30E03000 - Data Science for Business I (2021)

Assignment 2: Credit Risk Modeling

Import libraries

In [3]:

```
import pandas as pd

#add all necessary libraries here
import numpy as np #scientific computing
import pandas as pd #data management
import itertools

#matplotlib for plotting
import matplotlib.pyplot as plt
from matplotlib import gridspec
import matplotlib.ticker as mtick #for percentage ticks

#sklearn for modeling
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier #Decision Tree algorithm
from sklearn.model_selection import train_test_split #Data split function
from sklearn.preprocessing import LabelEncoder #OneHotEncoding
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.metrics import roc_curve, auc
from sklearn.decomposition import PCA

#Decision tree plot
import pydotplus
from IPython.display import Image
from collections import Counter
```

Import data

In [4]:

```
#import the data into a Pandas dataframe and show it
data = pd.read_csv('credit.csv')
data.head(10).style
```

Out[4]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIOTV
0	1	0	6	4	0	0	0	1
1	2	1	48	2	0	0	0	1
2	3	3	12	4	0	0	0	0
3	4	0	42	2	0	0	1	0
4	5	0	24	3	1	0	0	0
5	6	3	36	2	0	0	0	0
6	7	3	24	2	0	0	1	0
7	8	1	36	2	0	1	0	0
8	9	3	12	2	0	0	0	1
9	10	1	30	4	1	0	0	0

Data exploration

In [5]:

```
data.shape
# The dataset has 1000 rows (entries/instances/observations) and 32 columns (features).
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 32 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   OBS#             1000 non-null   int64
 1   CHK_ACCT         1000 non-null   int64
 2   DURATION         1000 non-null   int64
 3   HISTORY          1000 non-null   int64
 4   NEW_CAR          1000 non-null   int64
 5   USED_CAR         1000 non-null   int64
 6   FURNITURE        1000 non-null   int64
 7   RADIOTV          1000 non-null   int64
 8   EDUCATION        1000 non-null   int64
 9   RETRAINING       1000 non-null   int64
 10  AMOUNT           1000 non-null   int64
 11  SAV_ACCT         1000 non-null   int64
 12  EMPLOYMENT       1000 non-null   int64
 13  INSTALL_RATE     1000 non-null   int64
 14  MALE_DIV         1000 non-null   int64
 15  MALE_SINGLE      1000 non-null   int64
 16  MALE_MAR_or_WID  1000 non-null   int64
 17  COAPPLICANT      1000 non-null   int64
 18  GUARANTOR        1000 non-null   int64
 19  PRESENT_RESIDENT 1000 non-null   int64
 20  REAL_ESTATE      1000 non-null   int64
 21  PROP_UNKN_NONE   1000 non-null   int64
 22  AGE              1000 non-null   int64
 23  OTHER_INSTALL    1000 non-null   int64
 24  RENT             1000 non-null   int64
 25  OWN_RES          1000 non-null   int64
 26  NUM_CREDITS      1000 non-null   int64
 27  JOB              1000 non-null   int64
 28  NUM_DEPENDENTS   1000 non-null   int64
 29  TELEPHONE        1000 non-null   int64
 30  FOREIGN          1000 non-null   int64
 31  RESPONSE         1000 non-null   int64
dtypes: int64(32)
memory usage: 250.1 KB
```

In [6]:

```
data.describe().round().style
```

Out[6]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURN
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.(
mean	500.000000	2.000000	21.000000	3.000000	0.000000	0.000000	0.(
std	289.000000	1.000000	12.000000	1.000000	0.000000	0.000000	0.(
min	1.000000	0.000000	4.000000	0.000000	0.000000	0.000000	0.(
25%	251.000000	0.000000	12.000000	2.000000	0.000000	0.000000	0.(
50%	500.000000	1.000000	18.000000	2.000000	0.000000	0.000000	0.(
75%	750.000000	3.000000	24.000000	4.000000	0.000000	0.000000	0.(
max	1000.000000	3.000000	72.000000	4.000000	1.000000	1.000000	1.(

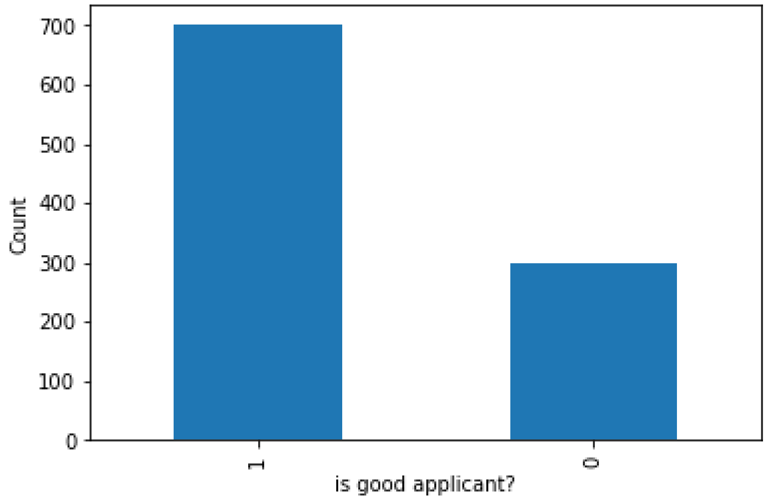
Data visualization

In [6]:

```
ax = data['RESPONSE'].value_counts().plot(kind='bar')
ax.set_xlabel('is good applicant?')
ax.set_ylabel('Count')
```

