Bart G. Rando
Pokuong lao
Abhishek Rajput
Christian Armendariz
Professor Mohammad Pourhomayoun
Data Science CS4661-01
15 December 2022

Final Project Report

Our goal for this project was to train a model to accurately predict whether a credit card application should or should not be approved based on data previously collected by lenders. The idea is that lenders have access to vast amounts of user data based on personal information and the corresponding credit histories on the loans they provide. Using data they already have access to, they could use machine learning to predict which applicants are safe and risky accurately. With the constant collection of a large amount of new lending data daily, these models could be trained to extreme accuracy given enough time and data.

The dataset came from Kaggle and consisted of actual application record data and credit history data stripped of any personally identifiable information. The data was much more challenging to work with than we initially realized. It was broken down into two sets of records: the application and credit records. The application records consisted of background information submitted on applications filed with lenders. The credit records consisted of an ongoing history of the status of loans issued to customers. A customer ID linked the two datasets but was not balanced or corresponded evenly. By definition, customers and ongoing customer records are a one-to-many relationship. This is not ideal for training a model of this type since any specific customer record might not accurately represent the particular customer. Still, the bulk of the

records for each customer provides a more accurate context. Aside from this issue, there also existed the possibility that application records with no credit records on file could exist. As we learned later in the project, there could also be credit records with no corresponding application on file. Since these datasets were a small sample of actual data, either one could be incomplete compared to the other.

The application data was primarily categorical, with only age, number of children, and income being continuous values. The credit records only had customer ID, the number of months from the present, and payment status as elements. Payment status was already broken down into categories of 1-5, each representing several months past due, and C for current and X for over-extended accounts. We first converted the character values into numerical ones to deal with this issue. We chose zero for current and 6 for over-extended. We then used the "groupby" method and the "mean" method to get a new feature column of the average of all the combined status records for each unique customer ID. We then wrote a function that flagged any averages above 1 as risky. In this way, we ensured that anyone behind for over 30 days would be labeled risky. This can be adjusted simply depending on what a lender would consider risky and allows for comparisons against decimal values since we took the averages. For instance, the risk threshold could be set to 1.5, allowing someone who got behind once a chance to average out over time.

In contrast, consistently late individuals would retain a high average over a longer time span. After the groupby averaging and applying the risk assessment method, we were left with a two-feature dataset of credit records, with no repeated rows and a clear categorical label to use in our modeling. We then performed an inner join merge with our application records. By using inner join on ID, we ensured that only data corresponding to our chosen label would be utilized

in our modeling. Finally, we were left with a complete data set of features and a label, and all that was left was to clean up further and polish our features and test our models for accuracy.

At this point, we needed to remove any unnecessary information from the dataset. There were many “Flag” features that were meant to signify if the person had submitted various information on their application, such as phone, email, work phone, etc. These provided no usable real-world data and were only placeholders for removed identifying information. For the categorical values in the dataset, we utilized the label encoder from sclearn in a loop to convert string entries into numerical values. We also filled in missing information for occupation type with other”. Then we used the min-max scaler method for preprocessing our dataset to normalize the continuous values.

To get a better sense of the data, we utilized several data visualization techniques, including scatter plots, histograms, heatmaps, and even rendered a very detailed graph of the progressive steps that our decision tree classifier was taking. These tools were invaluable in creating this project. We have their place at strategic points within our notebook so that we can visually monitor the changes in the dataset as we went along. We wrote an adaptive function to display scatterplots; this function breaks scatter plots down into a grid 3 columns wide and grows rows as needed depending on the number of features contained within the dataset. Although the scatterplots don’t provide much helpful information, they provide a quick and easy indicator of changes made. This was very helpful in preparing the data and debugging any potential errors in the code.

We utilized histograms and a heatmap for more detailed information about the data. These provided a much clearer sense of the accurate contents of each feature and helped guide us

along the data-cleaning process. These were placed before the cells containing the model training, so we could know exactly what data we were feeding the models.

Finally, we wrote a loop to iterate across several different classifiers and print out each one's accuracy and the corresponding ROC graphs. The classifiers we included in our project are logistic regression, random forest, decision tree, gaussian, and KNN. The accuracy rates varied between 0.6884 and 0.7125, the least accurate being gaussian and the most accurate being random forest. Through trial and error, we attempted to make adjustments and modifications to our code to increase the accuracy of our results, but this was the best we could do with our current limited background in data science and the inherent limitations of the datasets we were working with. Results

Classifier	Accuracy	AUC
Logistic Regression	0.6885414523760544	0.5081639518397293
Random Forest	0.712542000960022	0.7212432768493737
Decision Tree	0.696221627922924	0.6429654269542432
Gaussian	0.6884728793801002	0.5042535643491803
K Neighbors	0.7021189055749846	0.6780217620306282

