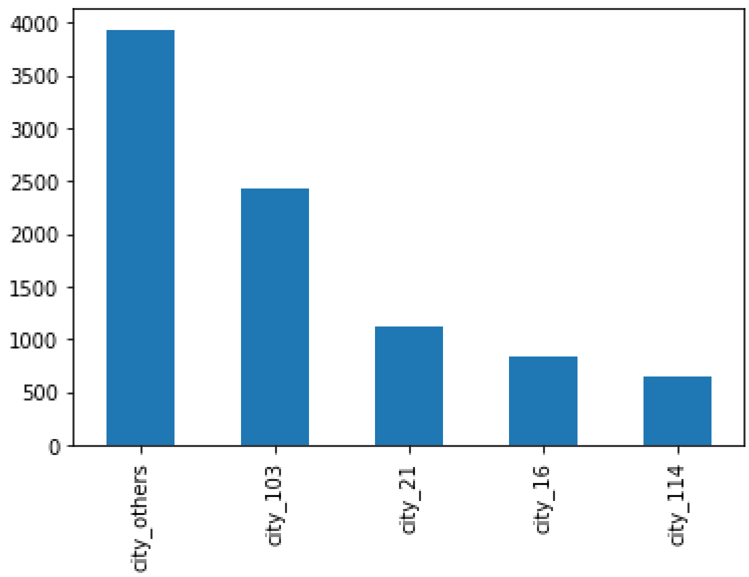
```
5021
3934
```

```
#Replace the city name with city_others if the city name is not within the top 4 city
result = set(data["city"])
result.remove('city_103')
result.remove('city_21')
result.remove('city_16')
result.remove('city_114')
data.loc[data['city'].isin(result), 'city'] = "city_others"
```

```
#Show some sample data that the records have changed appropriately
data["city"].value_counts().plot(kind='bar')
```

```
plt.show
```

```
<function matplotlib.pyplot.show>
```



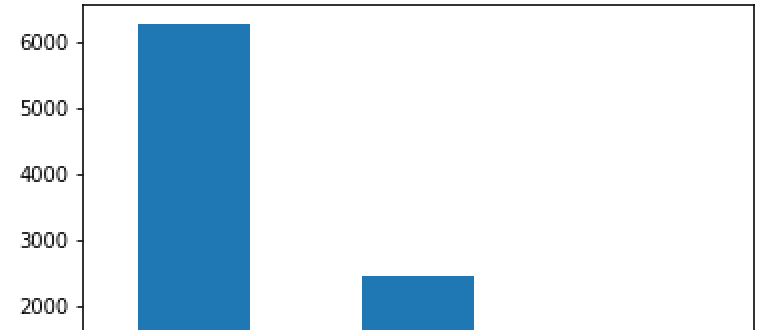

```
#Education Level:
##Show the unique values of education level.
print(data["education_level"])
```

```
0       Graduate
1        Masters
2       Graduate
3       Graduate
4       Graduate
          ...
8950    Graduate
8951     Masters
8952    Graduate
8953    Graduate
8954    Graduate
Name: education_level, Length: 8955, dtype: object
```

```
#Replace the value of Education level column like ordinal values, "Graduate" -> 0, Mas
delta={'Masters':1, 'Phd' : 2, 'Graduate' :0}
var=data.replace({'education_level' : delta})
data=var
```

```
#Show some sample data that the records have changed appropriately
data["education_level"].value_counts().plot(kind='bar')
plt.show
```

```
<function matplotlib.pyplot.show>
```

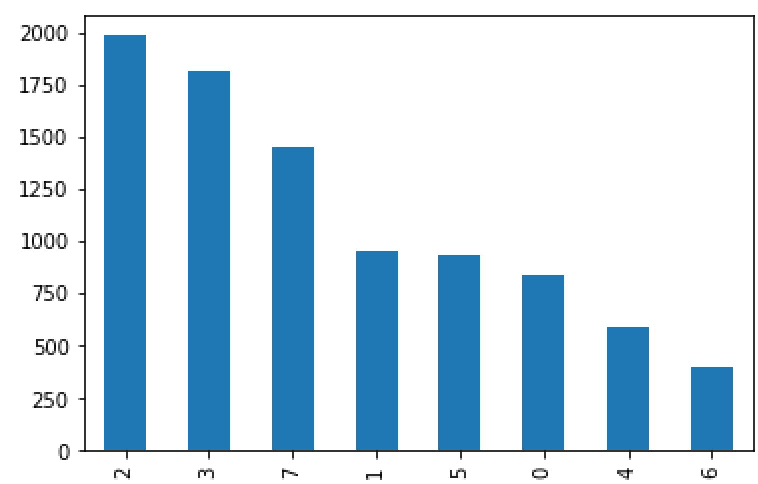



```
#company_size column:
##Show the unique values of the company_size column
print(data["company_size"])
```

```
0          50-99
1          50-99
2            <10
3          50-99
4      5000-9999
         ...
8950     100-500
8951       50-99
8952     100-500
8953       10/49
8954       50-99
Name: company_size, Length: 8955, dtype: object
```

```
#Change the values of the company_size column from 0 to 7 where e0 is <10 and 7 is 10(
delta={'<10': 0, '10/49': 1, '50-99' : 2,'100-500':3, '500-999': 4,'1000-4999':5, '50(
var=data.replace({'company_size' : delta})
data=var


#Show the updated unique values
data["company_size"].value_counts().plot(kind='bar')
plt.show
```

```
<function matplotlib.pyplot.show>
```