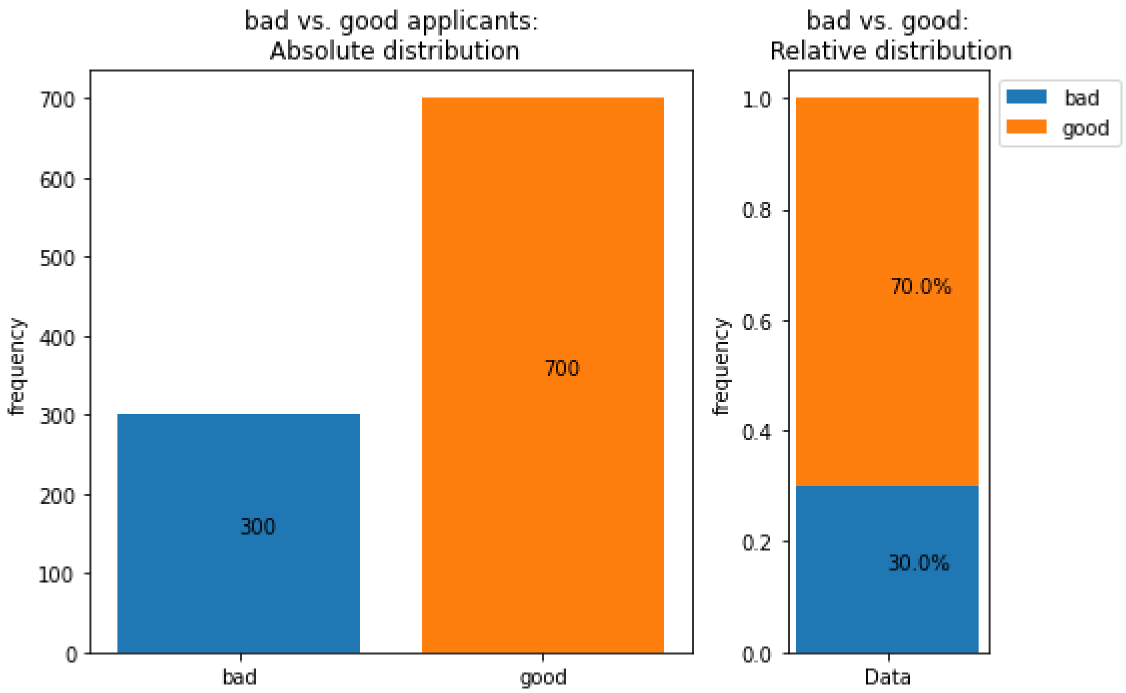
Out[6]:

```
Text(0, 0.5, 'Count')
```



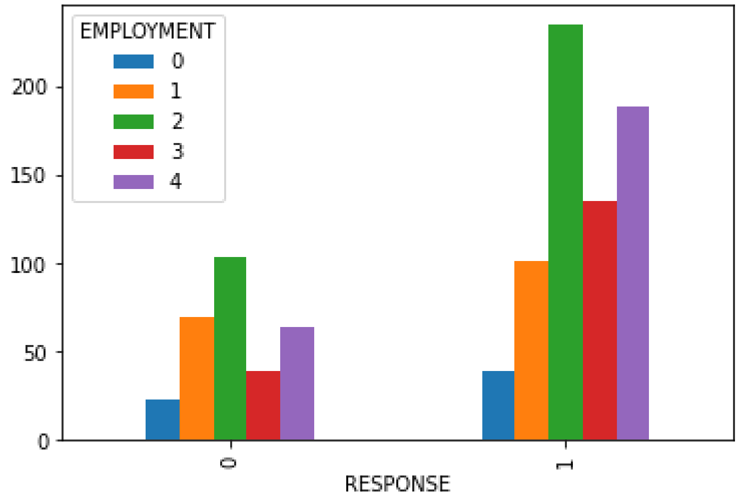
In [7]:

```
# plot fraud vs. non-fraud
keys, counts = np.unique(data.RESPONSE, return_counts=True)
counts_norm = counts/counts.sum()

fig = plt.figure(figsize=(8, 5)) #specify figure size
gs = gridspec.GridSpec(1, 2, width_ratios=[3,1]) #specify relative size of left and rig
ht plot

#Absolute values
ax0 = plt.subplot(gs[0])
ax0 = plt.bar(['bad', 'good'], counts, color=['#1f77b4','#ff7f0e']) #left bar plot
ax0 = plt.title('bad vs. good applicants:\n Absolute distribution')
ax0 = plt.ylabel('frequency')
ax0 = plt.text(['bad'], counts[0]/2, counts[0]) #add text box with count of non-fraudul
ent cases
ax0 = plt.text(['good'], counts[1]/2, counts[1]) #add text box with count of fraudulent
cases

#Normalized values
ax1 = plt.subplot(gs[1])
ax1 = plt.bar(['Data'], [counts_norm[0]], label='bad')
ax1 = plt.bar(['Data'], [counts_norm[1]], bottom=counts_norm[0], label='good')
ax1 = plt.legend(bbox_to_anchor=(1, 1))
ax1 = plt.title('bad vs. good:\n Relative distribution')
ax1 = plt.ylabel('frequency')
ax1 = plt.text(['Data'],counts_norm[0]/2, '{}%'.format((counts_norm[0]*100).round(1)))
ax1 = plt.text(['Data'],(counts_norm[1]/2)+counts_norm[0], '{}%'.format((counts_norm[1]
*100).round(1)))

plt.tight_layout()
plt.show()
```