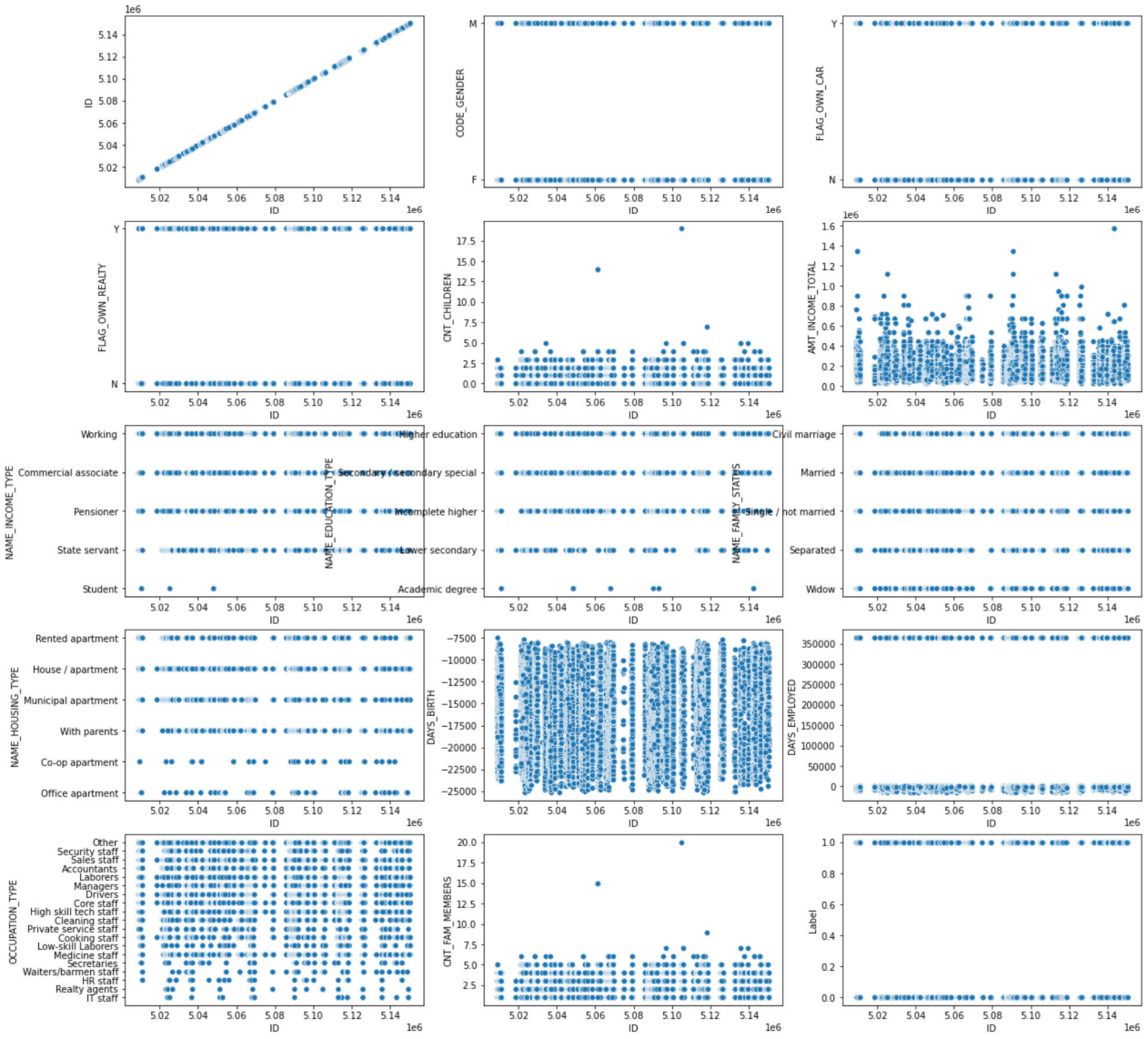
In [274…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#read application records
applicationRecord = pd.read_csv("./kaggle/input/credit-card-approval-prediction/application_recor
#deleted duplicate application, most recent kept
applicationRecord= applicationRecord.drop_duplicates('ID', keep='last')

#read Credit Records
creditRecord = pd.read_csv("./kaggle/input/credit-card-approval-prediction/credit_record.csv")
```

In [275…
```
#replaced string values as numerical values
creditRecord['STATUS'].replace(['C','X'],[0,6], inplace= True)
#converted column to type int
creditRecord['STATUS']=creditRecord['STATUS'].astype(int)

#removed records with same ID number by averaging the values
#Status ends up as a float rather than an int
avrStat=creditRecord.groupby(['ID'])['STATUS'].mean()

#risk assestment Method
def solve(status):
    if status > 1: #threshold for Risk Assesment can be adjusted as needed
        label=1
    else :
        label=0

    return label

#merged application and credit records with an inner join to eliminate non-coresponding entries
data=pd.merge(applicationRecord,avrStat,how="inner",on="ID")

#applied Risk assestment Method to Data
#renamed category as Label
data['Label']=data['STATUS'].apply(solve)
```

In [276…
```
#This method made displaying scatter plots easier
#It adapts to the number of features so i didn't
#have to adjust it mannually each time
#not the most ellegant, but works well

def scatter_plots(dataset):
    feature_cols=dataset.columns
    n=(len(feature_cols)//3)
    if (len(feature_cols)%3)!=0:
        n+=1
    fig, ax= plt.subplots(nrows= n, ncols = 3, figsize= (20,20))
    i=0
    j=0
    for feature in feature_cols :
        sns.scatterplot(x='ID', y=feature, data=dataset, ax=ax[i][j])
        j=j+1
        if j>2:
            j=0
            i=i+1
```

In [277…
```
#removed non-useful data
data.drop(['FLAG_MOBIL','FLAG_WORK_PHONE','FLAG_PHONE','FLAG_EMAIL','STATUS'], inplace = True, ax