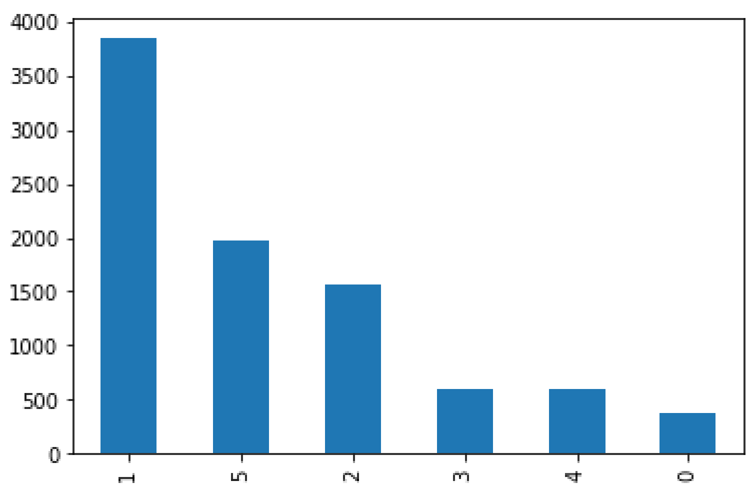
```
#Show the unique values of the last_new_job column
print(data["last_new_job"])
```

```
0       >4
1        4
2       >4
3        1
4        1
        ..
8950     1
8951     1
8952     3
8953     1
8954     4
Name: last_new_job, Length: 8955, dtype: object
```

```
#Convert the values of this column to never->0, 1->1,....>4 -->5
delta={'never': 0, '1': 1, '2' : 2,'3':3, '4': 4,'>4':5,}
var=data.replace({'last_new_job' : delta})
data=var
```

```
#Show the updated values
data["last_new_job"].value_counts().plot(kind='bar')
plt.show
```

```
<function matplotlib.pyplot.show>
```




```
#Show the unique values of company_type, major_descipline, enrolled_university, releva

output = set()
for x in data['company_type']:
    output.add(x)
print(output)
output = set()
for x in data['major_discipline']:
    output.add(x)
```

```
print(output)
output = set()
for x in data['enrolled_university']:
    output.add(x)
print(output)

output = set()
for x in data['relevent_experience']:
    output.add(x)
print(output)

output = set()
for x in data['gender']:
    output.add(x)
print(output)
output = set()
for x in data['city']:
    output.add(x)
print(output)
```

```
{'Pvt Ltd', 'Funded Startup', 'Other', 'NGO', 'Public Sector', 'Early Stage Star
{'Other', 'Business Degree', 'Humanities', 'STEM', 'Arts', 'No Major'}
{'no_enrollment', 'Part time course', 'Full time course'}
{'No relevent experience', 'Has relevent experience'}
{'Male', 'Other', 'Female'}
{'city_21', 'city_103', 'city_114', 'city_others', 'city_16'}
```

```
#As one-hot encoding is a bit strict, use panda's get_dummies function to create binar
data= pd.get_dummies(data=data, columns=['company_type','major_discipline','enrolled_u
```

```
#SHow the top 5 and last 5 rows to show that the table has changed [You must set this
data.shape
```

```
(8955, 34)
```

```
data.head()
```

	Unnamed: 0	enrollee_id	city_development_index	education_level	experience	co
0	1	29725	0.776	0	15.0	
1	4	666	0.767	1	21.0	
2	7	402	0.762	0	13.0	
3	8	27107	0.920	0	7.0	
4	11	23853	0.920	0	5.0	

5 rows × 34 columns

```
#Also, show the shape of the table
data.shape
```

```
(8955, 34)
```

```
#Drop the enrollee_id and any duplicate columns
data.drop("enrollee_id",axis=1)
```

	Unnamed: 0	city_development_index	education_level	experience	company_size
0	1	0.776	0	15.0	2
1	4	0.767	1	21.0	2
2	7	0.762	0	13.0	0
3	8	0.920	0	7.0	2
4	11	0.920	0	5.0	6
...	...	...	...	...	...
8950	19147	0.624	0	1.0	3
8951	19149	0.920	1	9.0	2
8952	19150	0.920	0	10.0	3
8953	19152	0.920	0	7.0	1
8954	19155	0.920	0	21.0	2

8955 rows × 33 columns

```
#Feature Scaling
##Use sklearn.preprocessing's MinMaxScaler to perform min max scaling to all the colur
```

```
scaler = MinMaxScaler()
col_names = data.columns
data_scaled = scaler.fit_transform(data.to_numpy())
data_scaled = pd.DataFrame(data_scaled,columns=col_names)


#Show sample records that show some the scaled records
data_scaled.head()
```

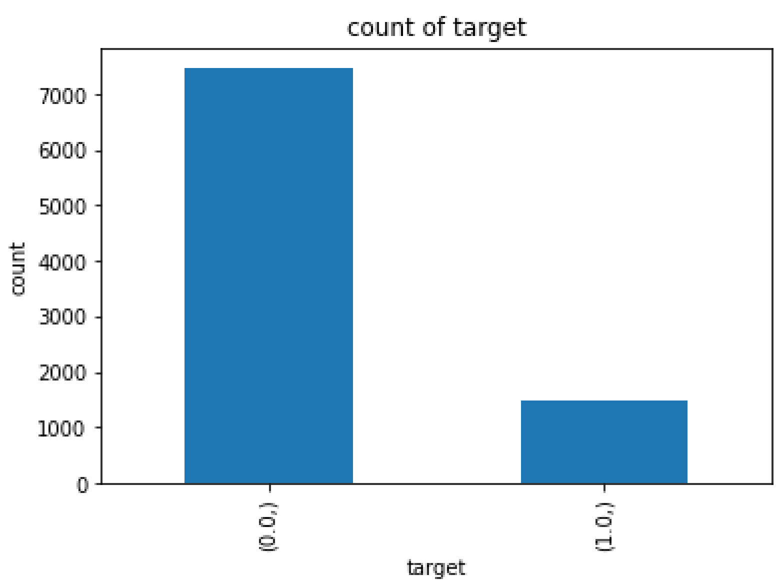
	Unnamed: 0	enrollee_id	city_development_index	education_level	experience	co
0	0.000000	0.890497	0.654691	0.0	0.714286	
1	0.000157	0.019893	0.636727	0.5	1.000000	
2	0.000313	0.011984	0.626747	0.0	0.619048	
3	0.000365	0.812062	0.942116	0.0	0.333333	
4	0.000522	0.714572	0.942116	0.0	0.238095	

5 rows × 34 columns

```
#Move the target column to the last column of the data frame and show that it has char
data_scaled = data_scaled[['Unnamed: 0', 'city_development_index', 'education_level',
       'company_size', 'last_new_job', 'training_hours',
       'company_type_Early Stage Startup', 'company_type_Funded Startup',
       'company_type_NGO', 'company_type_Other', 'company_type_Public Sector',
       'company_type_Pvt Ltd', 'major_discipline_Arts',
       'major_discipline_Business Degree', 'major_discipline_Humanities',
       'major_discipline_No Major', 'major_discipline_Other',
       'major_discipline_STEM', 'enrolled_university_Full time course',
       'enrolled_university_Part time course',
       'enrolled_university_no_enrollment',
       'relevent_experience_Has relevent experience',
       'relevent_experience_No relevent experience', 'gender_Female',
       'gender_Male', 'gender_Other', 'city_city_103', 'city_city_114',
       'city_city_16', 'city_city_21', 'city_city_others','target']]
data_scaled.head()
```