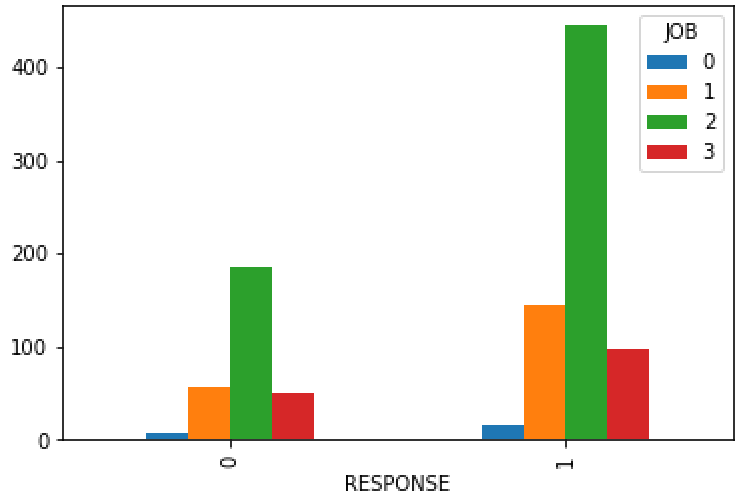
In [8]:

```
ax = data.groupby(['RESPONSE', 'EMPLOYMENT'])['EMPLOYMENT'].count().unstack().plot.bar
()
#present employment since
```



In [9]:

```
ax = data.groupby(['RESPONSE', 'JOB'])['JOB'].count().unstack().plot.bar()
# 0 : unemployed/ unskilled  - non-resident
# 1 : unskilled - resident
# 2 : skilled employee / official
# 3 : management/ self-employed/highly qualified employee/ officer
```



Data preprocessing

In [10]:

```
data.corr().style
```

Out[10]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR
OBS#	1.000000	0.005852	0.030788	-0.011691	0.057692	0.008245
CHK_ACCT	0.005852	1.000000	-0.072013	0.192191	-0.069559	0.064303
DURATION	0.030788	-0.072013	1.000000	-0.077186	-0.109999	0.144939
HISTORY	-0.011691	0.192191	-0.077186	1.000000	0.042480	0.039096
NEW_CAR	0.057692	-0.069559	-0.109999	0.042480	1.000000	-0.187291
USED_CAR	0.008245	0.064303	0.144939	0.039096	-0.187291	1.000000
FURNITURE	-0.003846	-0.098016	-0.062804	-0.025539	-0.259831	-0.159301
RADIOTV	-0.017483	0.110632	-0.044319	0.021396	-0.344672	-0.211317
EDUCATION	-0.025065	0.007848	0.003750	0.054039	-0.126799	-0.077740
RETRAINING	-0.018066	0.021587	0.164113	-0.090091	-0.181149	-0.111062
AMOUNT	0.013488	-0.042705	0.624984	-0.059905	-0.040793	0.252101
SAV_ACCT	0.003730	0.222867	0.047661	0.039058	-0.002348	0.112880
EMPLOYMENT	-0.020078	0.106339	0.057381	0.138225	-0.021232	0.039358
INSTALL_RATE	0.010076	-0.005280	0.074749	0.044375	-0.045801	-0.094797
MALE_DIV	0.008504	-0.050555	0.006415	-0.009536	-0.018424	-0.032455
MALE_SINGLE	-0.008311	0.052436	0.121889	0.086008	0.027374	0.089610
MALE_MAR_or_WID	0.005190	-0.011241	-0.084418	-0.026015	-0.012487	-0.039567
COAPPLICANT	0.018230	-0.050780	0.029698	0.007710	0.004836	-0.053474
GUARANTOR	-0.004197	-0.114673	-0.039594	-0.047179	-0.012426	-0.034910
PRESENT_RESIDENT	0.023697	-0.042234	0.034067	0.063198	0.019848	0.107257
REAL_ESTATE	-0.035544	0.035865	-0.242586	0.045799	0.042056	-0.131941
PROP_UNKN_NONE	-0.015279	-0.074624	0.212838	-0.025412	0.025940	0.128863
AGE	-0.010096	0.059751	-0.036136	0.147086	0.075044	0.050858
OTHER_INSTALL	-0.004149	-0.043593	0.067602	-0.121950	-0.027462	-0.009791
RENT	0.025442	-0.091897	-0.064417	-0.102540	-0.011620	0.039160
OWN_RES	-0.013244	0.129434	-0.075169	0.100905	-0.009618	-0.141375
NUM_CREDITS	0.022838	0.076005	-0.011284	0.437066	0.035845	-0.005248
JOB	-0.027345	0.040663	0.210910	0.010350	-0.088711	0.180730
NUM_DEPENDENTS	0.026662	-0.014145	-0.023834	0.011550	0.102663	0.054862
TELEPHONE	-0.007829	0.066296	0.164718	0.052370	-0.036275	0.136693
FOREIGN	-0.018177	-0.026758	-0.138196	0.013873	0.154436	-0.031564
RESPONSE	-0.034606	0.350847	-0.214927	0.228785	-0.096900	0.099791