#filled missing data from this feature
data['OCCUPATION_TYPE'].fillna('Other',inplace=True)
```

In [278…
```
#displayed data so that I could observe changes throughout the project
scatter_plots(data)
```



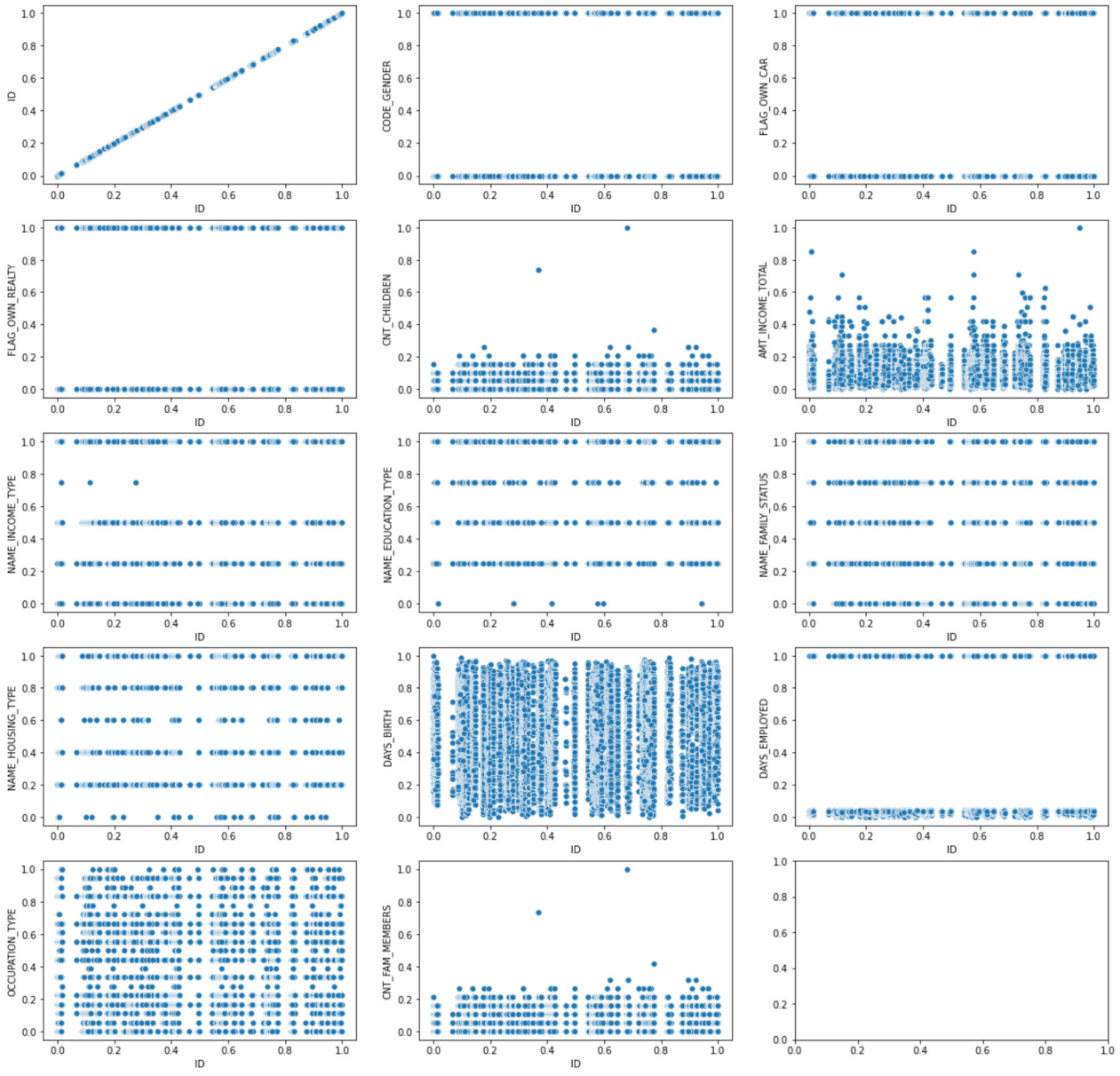
In [279…
```
#converted categorical string values in numberical values
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
for i in data:
    if data[i].dtypes=='object':
        data[i] = LE.fit_transform(data[i])
```

In [280…
```
#Normalized the features
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

x= data.loc[:, data.columns!='Label']
y= data['Label']
scaler=MinMaxScaler()

scaler.fit(x)
scaler.transform(x)
feature_scale = [feature for feature in data.columns if feature!='Label']
data = pd.concat([data['Label'].reset_index(drop=True),pd.DataFrame(scaler.transform(x), columns=
x= data.loc[:, data.columns!='Label']
```