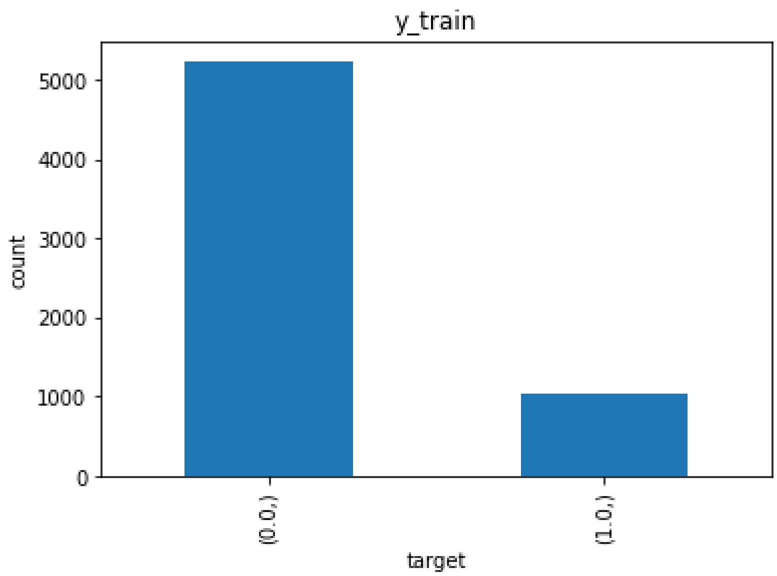
	Unnamed: 0	city_development_index	education_level	experience	company_size	1
0	0.000000	0.654691	0.0	0.714286	0.285714	
1	0.000157	0.636727	0.5	1.000000	0.285714	
2	0.000313	0.626747	0.0	0.619048	0.000000	
3	0.000365	0.942116	0.0	0.333333	0.285714	

```
#3. X/Y and Training/Test Split with stratified sampling and SMOTE
##Copy all the features into X and the target to Y
X = data[['city_development_index', 'education_level', 'experience',
       'company_size', 'last_new_job', 'training_hours',
       'company_type_Early Stage Startup', 'company_type_Funded Startup',
       'company_type_NGO', 'company_type_Other', 'company_type_Public Sector',
       'company_type_Pvt Ltd', 'major_discipline_Arts',
       'major_discipline_Business Degree', 'major_discipline_Humanities',
       'major_discipline_No Major', 'major_discipline_Other',
       'major_discipline_STEM', 'enrolled_university_Full time course',
       'enrolled_university_Part time course',
       'enrolled_university_no_enrollment',
       'relevent_experience_Has relevent experience',
       'relevent_experience_No relevent experience', 'gender_Female',
       'gender_Male', 'gender_Other', 'city_city_103', 'city_city_114',
       'city_city_16', 'city_city_21', 'city_city_others']]
Y = data[['target']]


#Show the ratio of 1 and 0 in Y
Y.value_counts().plot(kind='bar')
plt.title('count of target')
plt.xlabel('target')
plt.ylabel('count')
plt.show()
```



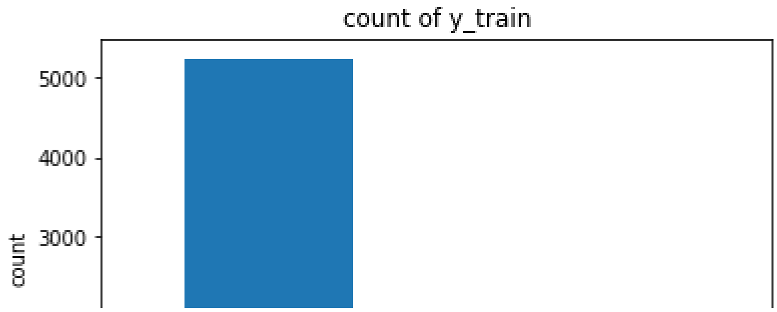
```
#Use sklearn's train_test_split to split the data set into training and test sets.
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, Y, test
```

```
#Show the ratio of 1 and 0 in y_train and then y_test
y_train.value_counts().plot(kind='bar')
plt.title('y_train')
plt.xlabel('target')
plt.ylabel('count')
plt.show()
```



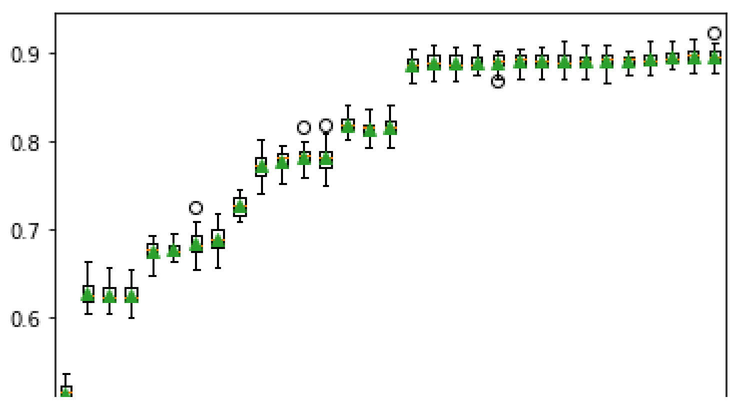

```
#Rebalance:
##Use imblearn's SMOTE to balance the x_train
X_bal, y_bal = SMOTE().fit_resample(X_train, y_train)
##change scale to bal
```

```
#Show the ratio of 0 and 1 in Y_train after rebalancing.
y_train.value_counts().plot(kind='bar')
plt.title('count of y_train')
plt.xlabel('target')
plt.ylabel('count')
plt.show()
#do you have 50% of each class now? Yes I believe so
```




```
#4PCA and Logistic Regression
#As part of it, create pipeline to find how many dimensions give you the best logistic
def get_dataset():
    X, y = make_classification(n_samples=5000, n_features=20, n_informative=15, n_redu
    return X, y

# get a list of models to evaluate
def get_models():
    models = dict()
    for i in range(1,32):
        steps = [('pca', PCA(n_components=i)), ('m', LogisticRegression())]
        models[str(i)] = Pipeline(steps=steps)
    return models

# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_
    return scores

# define dataset
#X, y = get_dataset()

print(X_bal.shape, y_bal.shape)
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_bal, y_bal)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
plt.boxplot(results, labels=names, showmeans=True)
plt.xticks(rotation=45)
plt.show()
```

```
(10460, 31) (10460, 1)
>1 0.515 (0.014)
>2 0.628 (0.015)
>3 0.626 (0.012)
>4 0.627 (0.012)
>5 0.675 (0.011)
>6 0.678 (0.008)
>7 0.684 (0.016)
>8 0.689 (0.014)
>9 0.727 (0.011)
>10 0.772 (0.016)
>11 0.777 (0.012)
>12 0.782 (0.012)
>13 0.781 (0.015)
>14 0.818 (0.009)
>15 0.813 (0.010)
>16 0.816 (0.012)
>17 0.886 (0.009)
>18 0.888 (0.010)
>19 0.888 (0.010)
>20 0.889 (0.009)
>21 0.889 (0.009)
>22 0.889 (0.009)
>23 0.890 (0.010)
>24 0.889 (0.010)
>25 0.890 (0.009)
>26 0.890 (0.010)
>27 0.889 (0.007)
>28 0.892 (0.009)
>29 0.894 (0.007)
>30 0.895 (0.010)
>31 0.895 (0.010)
```




```
#Based on the number of features chosen in the above step, use the test set to evaluat
# define the model
steps = [('pca', PCA(n_components=15)), ('m', LogisticRegression())]
model = Pipeline(steps=steps)
# fit the model on the whole dataset
model.fit(X_bal, y_bal)
# make a single prediction
y_pred = model.predict(X_test)
sklearn.metrics.accuracy_score(y_test, y_pred, normalize=True, sample_weight=None)
```

```
    pected. Please change the shape of y to (n_samples, ), for example using ravel().
```

```
#Show the confusion matrix and interpret the numbers in the confusion matrix
(([2166,   76],
[ 440,    5]))
```

```
([2166, 76], [440, 5])
```

```
#Show precision, recall, and f1 score
precision_score(y_test, y_pred)
```

```
0.362012987012987
```