In [9]:

```
# X, y = data[['CHK_ACCT', 'DURATION', 'HISTORY','NEW_CAR','USED_CAR','RADIOTV','AMOUN
T',
#               'SAV_ACCT','EMPLOYMENT','REAL_ESTATE','PROP_UNKN_NONE','OTHER_INSTAL
L','OWN_RES']], data['RESPONSE'] #define feature matrix X and labels y
# X.head()

X, y = data.loc[:, data.columns != 'RESPONSE'], data['RESPONSE'] #define feature matrix
X and labels y
```

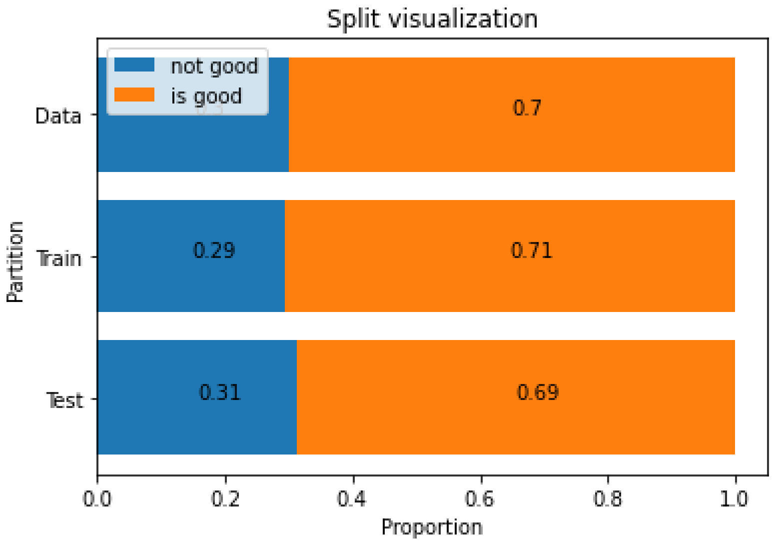
Data split

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
= 1234) #split data 70:30
```

In [11]:

```
train_dist = y_train.value_counts() / len(y_train) #normalize absolute count values for plotting
test_dist = y_test.value_counts() / len(y_test)
data_dist = data['RESPONSE'].value_counts() / len(data)

fig, ax = plt.subplots()

ax.barh(['Test','Train','Data'], [test_dist[0], train_dist[0], data_dist[0]], color='#1f77b4', label='not good')
ax.barh(['Test','Train','Data'], [test_dist[1], train_dist[1], data_dist[1]], left=[test_dist[0], train_dist[0], data_dist[0]], color='#ff7f0e', label='is good')
ax.set_title('Split visualization')
ax.legend(loc='upper left')
plt.xlabel('Proportion')
plt.ylabel('Partition')

#plot bar values
for part, a, b in zip(['Test', 'Train','Data'], [test_dist[0], train_dist[0], data_dist[0]], [test_dist[1], train_dist[1], data_dist[1]]):
    plt.text(a/2, part, str(np.round(a, 2)))
    plt.text(b/2+a, part, str(np.round(b, 2)));
```



Build an (unbalanced) Decision Tree model

In [14]:

```
#Define Decision tree classifier with some default parameters
clf = tree.DecisionTreeClassifier(criterion = "gini", random_state = 100,
                               max_depth=3, min_samples_leaf=3)

#Fit the training data
clf.fit(X_train, y_train) #what do we need here?
```

Out[14]:

```
DecisionTreeClassifier(max_depth=3, min_samples_leaf=3, random_state=100)
```

In [15]:

```
#Use classifier to predict labels
y_pred = clf.predict(X_test) #what do we need here?
```

In [16]:

```
y_pred
```

Out[16]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1])
```

In [17]:

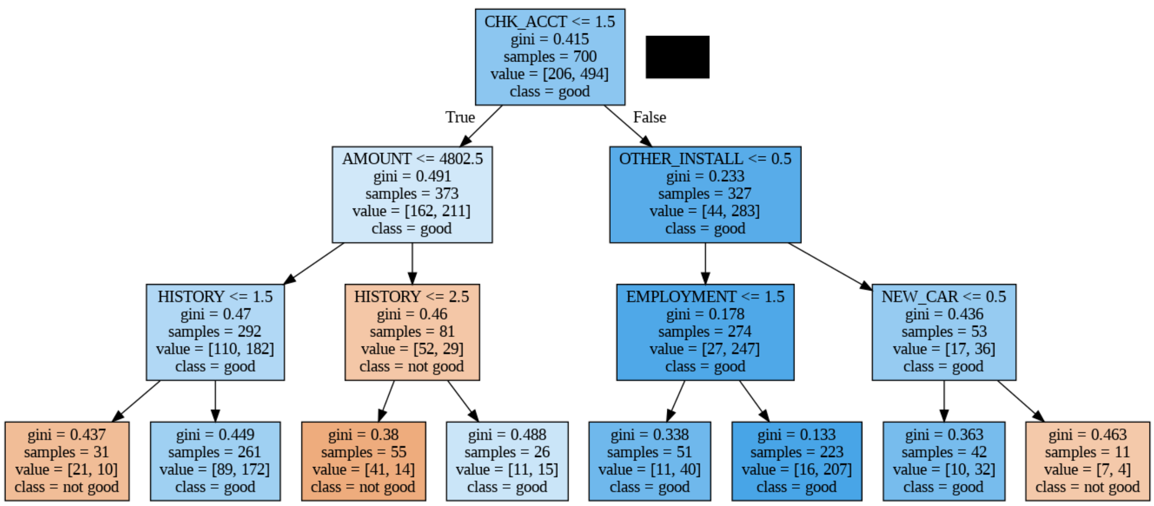
```
#probabilities
y_pred_probs = clf.predict_proba(X_test)
```