```
#split the data into training and testing data
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4,random_state = 42)
```

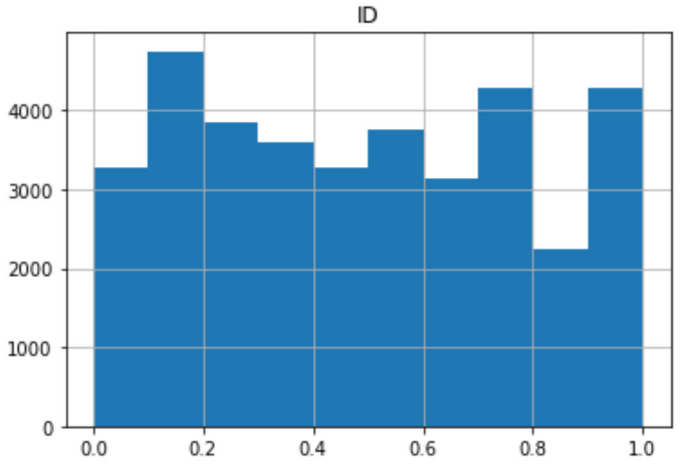
In [281…

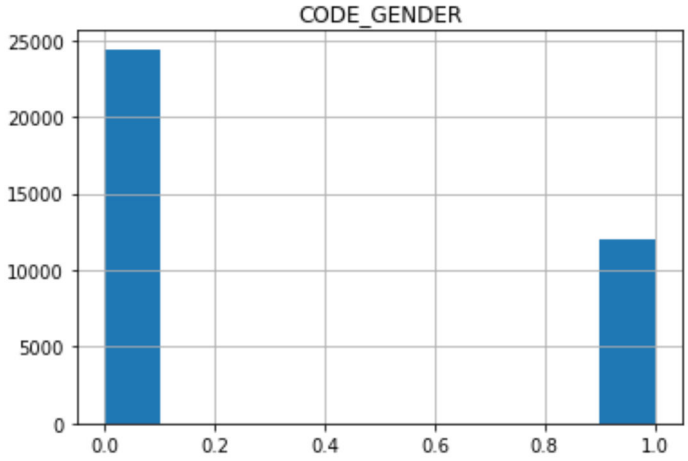
```
#displayed data so that I could observe changes throughout the project
scatter_plots(x)
```

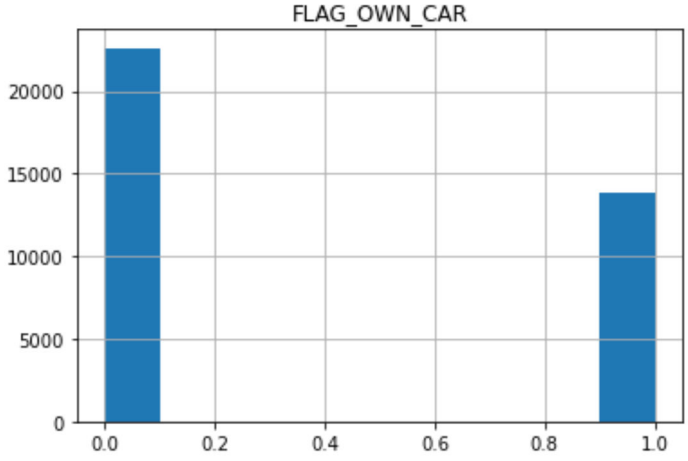


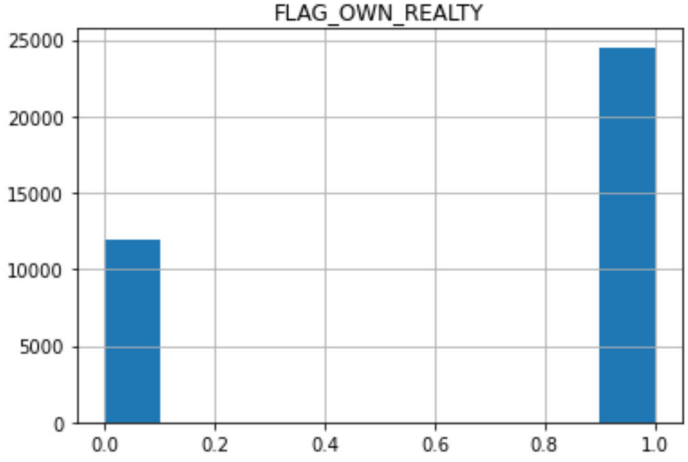
In [282…

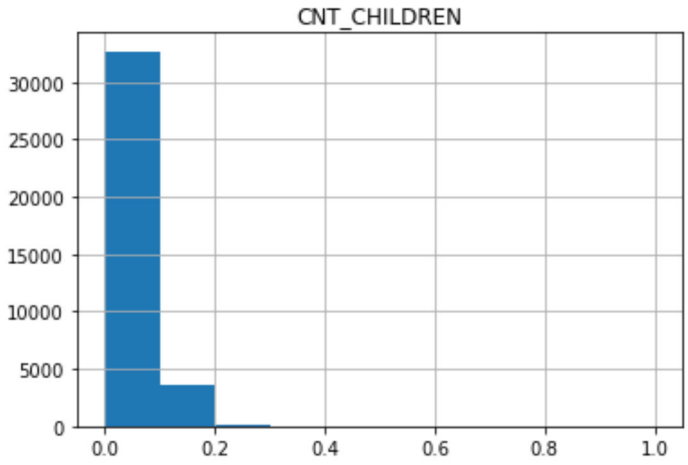
```
#displayed histograms so that I could visualize the features better
feature_cols=x.columns
for feature in feature_cols :
    x.hist(column=feature)
```

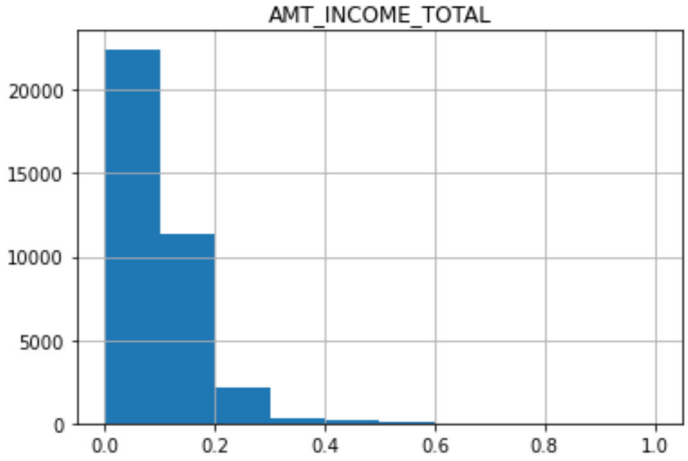

ID
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

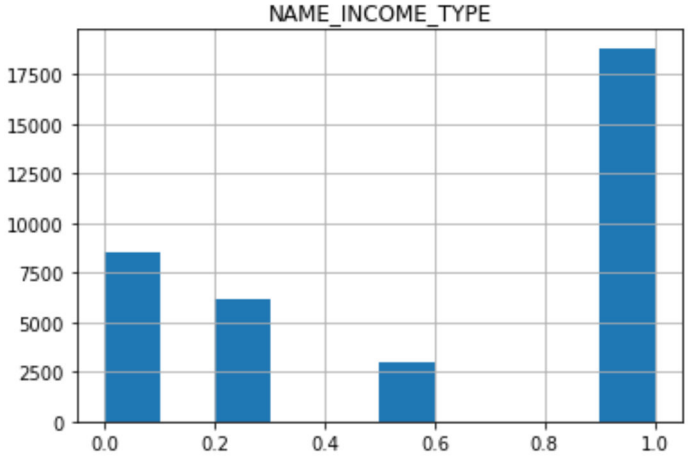

CODE_GENDER
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

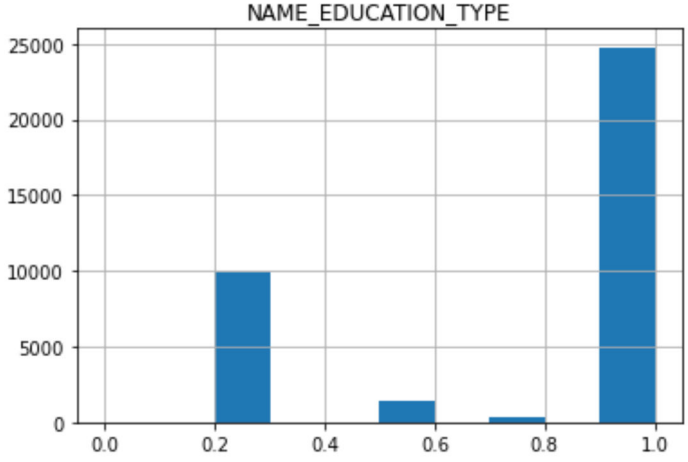

FLAG_OWN_CAR
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

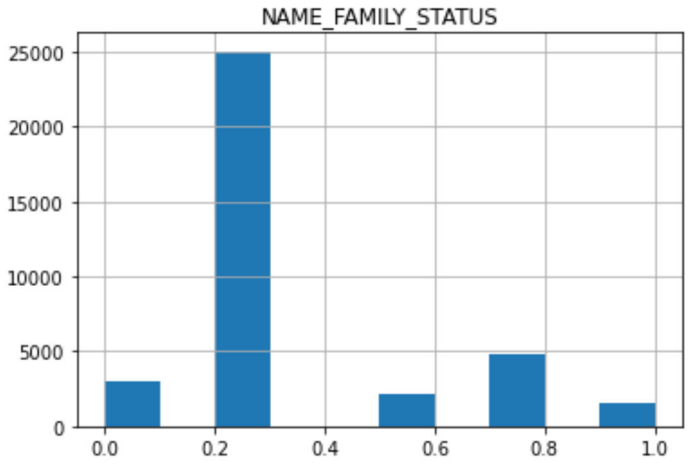

FLAG_OWN_REALTY
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

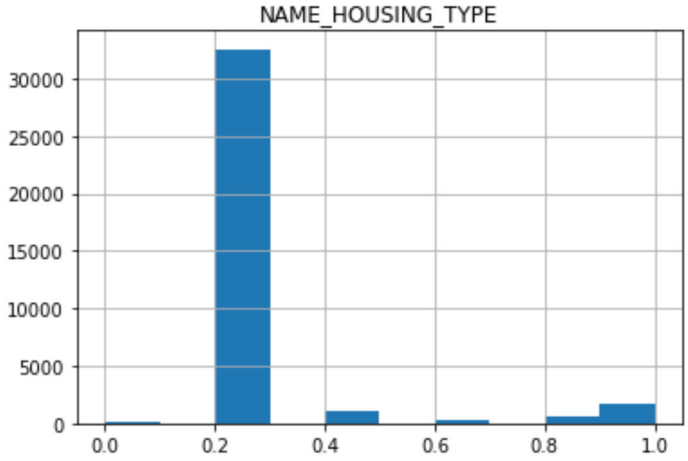

CNT_CHILDREN
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

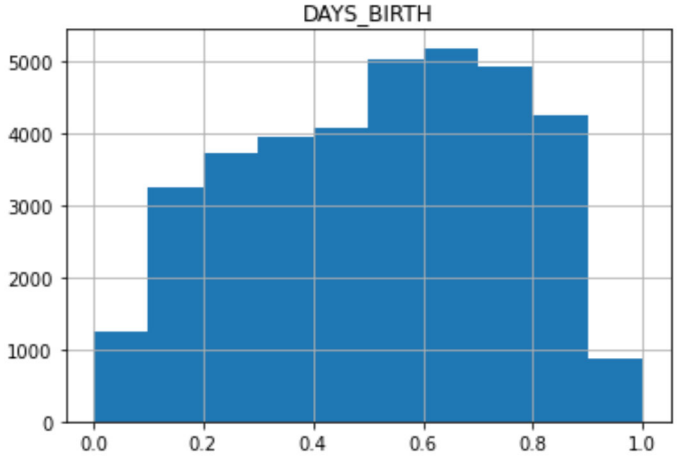