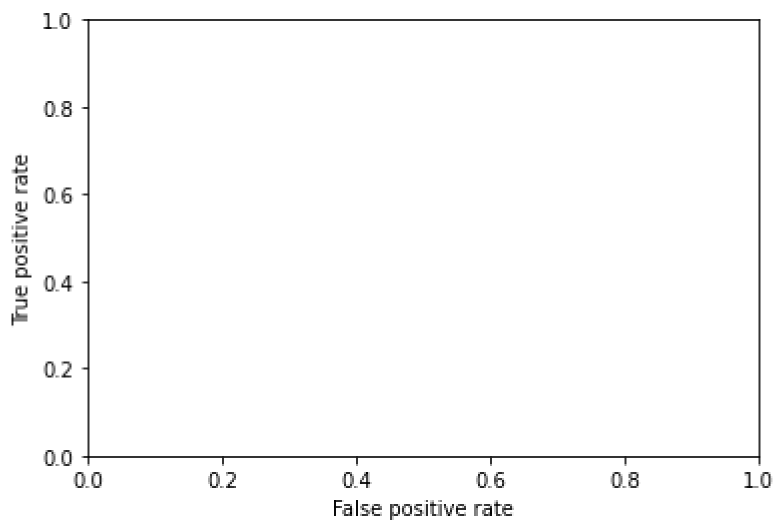
```
recall_score(y_test, y_pred)
```

```
f1_score(y_test, y_pred)
```

```
0.4203581526861452
```

```
# Plot ROC curve and find AUC (the same google colab link should help you)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

print(tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

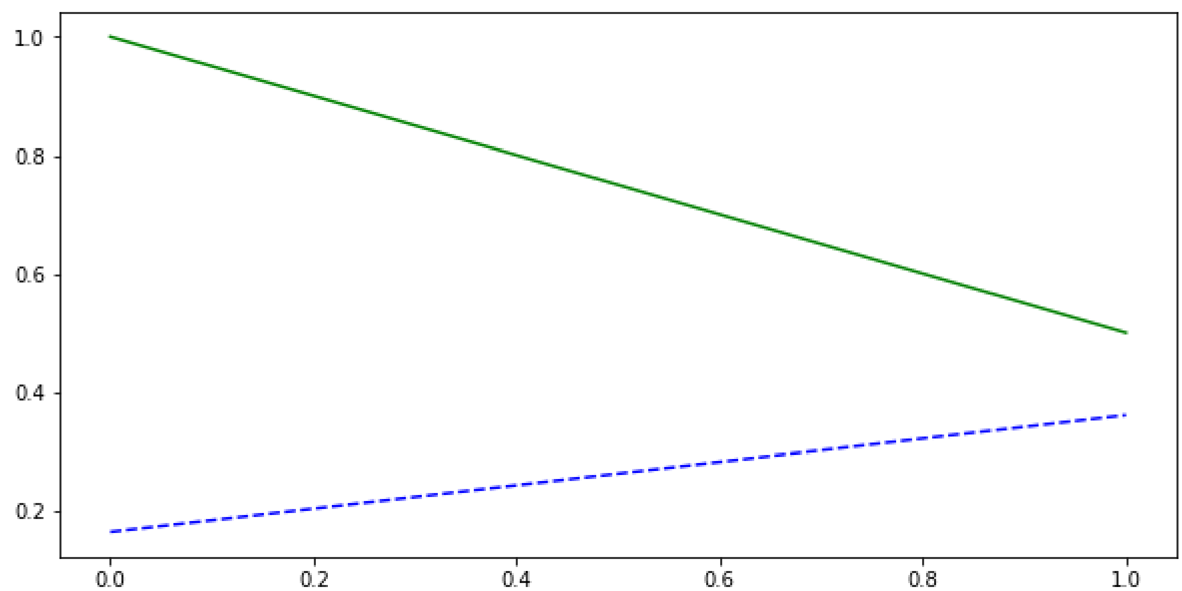
```
[0.        0.5011236 1.       ]
```




```
roc_auc_score(y_test, y_pred)
```

```
0.6629168378955387
```

```
#plot precision-recall curve for different thresholds and discuss the plot

precisions, recalls, thresholds = precision_recall_curve(y_test, y_pred)
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
 plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
 plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

plt.figure(figsize=(10,5))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```




```
#How softmax regression is related to logistic regression? What library can you use fo
#Softmax Regression is a generalization of Logistic Regression that
#summarizes a k dimensional vector of arbitrary values to a k dimensional vector of va
# we could use the library from mlxtend.classier import SoftmaxRegression
```

```
#6 KNN
##Use sklearn's KNN classifier to train (with k=  10) and predict the model based on t
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
  return self._fit(X, y)
[0. 0. 0. ... 0. 0. 0.]
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[2227   15]
 [ 433   12]]
              precision    recall  f1-score   support

         0.0       0.84      0.99      0.91      2242
         1.0       0.44      0.03      0.05       445

    accuracy                           0.83      2687
   macro avg       0.64      0.51      0.48      2687
weighted avg       0.77      0.83      0.77      2687
```

```
#Use sklearn's KNN classifier to train (with k=  10)
#and predict the model based on the rebalanced training set and test it and show the c
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(X_bal, y_bal)
y_pred = classifier.predict(X_test)
print(y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
  return self._fit(X, y)
[1. 0. 1. ... 1. 1. 0.]
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[1516  726]
 [ 220  225]]
              precision    recall  f1-score   support

         0.0       0.87      0.68      0.76      2242
         1.0       0.24      0.51      0.32       445

    accuracy                           0.65      2687
   macro avg       0.55      0.59      0.54      2687
weighted avg       0.77      0.65      0.69      2687
```

```
#Use grid search to tune the following hyperparameters of KNN
#: number of neighbors (between 1 and 20), weights  (uniform or distance),
#and metrics (Euclidean, Manhattan, or Minkowski)istance) to use for KNN.
knn_params = {
    "n_neighbors": range(1, 20),
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan", "minkowski"],
}
scoring = {"AUC": "roc_auc", "Accuracy": make_scorer(accuracy_score)}
knn = KNeighborsClassifier()
```

```
#grid search
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3)
grid_search = GridSearchCV(estimator=knn, param_grid=knn_params, n_jobs=-1, cv=cv, sco
grid_results = grid_search.fit(X_bal, y_bal)

#best model
final_model = knn.set_params(**grid_results.best_params_)
final_model.fit(X_train, y_train)
y_pred = final_model.predict(X_test)

#summarize results
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(grid_results.best_params_)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
  return self._fit(X, y)
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
  return self._fit(X, y)
              precision    recall  f1-score   support

         0.0       0.85      0.97      0.91      2242
         1.0       0.47      0.14      0.22       445

    accuracy                           0.83      2687
   macro avg       0.66      0.56      0.56      2687
weighted avg       0.79      0.83      0.79      2687

[[2170   72]
 [ 381   64]]
{'metric': 'manhattan', 'n_neighbors': 8, 'weights': 'distance'}
```

```
#After completing the process, print the best_params_
print(grid_results.best_params_)
```