In [18]:

```
'''
The graphviz library is used to visualize the tree.
'''

#Decision tree plot
import pydotplus
from IPython.display import Image

# Create DOT data
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=X_train.columns,
                                class_names=['not good', 'good'], filled=True) #or use
 y_train.unique()

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png())

# Create PNG
#graph.write_png("clf.png") #uncomment this line to save the plot as a .png file
```

```
Fontconfig error: Cannot load default config file
```

Out[18]:



Rebalancing with SMOTE

In [12]:

```
smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X_train, y_train) #ONLY APPLIED TO TRAINING!!!
```

In [13]:

```
X_sm
```

Out[13]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIOT
0	97	3	12	4	0	0	0	
1	793	3	6	4	0	0	1	
2	219	0	24	2	0	0	1	
3	968	3	15	2	0	0	0	
4	171	0	15	0	1	0	0	
...	...	...	...	...	...	...	...	
983	194	1	53	0	0	0	0	
984	598	0	12	1	0	0	0	
985	289	0	13	0	0	0	0	
986	741	2	12	2	0	0	0	
987	273	1	24	2	0	0	0	

988 rows × 31 columns

In [14]:

```
X_train
```

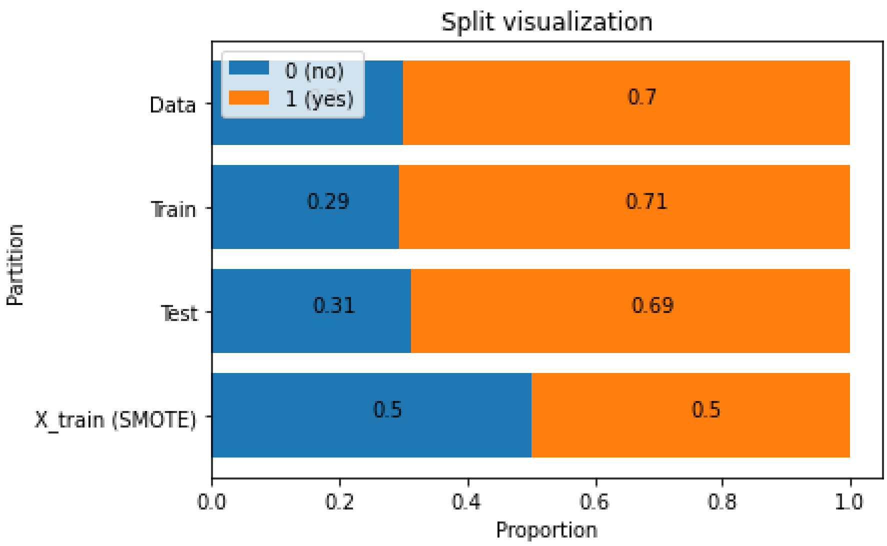
Out[14]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIOT
96	97	3	12	4	0	0	0	
792	793	3	6	4	0	0	1	
218	219	0	24	2	0	0	1	
967	968	3	15	2	0	0	0	
170	171	0	15	0	1	0	0	
...	...	...	...	...	...	...	...	
204	205	3	12	4	1	0	0	
53	54	3	18	2	0	1	0	
294	295	3	48	4	0	0	0	
723	724	1	9	2	0	0	0	
815	816	1	36	3	1	0	0	

700 rows × 31 columns

In [20]:

```
train_dist = y_train.value_counts() / len(y_train) #normalize absolute count values for
plotting
test_dist = y_test.value_counts() / len(y_test)
data_dist = y.value_counts() / len(y)
smote_dist = pd.Series(y_sm).value_counts() / len(pd.Series(y_sm))

fig, ax = plt.subplots()

ax.barh(['X_train (SMOTE)','Test','Train','Data'], [smote_dist[0], test_dist[0], train_
dist[0], data_dist[0]], color='#1f77b4', label='0 (no)')
ax.barh(['X_train (SMOTE)','Test','Train','Data'], [smote_dist[1], test_dist[1], train_
dist[1], data_dist[1]], left=[smote_dist[0], test_dist[0], train_dist[0], data_dist[0
]], color='#ff7f0e', label='1 (yes)')
ax.set_title('Split visualization')
ax.legend(loc='upper left')
plt.xlabel('Proportion')
plt.ylabel('Partition')

#plot bar values
for part, a, b in zip(['X_train (SMOTE)', 'Test', 'Train','Data'], [smote_dist[0], test
_dist[0], train_dist[0], data_dist[0]], [smote_dist[1], test_dist[1], train_dist[1], da
ta_dist[1]]):
    plt.text(a/2, part, str(np.round(a, 2)))
    plt.text(b/2+a, part, str(np.round(b, 2)));
```



Build a balanced Decision Tree model

In [21]:

```
clf_b = tree.DecisionTreeClassifier(criterion = "gini", random_state = 100,
                               max_depth=3, min_samples_leaf=3)

#Fit the training data
clf_b.fit(X_sm, y_sm) #what do we need here?
```

Out[21]:

```
DecisionTreeClassifier(max_depth=3, min_samples_leaf=3, random_state=100)
```

In [22]:

```
#Use classifier to predict labels
y_pred_b = clf_b.predict(X_test) #what do we need here?
```