AMT_INCOME_TOTAL
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

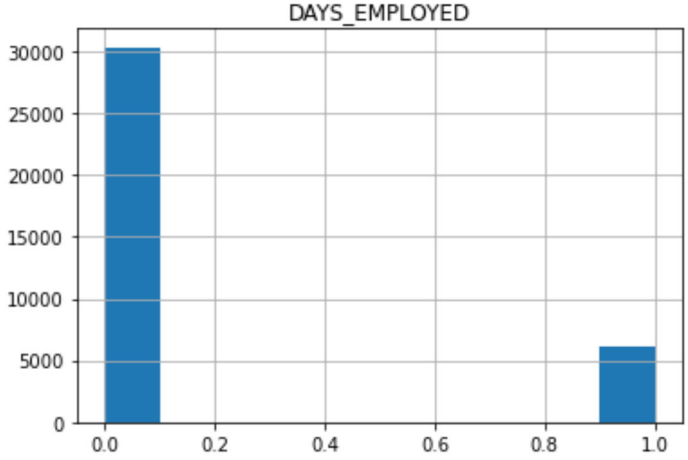

NAME_INCOME_TYPE
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

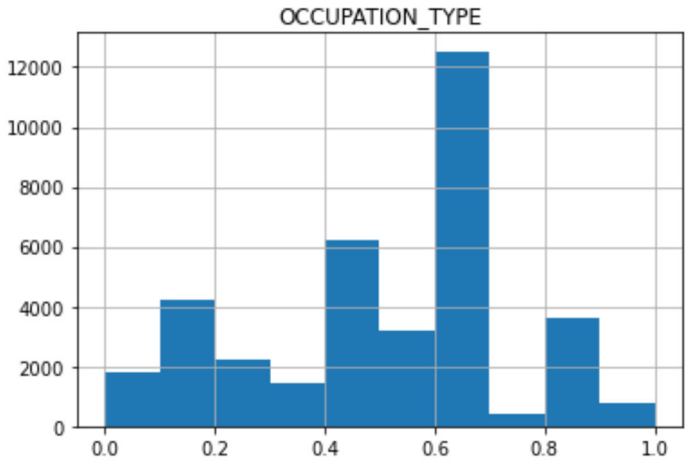

NAME_EDUCATION_TYPE
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

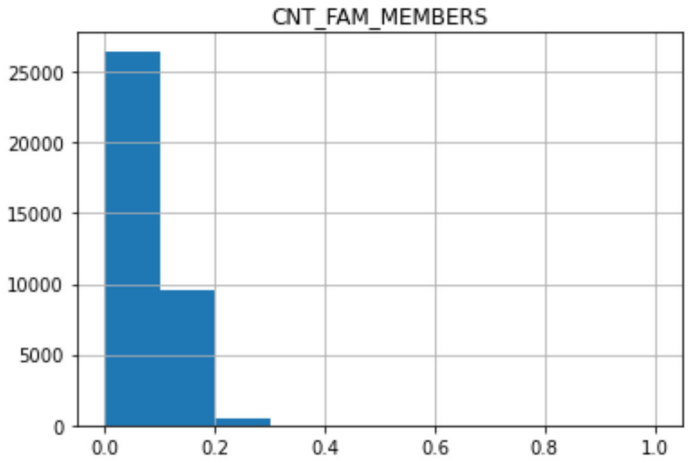

NAME_FAMILY_STATUS
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


NAME_HOUSING_TYPE
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


DAYS_BIRTH
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

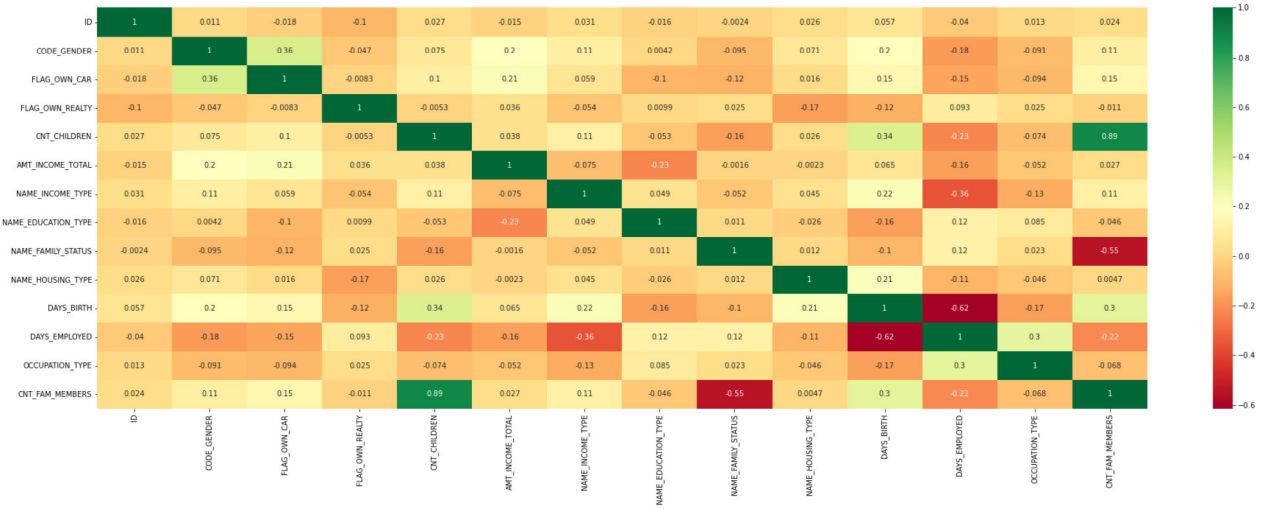

DAYS_EMPLOYED
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0





In [283…
```
#displayed a heatmap so that I could visualize the features better
plt.figure(figsize=(32,10))
sns.heatmap(x_train.corr(),annot=True,cmap='RdYlGn')
```

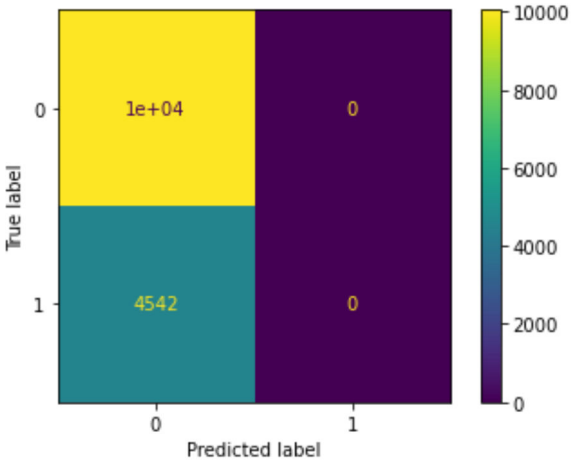
Out[283]:
```
<AxesSubplot:>
```

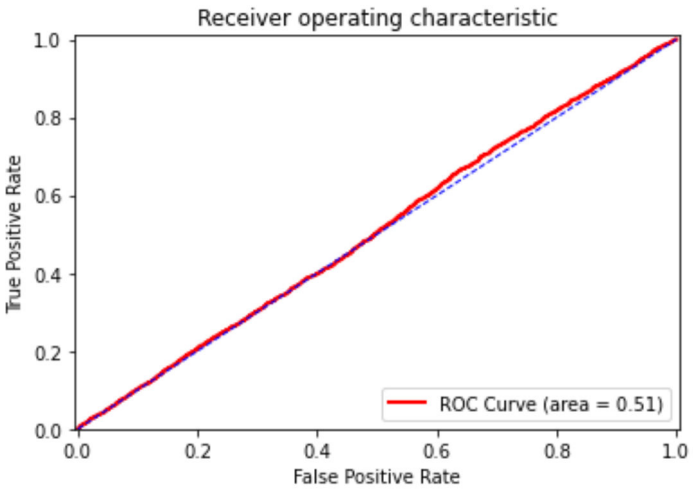


In [284…
```
#Fitted, tested, and graphed five different models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import ConfusionMatrixDisplay,accuracy_score
from sklearn import metrics
```