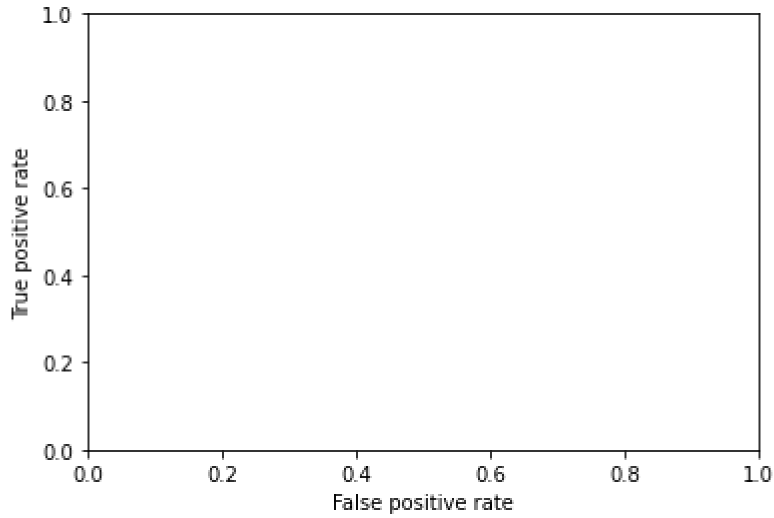
```
#Based on the result from grid search, use the parameters to train a model, test it wi
#and classification report. Also, show the AUC of ROC.
classifier = KNeighborsClassifier(n_neighbors=13, weights='distance', metric='manhatta
classifier.fit(X_bal, y_bal)
y_pred = classifier.predict(X_test)
print(y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
  return self._fit(X, y)
[0. 0. 1. ... 1. 0. 0.]
```

```
#Use PCA and based on that train model, test it and then print the confusion matrix an
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')

print(tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

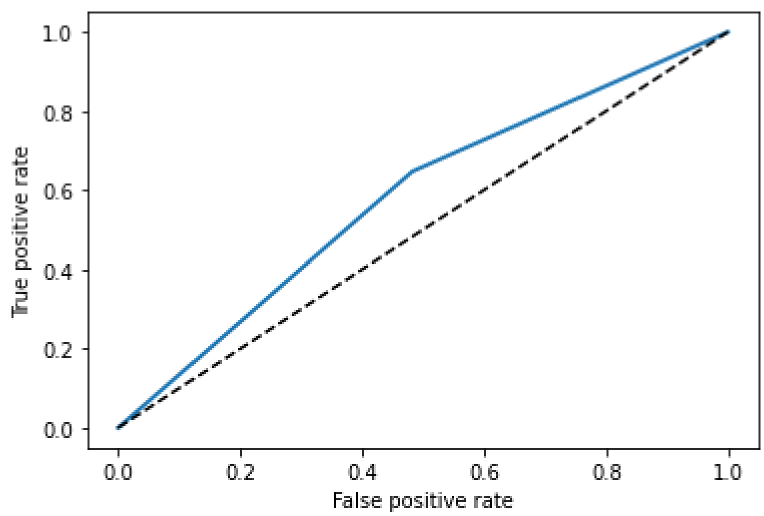
```
[0.         0.52359551 1.        ]
```




```
#Also, show the AUC of ROC.
roc_auc_score(y_test, y_pred)


#A short discussion on the 4 models and their differences.


#7
#Train a model with GaussianNB,
##test it and then print the confusion matrix and classification report.
###Also, plot ROC curve and show the AUC of ROC, and the count of the number of miscla

#train w/ with GaussianNB,
GaussianNB()
gnb = GaussianNB()
gnb.fit(X_bal, y_bal)
#test
y_pred_gnb = gnb.predict(X_test)

#print
print("GaussianNB")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

#plot
```

```
y_pred = gnb.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

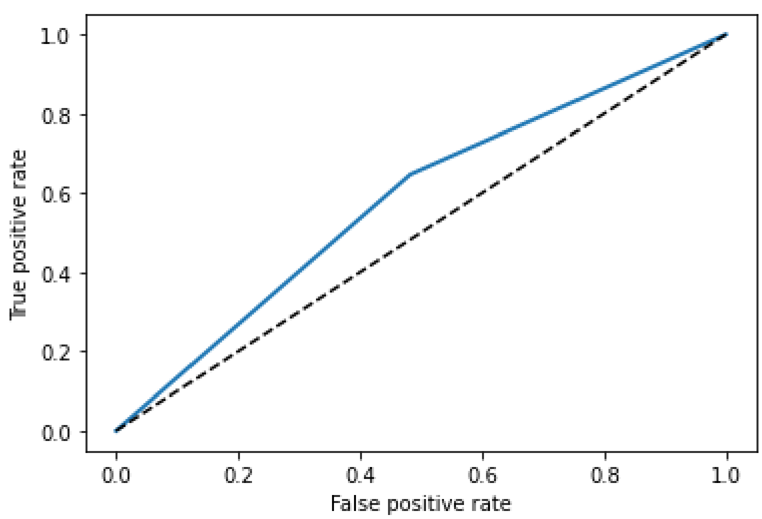
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConv
  y = column_or_1d(y, warn=True)
GaussianNB
[[2170   72]
 [ 381   64]]
              precision    recall  f1-score   support

         0.0       0.85      0.97      0.91      2242
         1.0       0.47      0.14      0.22       445

    accuracy                           0.83      2687
   macro avg       0.66      0.56      0.56      2687
weighted avg       0.79      0.83      0.79      2687
```




```
#Train a model with CategoricalNB,
##test it and then print the confusion matrix and classification report.
##Also, plot ROC curve, and show the AUC of ROC and the count of the number of misclas

#train
#CategoricalNB()
gnb = GaussianNB()
gnb.fit(X_bal, y_bal)
#test
y_pred_cnb = gnb.predict(X_test)
```

```
print("CategoricalNB")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
#plot
y_pred = gnb.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConv
  y = column_or_1d(y, warn=True)
CategoricalNB
[[1160 1082]
 [ 157  288]]
              precision    recall  f1-score   support

         0.0       0.88      0.52      0.65      2242
         1.0       0.21      0.65      0.32       445

    accuracy                           0.54      2687
   macro avg       0.55      0.58      0.48      2687
weighted avg       0.77      0.54      0.60      2687
```




```
#8 Support Vector Machine
##Build a support vector machine model using SVC.
##Use grid search to tune some parameters and then based on that show the best paramet
```