In [23]:

```
y_pred_b
```

Out[23]:

```
array([1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1])
```

In [24]:

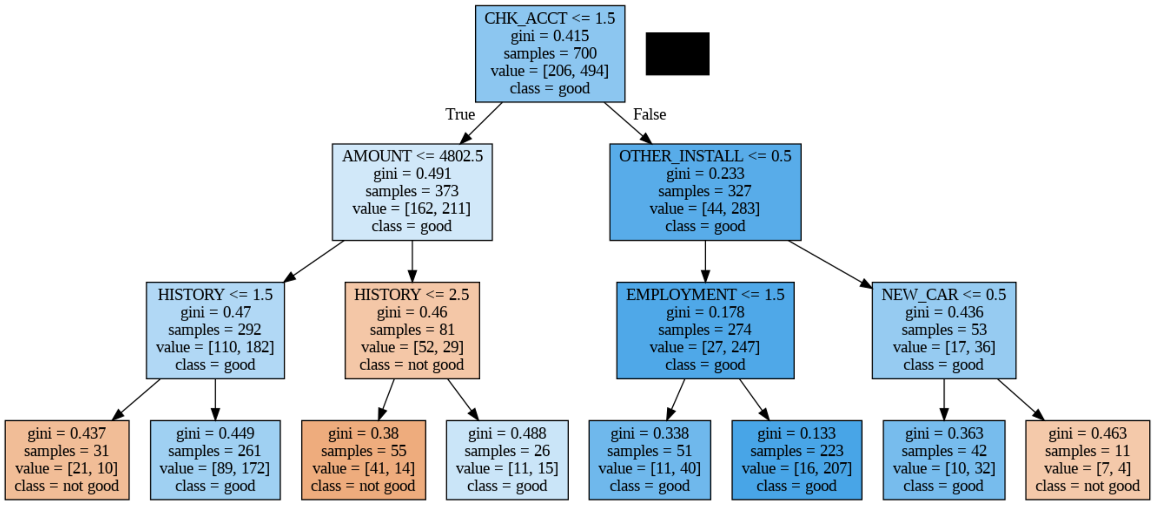
```
#probabilities
y_pred_probs_b = clf_b.predict_proba(X_test)
```

In [25]:

```
'''
The graphviz library is used to visualize the tree.
'''

#Decision tree plot
import pydotplus
from IPython.display import Image

# Create DOT data
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=X_train.columns,
                                class_names=['not good', 'good'], filled=True) #or use
 y_train.unique()

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png())

# Create PNG
#graph.write_png("clf.png") #uncomment this line to save the plot as a .png file
```

```
Fontconfig error: Cannot load default config file
```

Out[25]:



Model evaluation

1. Confusion Matrix

2. ROC and AUC

3. Expected value framework (Excel)

In [26]:

```
print ("Accuracy is: ", (accuracy_score(y_test,y_pred)*100).round(2))
```

```
Accuracy is:  69.67
```

In [27]:

```
print ("Accuracy is: ", (accuracy_score(y_test,y_pred_b)*100).round(2))
```