```
#Model list and names, stored as tupples with string to make iterating in a loop easier
model_list=[(LogisticRegression,"LogisticRegression"),(RandomForestClassifier,"RandomForestClassi
            (DecisionTreeClassifier,"DecisionTreeClassifier"),(GaussianNB,"GaussianNB"),
            (KNeighborsClassifier,"KNeighborsClassifier")]

#Loop to iterate through each model
for model in model_list :
    modelname=model[1]
    model = model[0]
    model = model()
    model.fit(x_train, np.ravel(y_train,order='C'))
    y_predict = model.predict(x_test)
    acc=accuracy_score(y_test,y_predict)
    print(modelname," Accuracy: ",acc)
    ConfusionMatrixDisplay.from_predictions(y_test,y_predict)
    y_predict_prob_lr = model.predict_proba(x_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_predict_prob_lr[:,1], pos_label=1)
    AUC = metrics.auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='red', lw=2,
         label='ROC Curve (area = %0.2f)' % AUC)
    plt.plot([0, 1], [0, 1], color='blue', lw=1, linestyle='--')
    plt.xlim([-0.005, 1.005])
    plt.ylim([0.0, 1.01])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
    print("Area Under Curve: ",AUC,"\n\n")
```

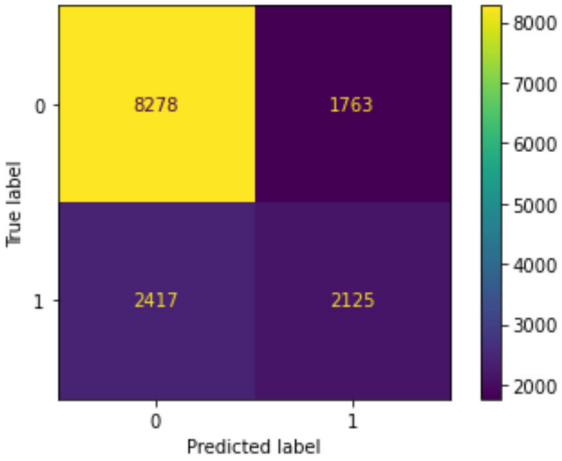
```
LogisticRegression  Accuracy:  0.6885414523760544
```

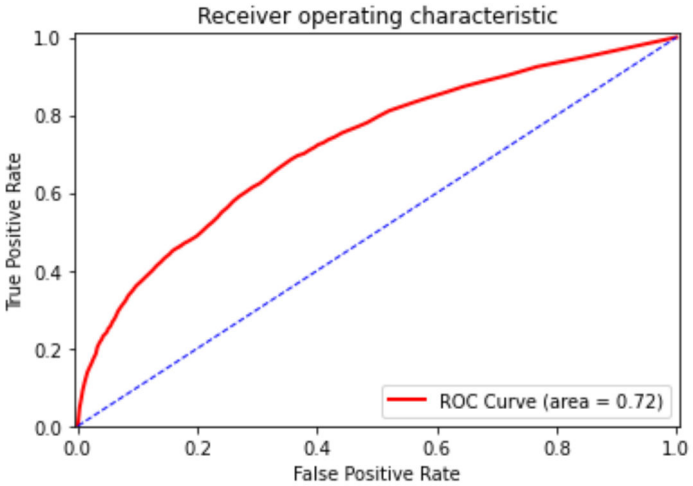