```
#build SVC
param_grid = {'C': [0.1,1, 10],
              'kernel': ['rbf', 'sigmoid', 'poly']
              } # create list of paraeters you would like to tune. We have already gor

grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, cv =3, n_jobs=-1)
grid.fit(X_bal, y_bal)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConve
  y = column_or_1d(y, warn=True)
GridSearchCV(cv=3, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10],
                         'kernel': ['rbf', 'sigmoid', 'poly']},
             verbose=2)
```

```
grid.best_params_
```

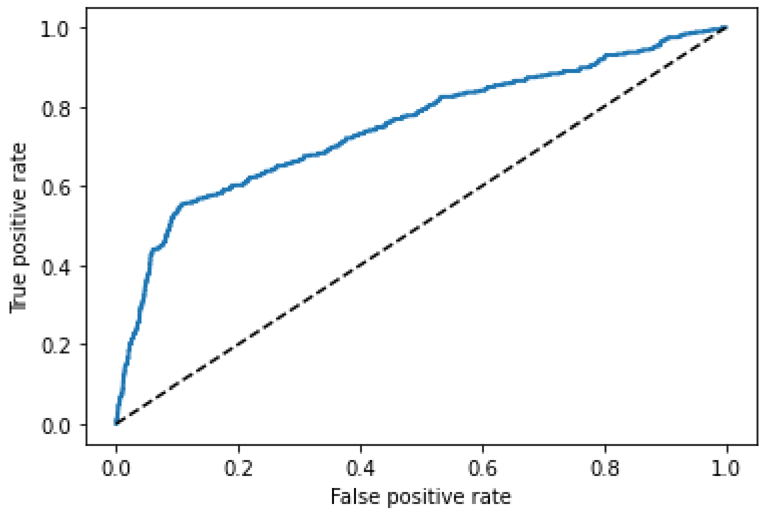
```
{'C': 10, 'kernel': 'rbf'}
```

```
model = SVC(kernel = 'rbf', C=100)
model.fit(X_bal, y_bal)
y_pred= model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConve
  y = column_or_1d(y, warn=True)
[[2104  138]
 [ 250  195]]
              precision    recall  f1-score   support

         0.0       0.89      0.94      0.92      2242
         1.0       0.59      0.44      0.50       445

    accuracy                           0.86      2687
   macro avg       0.74      0.69      0.71      2687
weighted avg       0.84      0.86      0.85      2687
```

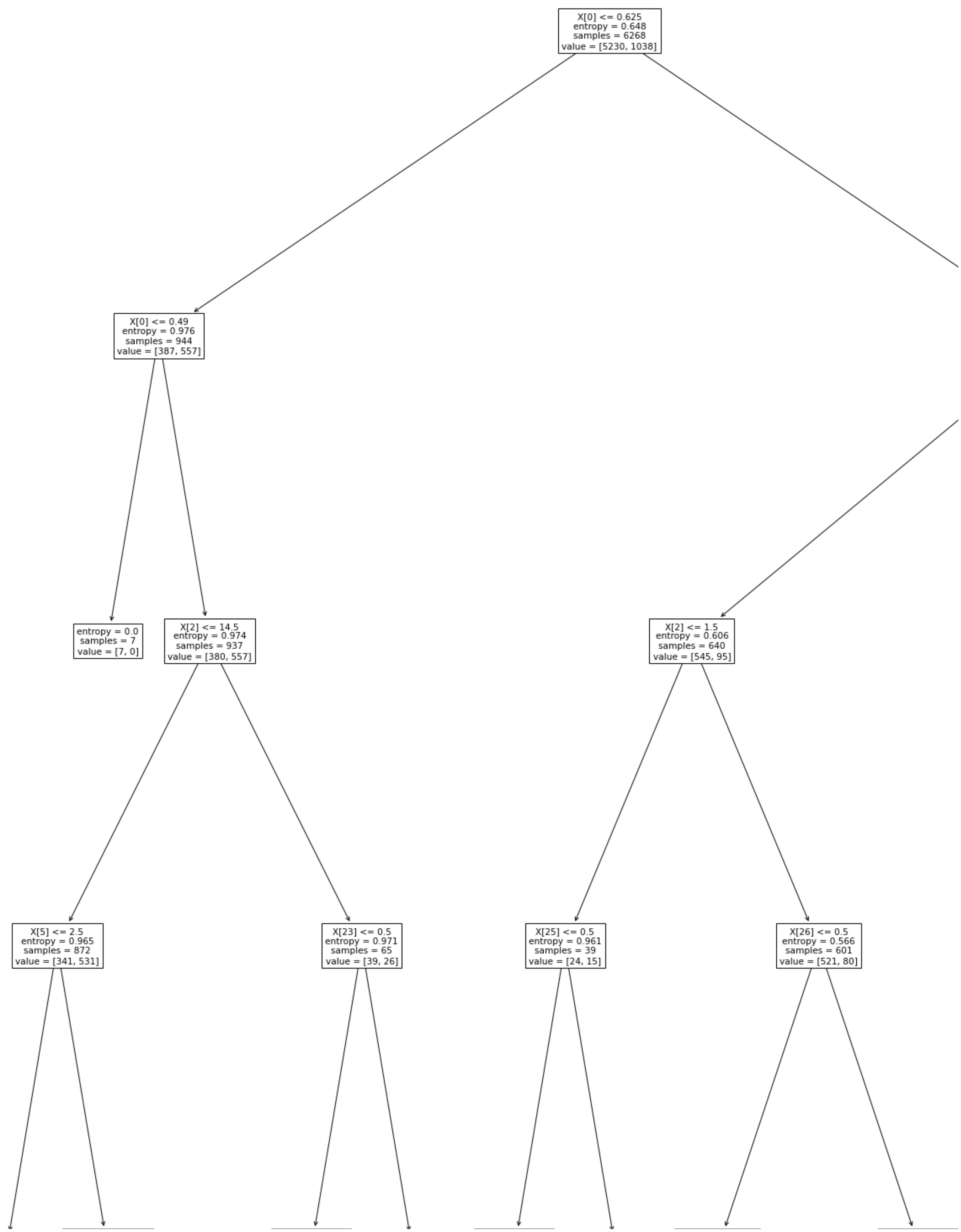
```
#Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclass
y_pred = model.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
```

```
plt.ylabel("True positive rate")
plt.show()
```



Double-click (or enter) to edit

```
#9
#Decision Tree
#Build a decision tree model using sklearns DecisionTreeClassifier.
#Use the unbalanced training set, entropy as the criterion. Try with different max_dep
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
tree_clf = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1, criterion='entropy'
tree_clf.fit(X_train, y_train)
plt.figure(figsize = (40,40))
tree.plot_tree(tree_clf)
plt.show()
```

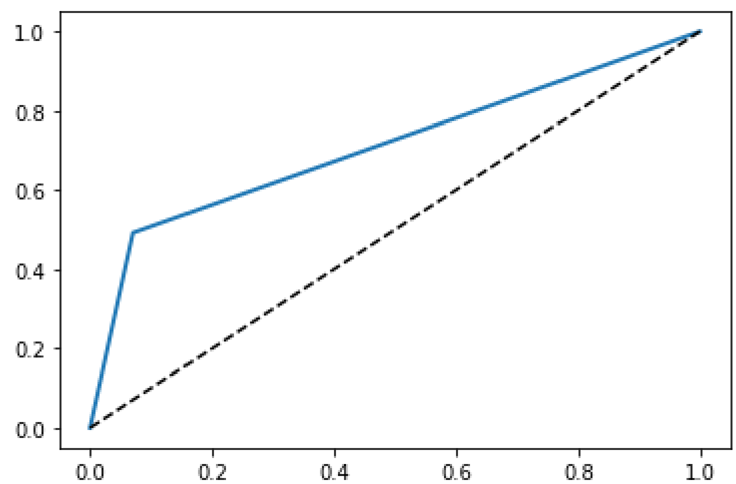



```
#After building model, test it and print the confusion matrix and classification repo
y_pred= tree_clf.predict(X_test)
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[2084  158]
 [ 226  219]]
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.92      2242
         1.0       0.58      0.49      0.53       445

    accuracy                           0.86      2687
   macro avg       0.74      0.71      0.72      2687
weighted avg       0.85      0.86      0.85      2687
```

```
y_pred = tree_clf.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')
plot_roc_curve(fpr, tpr)
```




```
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

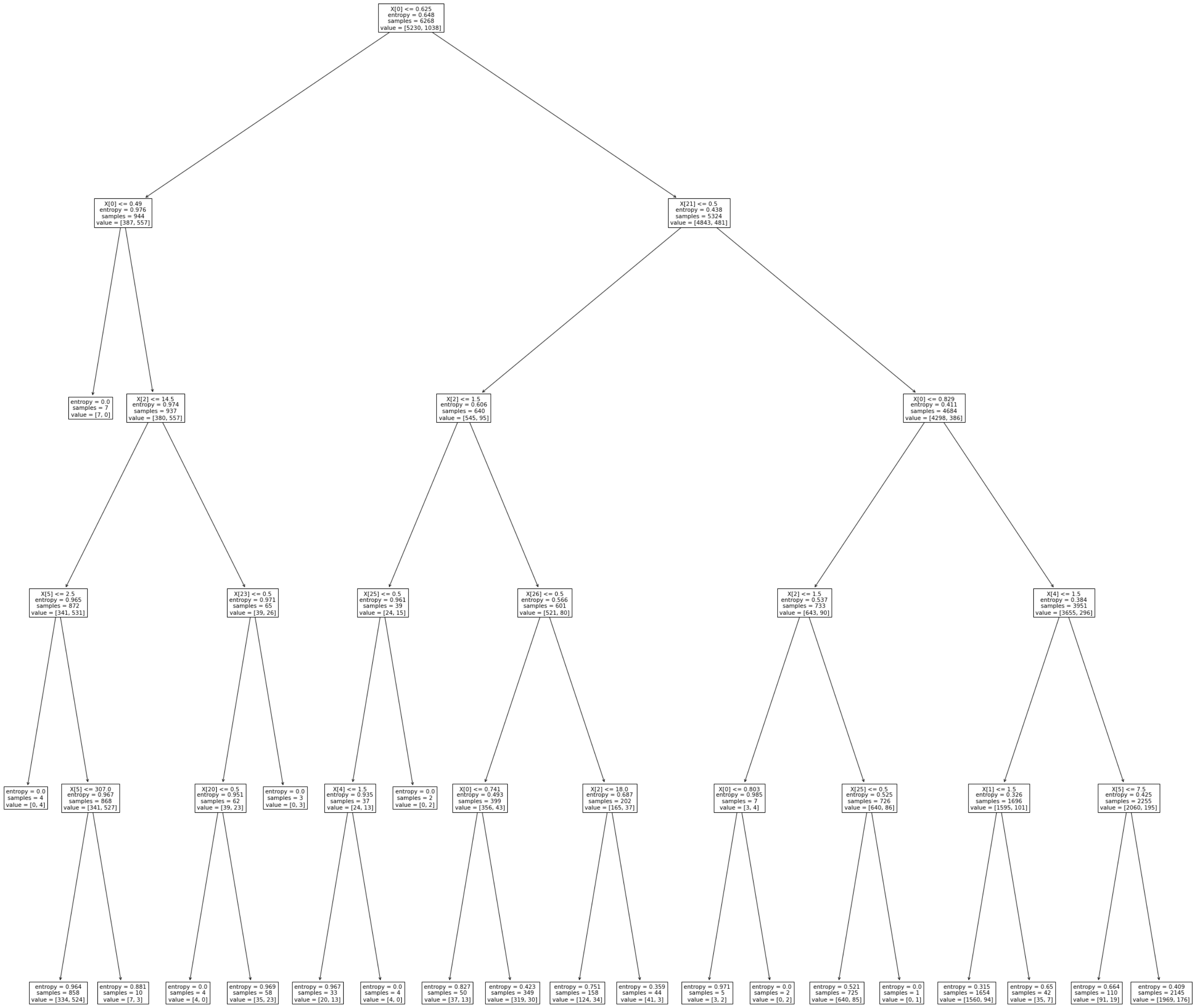
```
roc_auc_score(y_test, y_pred)
```

    0.710831019655404

```
y_new=np.ravel(y_test)
p = (y_new)!= (y_pred)
print(p.sum())
```

    384

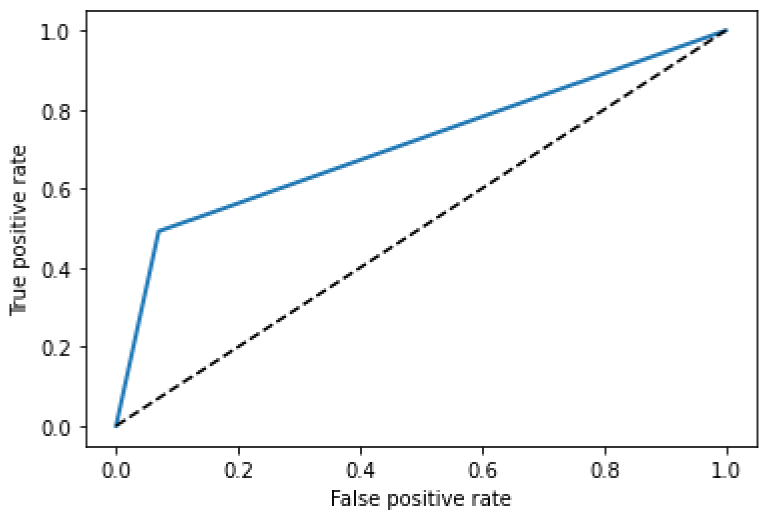
```
#Perform the same tasks as 9.1 with the balanced training set
tree_clf = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1, criterion='entropy'
tree_clf.fit(X_train, y_train)
plt.figure(figsize = (40,40))
tree.plot_tree(tree_clf)
plt.show()
```



```
y_pred= tree_clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[2085  157]
 [ 226  219]]
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.92      2242
         1.0       0.58      0.49      0.53       445

    accuracy                           0.86      2687
   macro avg       0.74      0.71      0.72      2687
weighted avg       0.85      0.86      0.85      2687
```

```
y_pred = tree_clf.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```




```
roc_auc_score(y_test, y_pred)
```

```
0.7110540348204352
```

```
y_new=np.ravel(y_test)
p = (y_new)!= (y_pred)
print(p.sum())
```

```
383
```

```
#Discuss any difference and also discuss part of the tree of 9.2
##The scores look almost identical to each other but slight differences in their numer
```

```
#Random Forest
##Use grid search to tune the max_depth, min_samples_leaf, and n_estimators
rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5, n_estimators=100,
params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
    }
    # Instantiate the grid search model
```

```
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
grid_search.fit(X_bal, y_bal)
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:926: Da
  self.best_estimator_.fit(X, y, **fit_params)
GridSearchCV(cv=4,
             estimator=RandomForestClassifier(max_depth=5, n_jobs=-1,
                                              oob_score=True, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

```
#Print the best estimator
grid_search.best_estimator_
```

```
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=25,
                       n_jobs=-1, oob_score=True, random_state=42)
```

```
#Train the model. After building the model, test it and print the confusion matrix and
#Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclass
rf = RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=25, n_jobs=
rf.fit(X_bal, y_bal)
y_pred= rf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

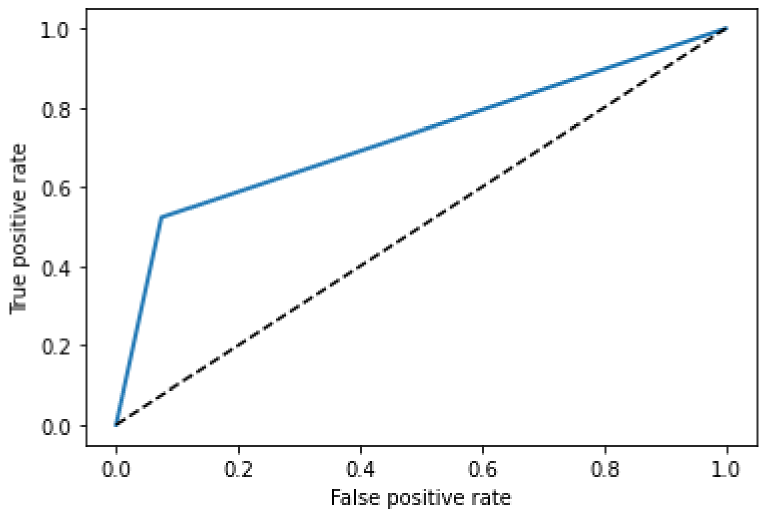
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: DataConversionWa
  after removing the cwd from sys.path.
[[2075  167]
 [ 212  233]]
              precision    recall  f1-score   support

         0.0       0.91      0.93      0.92      2242
         1.0       0.58      0.52      0.55       445

    accuracy                           0.86      2687
   macro avg       0.74      0.72      0.73      2687
weighted avg       0.85      0.86      0.86      2687
```

```
#plot ROC curve and show the AUC of ROC, and the count of the number of misclassificat
roc_auc_score(y_test, y_pred)
```

```
0.7245542202487747
```

```
y_pred = rf.predict(X_test)
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--')



#plotting
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```




```
y_new=np.ravel(y_test)
p = (y_new)!= (y_pred)
print(p.sum())
```

    379

```
#11
#Train an AdaBoostClassifier model with some manual/grid search-based parameters
#and then test it and then print the confusion matrix and classification report.
#Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclass
gd_boost= GradientBoostingClassifier(random_state=42)
params = {
   'learning_rate': [.01,0.2,0.5,0.7,0.9,1.0,1.5],
    'n_estimators': [10,25,30,50,100,200],
    "max_depth":[3,5,8]
}
grid_search = GridSearchCV(estimator=gd_boost,
                           param_grid=params,
                           cv = 3,
                           n_jobs=-1, verbose=1, scoring="accuracy")
grid_search.fit(X_bal, y_bal)
```

```
Fitting 3 folds for each of 126 candidates, totalling 378 fits
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversi
  y = column_or_1d(y, warn=True)
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=42),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.2, 0.5, 0.7, 0.9, 1.0, 1.5],
                         'max_depth': [3, 5, 8],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

```
grid_search.best_estimator_
```

```
GradientBoostingClassifier(learning_rate=0.2, max_depth=8, n_estimators=200,
                           random_state=42)
```

```
gd_boost = GradientBoostingClassifier(learning_rate=0.2, max_depth=8, n_estimators=20
gd_boost.fit(X_bal, y_bal)
y_pred= gd_boost.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

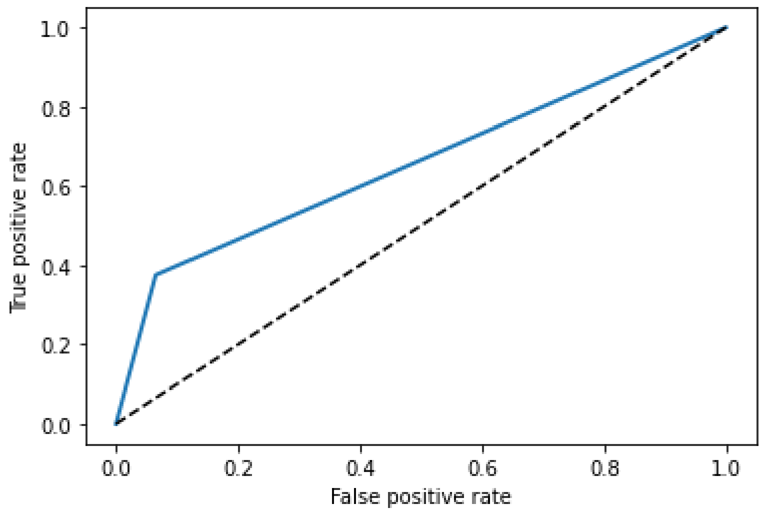
```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversi
  y = column_or_1d(y, warn=True)
[[2096  146]
 [ 278  167]]
              precision    recall  f1-score   support

         0.0       0.88      0.93      0.91      2242
         1.0       0.53      0.38      0.44       445

    accuracy                           0.84      2687
   macro avg       0.71      0.66      0.67      2687
weighted avg       0.83      0.84      0.83      2687
```

```
#roc
roc_auc_score(y_test, y_pred)
```

```
0.6550802353436438
```

```
y_pred = gd_boost.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed

#plots
plot_roc_curve(fpr, tpr)
plt.xlabel("False positive rate")
```

```
plt.ylabel("True positive rate")
plt.show()
```




```
#printing Sum
y_new=np.ravel(y_test)
p = (y_new)!= (y_pred)
print(p.sum())
```

```
424
```

```
#Finally, briefly discuss your finding such as which model could be most suitable for
##the best model suitable for this data set could be the Gaussian model based on the c
##of the true positve and the false positive rates based on the graphs on this project
##this project was a great experience for me to get comfortable with grid search and c
```

Double-click (or enter) to edit