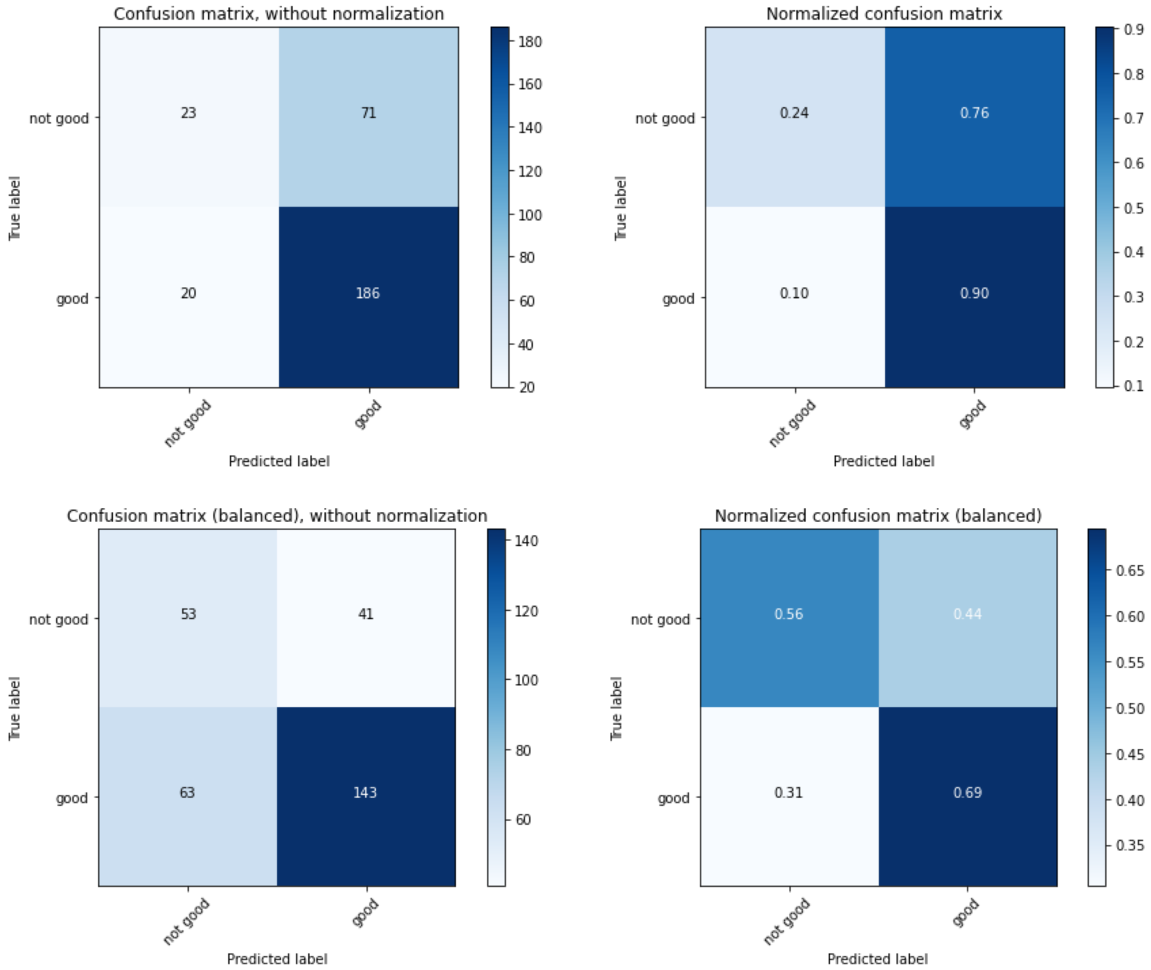
```
Accuracy is:  62.67
```

In [58]:

```
#confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylim([1.5, -0.5]) #added to fix a bug that causes the matrix to be squished
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [59]:

```
# Compute confusion matrix
class_names = ['not good', 'good']
cnf_matrix_original = confusion_matrix(y_test, y_pred)
cnf_matrix_balanced = confusion_matrix(y_test, y_pred_b)
np.set_printoptions(precision=2)
##imbalanced
# Plot non-normalized confusion matrix
plt.figure(figsize=(13, 5))
plt.subplot(121)
plot_confusion_matrix(cnf_matrix_original, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.subplot(122)
plot_confusion_matrix(cnf_matrix_original, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
##balanced
# Plot non-normalized confusion matrix
plt.figure(figsize=(13, 5))
plt.subplot(121)
plot_confusion_matrix(cnf_matrix_balanced, classes=class_names,
                      title='Confusion matrix (balanced), without normalization')

# Plot normalized confusion matrix
plt.subplot(122)
plot_confusion_matrix(cnf_matrix_balanced, classes=class_names, normalize=True,
                      title='Normalized confusion matrix (balanced)')


plt.show()
```



In [60]:

```
#AUC and ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Testing: " + str(roc_auc))
```

AUC score on Testing: 0.7089960751910762

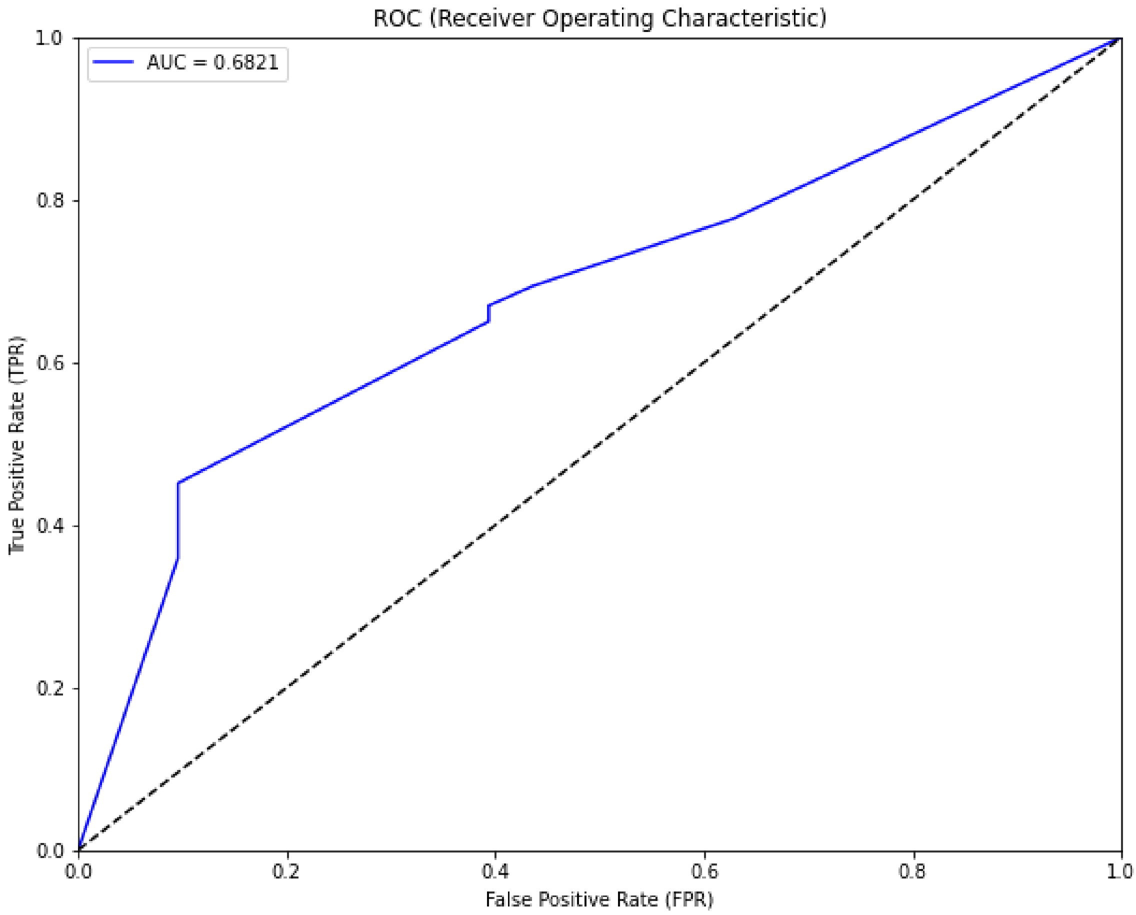
In [61]:

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs_b[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Testing: " + str(roc_auc))
```

AUC score on Testing: 0.682064656062797

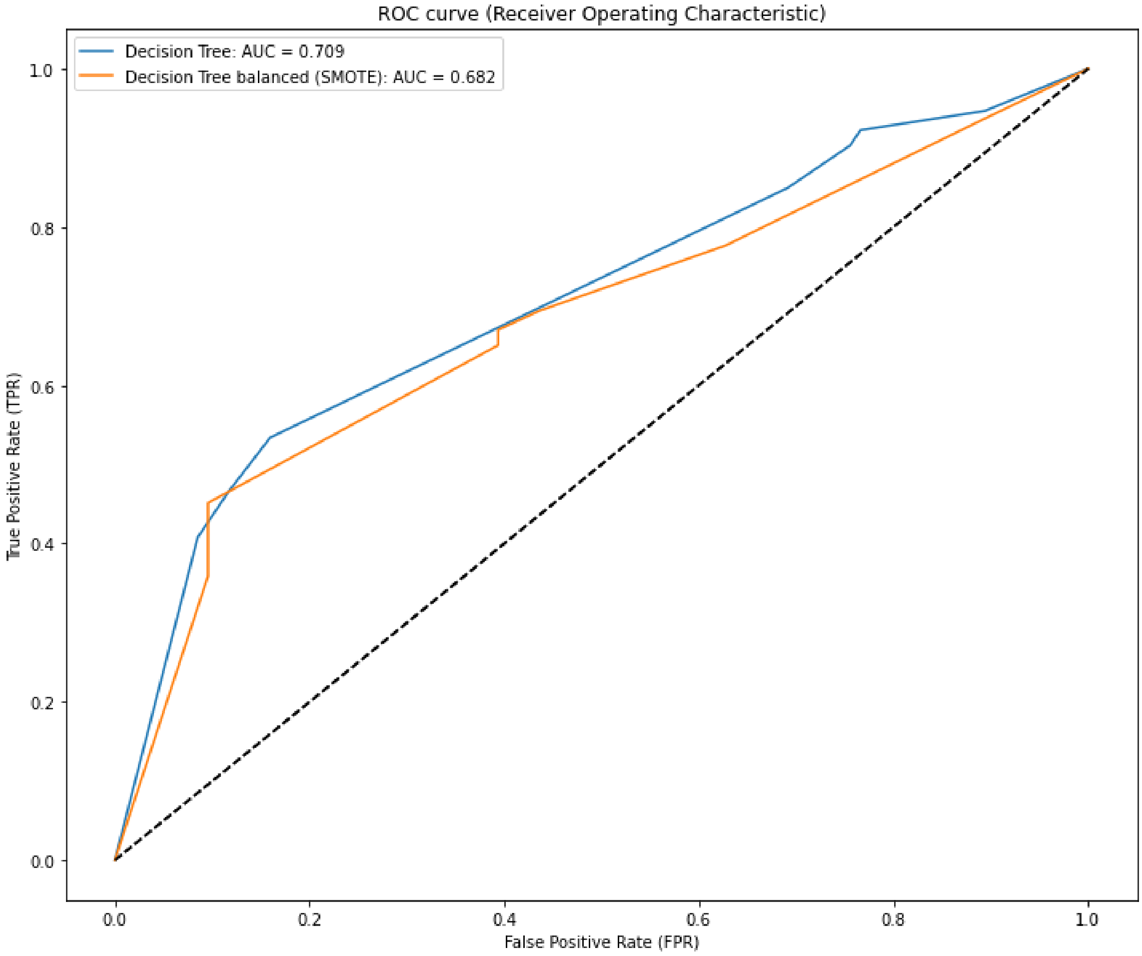
In [62]:

```
fig, axs = plt.subplots(1,1, figsize=(10,8))

plt.title('ROC (Receiver Operating Characteristic)')
plt.plot(fpr, tpr, 'b', label='AUC = %0.4f'% roc_auc)
plt.legend(loc='best')
plt.plot([0,1],[0,1],color='black', linestyle='--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)');
```



In [63]:

```
plt.figure(figsize=(12,10))

for test, pred, name in zip([y_test, y_test], [y_pred_probs[:,1], y_pred_probs_b[:,1]],
['Decision Tree', 'Decision Tree balanced (SMOTE)']):
    fpr, tpr, _ = roc_curve(test, pred)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='{}: AUC = {}'.format(name, round(roc_auc, 3)))
    plt.legend(loc='best')
    plt.plot([0,1],[0,1],color='black', linestyle='--')

plt.title('ROC curve (Receiver Operating Characteristic)')
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')

plt.show()
```



After comparing two modelsin Excel, I would suggest Decision tree with balanced data. Based on calculation, the expected benefit witht est set priors is $27, whichishigherthantheotheronewith$5.67