```
Area Under Curve:  0.5081639518397293
```

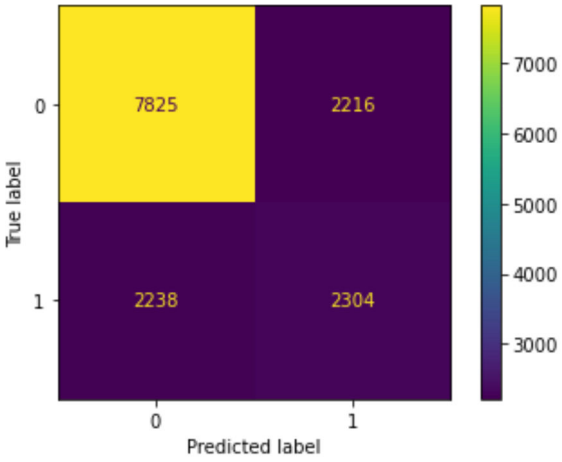
```
RandomForestClassifier  Accuracy:  0.7133648769114722
```

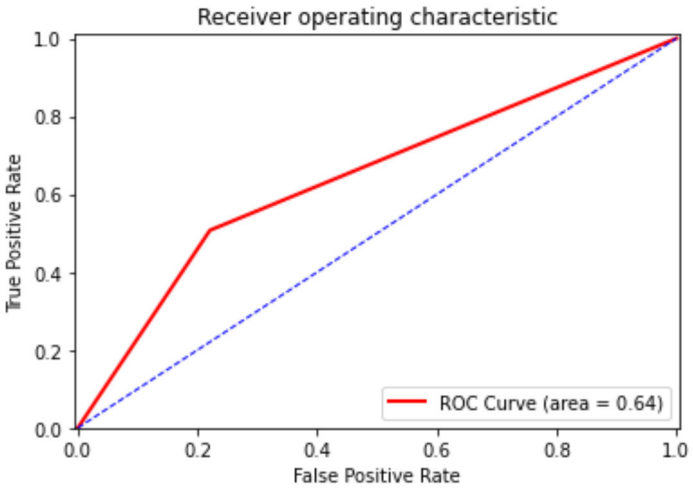





```
Area Under Curve:  0.7213499180002239
```

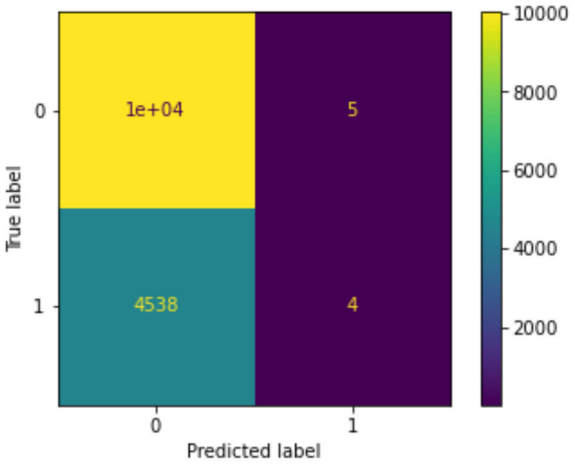
```
DecisionTreeClassifier  Accuracy:  0.6945758760200234
```

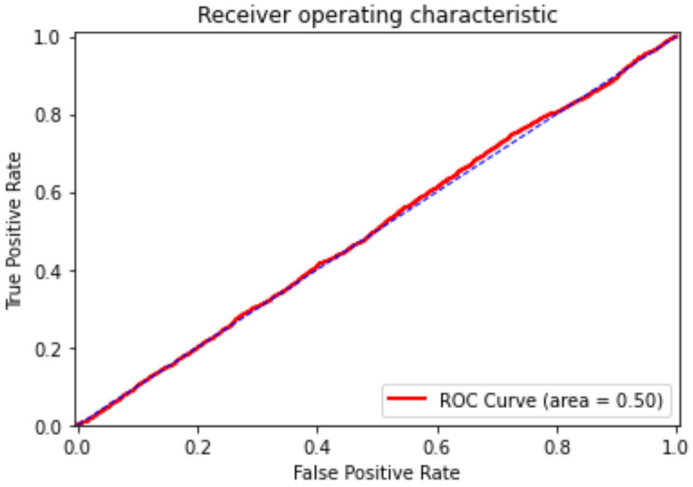




Area Under Curve: 0.643285185955548

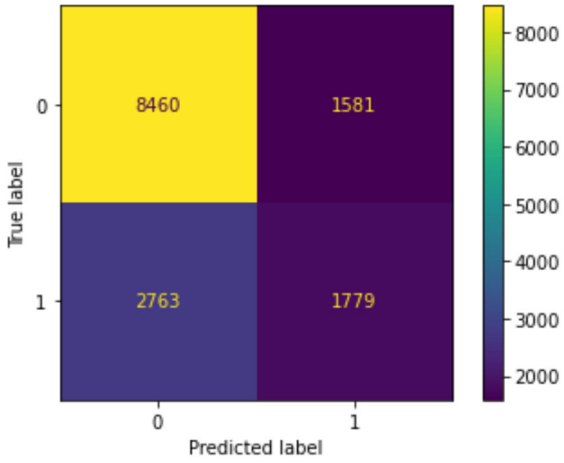
GaussianNB Accuracy: 0.6884728793801002

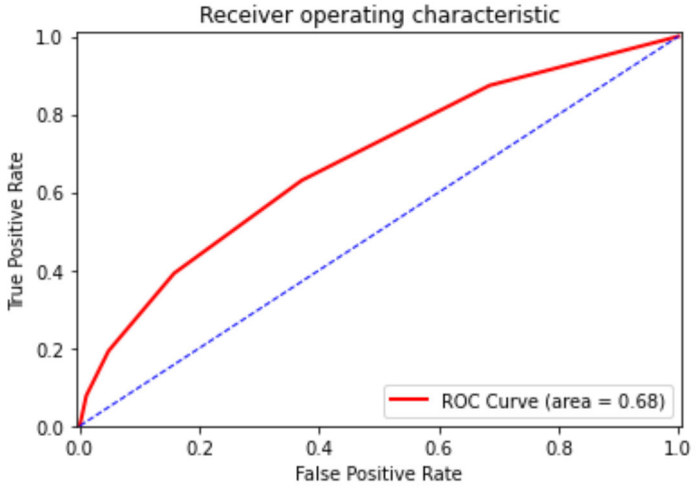




Area Under Curve: 0.5042535643491803

KNeighborsClassifier Accuracy: 0.7021189055749846






```
Area Under Curve:  0.6780217620306282
```

In [285…

```
#Graph we used to visualize the decision tree classifier
#We had initial issues getting decision tree to work
#and we were unsure of the issue, this visualization helped
#us realize the problem
#Left in to demonstrate some of the tools we utilized

from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

model = DecisionTreeClassifier(criterion='entropy',random_state=1)
model.fit(x_train, np.ravel(y_train,order='C'))
dot_data = StringIO()
feature_cols=x_train.columns
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('test.png')
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.161145 to fit

dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.161145 to fit
```

Out[285]: