Girls Who Code Humanize AI Challenge !

Project Name: RoboDebt Advisor, Using Machine Learning for Financial Education

Overview: In this practical application, our goal is to use Decision Trees, in detecting a customer who is likely to default on his credit card debt. We will utilize the dataset available from UC Irvine that is also hosted on Kaggle for this project. The datasets were made publicly available by UCI:**UCI (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients#)**.

> The basic idea is to develop a classifier using Random Forest algorithm, that takes in the UCI Credit Card transaction data and classifies it with a label (in case it's an "default") assigns the numeric value of 1 (one), and a label ("for normal credit ") assigns the numeric value of 0 (zero). The classifier will be an application sitting on top of the RoboAdvisor platform. The application will take the customer data from Mint and pass it through the classifier to classify whether the customer account is heading for a credit card default.

In [1]:

```
# Here we will import the libraries used for machine learning
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from scipy.stats import randint
import pandas as pd # data processing, CSV file I/O, data manipulation
import matplotlib.pyplot as plt # this is used for the plot the graph
import seaborn as sns # used for plot interactive graph.
from pandas import set_option
plt.style.use('ggplot') # nice plots
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split # to split the data in
to two parts
from sklearn.model_selection import KFold # for cross validation
from sklearn.preprocessing import StandardScaler # for normalization
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics # for the check the error and accuracy of the m
odel
import warnings
warnings.filterwarnings('ignore')
```

Step 1: Read in the Data

In [2]:

```
# We are reading the data which is stored in a comma separated value file a
nd storing into a dataframe.
# A dataframe is a structure which comprises of rows and columns and can be
thought of as something similar to
# a Microsoft Excel spreadsheet.
# The columns contain the attributes and the rows contain the information f
or one customer.

data = pd.read_csv('./data/UCI_Credit_Card.csv')
data.sample(5)
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3
4128	4129	270000.0	2	1	2	33	-2	-2	-2
26521	26522	70000.0	2	3	1	52	-1	-1	-1
7397	7398	360000.0	1	2	1	43	1	-2	-2
13946	13947	100000.0	1	6	2	51	2	0	0
6173	6174	160000.0	2	1	1	48	1	-2	-2

5 rows × 25 columns

In [3]:
```
# We are renaming the last column to make it more readable

data.rename(columns={"default.payment.next.month": "Default"}, inplace=Tru
e)
data.drop('ID', axis = 1, inplace =True) # drop column "ID"

# The next code snippet prints information about the DataFrame.
#The information contains the number of columns, column labels, column data
types, memory usage,
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   LIMIT_BAL  30000 non-null  float64
 1   SEX        30000 non-null  int64  
 2   EDUCATION  30000 non-null  int64  
 3   MARRIAGE   30000 non-null  int64  
 4   AGE        30000 non-null  int64  
 5   PAY_0      30000 non-null  int64  
 6   PAY_2      30000 non-null  int64  
 7   PAY_3      30000 non-null  int64  
 8   PAY_4      30000 non-null  int64  
 9   PAY_5      30000 non-null  int64  
 10  PAY_6      30000 non-null  int64  
 11  BILL_AMT1  30000 non-null  float64
 12  BILL_AMT2  30000 non-null  float64
 13  BILL_AMT3  30000 non-null  float64
 14  BILL_AMT4  30000 non-null  float64
 15  BILL_AMT5  30000 non-null  float64
 16  BILL_AMT6  30000 non-null  float64
 17  PAY_AMT1   30000 non-null  float64
 18  PAY_AMT2   30000 non-null  float64
 19  PAY_AMT3   30000 non-null  float64
 20  PAY_AMT4   30000 non-null  float64
 21  PAY_AMT5   30000 non-null  float64
 22  PAY_AMT6   30000 non-null  float64
 23  Default    30000 non-null  int64  
dtypes: float64(13), int64(11)
memory usage: 5.5 MB
```

As per UCI archive the attribute description is as follows:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables: X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit. X2: Gender (1 = male; 2 = female). X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others). X4: Marital status (1 = married; 2 = single; 3 = others). X5: Age (year). X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above. X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005. X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

In [4]:
```
# Separating features (all columns used to predict the target ) and target which is the variable holding the prediction
# In our case it is a 1 or a 0
y = data.Default     # target default=1 or non-default=0
features = data.drop('Default', axis = 1, inplace = False)
```

In [5]:
```
# The following method in pandas library is used to find the unique values from a series.
#A series is a single column of a data frame.

data['EDUCATION'].unique()
```

Out[5]:
```
array([2, 1, 3, 5, 4, 6, 0], dtype=int64)
```

In [6]:
```
data['EDUCATION']=np.where(data['EDUCATION'] == 5, 4, data['EDUCATION'])
data['EDUCATION']=np.where(data['EDUCATION'] == 6, 4, data['EDUCATION'])
data['EDUCATION']=np.where(data['EDUCATION'] == 0, 4, data['EDUCATION'])
```

In [7]:
```
data['EDUCATION'].unique()
```

Out[7]:
```
array([2, 1, 3, 4], dtype=int64)
```

In [8]:
```
data['MARRIAGE'].unique()
```

Out[8]:
```
array([1, 2, 3, 0], dtype=int64)
```

In [9]:
```
data['MARRIAGE']=np.where(data['MARRIAGE'] == 0, 3, data['MARRIAGE'])
data['MARRIAGE'].unique()
```

Out[9]:
```
array([1, 2, 3], dtype=int64)
```

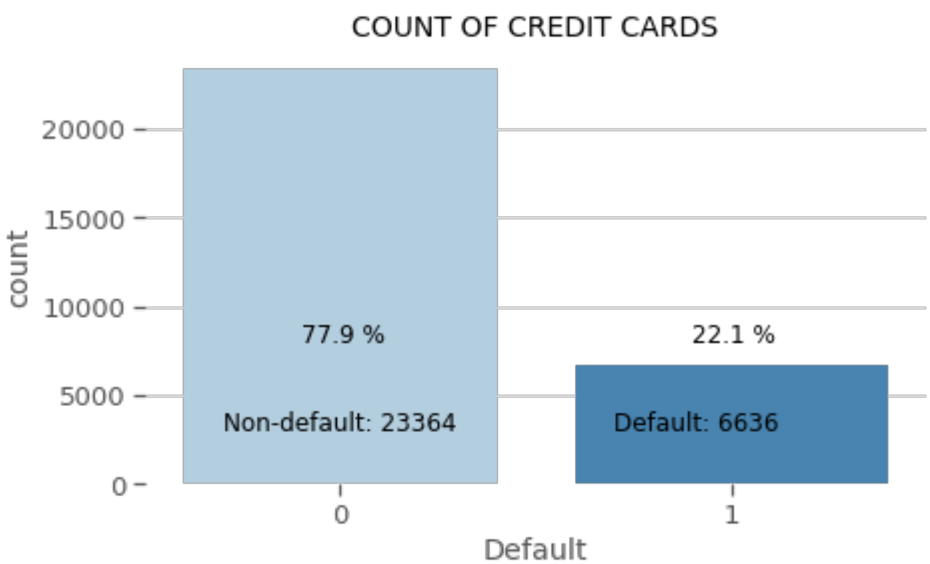
Step 3: Understanding the Features in the Dataset

In [10]:

```
# The frequency of defaults
yes = data.Default.sum()
no = len(data)-yes

# Percentage
yes_perc = round(yes/len(data)*100, 1)
no_perc = round(no/len(data)*100, 1)

import sys
plt.figure(figsize=(7,4))
sns.set_context('notebook', font_scale=1.2)
sns.countplot('Default',data=data, palette="Blues")
plt.annotate('Non-default: {}'.format(no), xy=(-0.3, 15000), xytext=(-0.3, 3000), size=12)
plt.annotate('Default: {}'.format(yes), xy=(0.7, 15000), xytext=(0.7, 3000), size=12)
plt.annotate(str(no_perc)+" %", xy=(-0.3, 15000), xytext=(-0.1, 8000), size=12)
plt.annotate(str(yes_perc)+" %", xy=(0.7, 15000), xytext=(0.9, 8000), size=12)
plt.title('COUNT OF CREDIT CARDS', size=14)
#Removing the frame
plt.box(False);
```



In [11]:

```
set_option('display.width', 100)
set_option('precision', 2)

print("SUMMARY STATISTICS OF NUMERIC COLUMNS")
print()
print(data.describe().T)
```

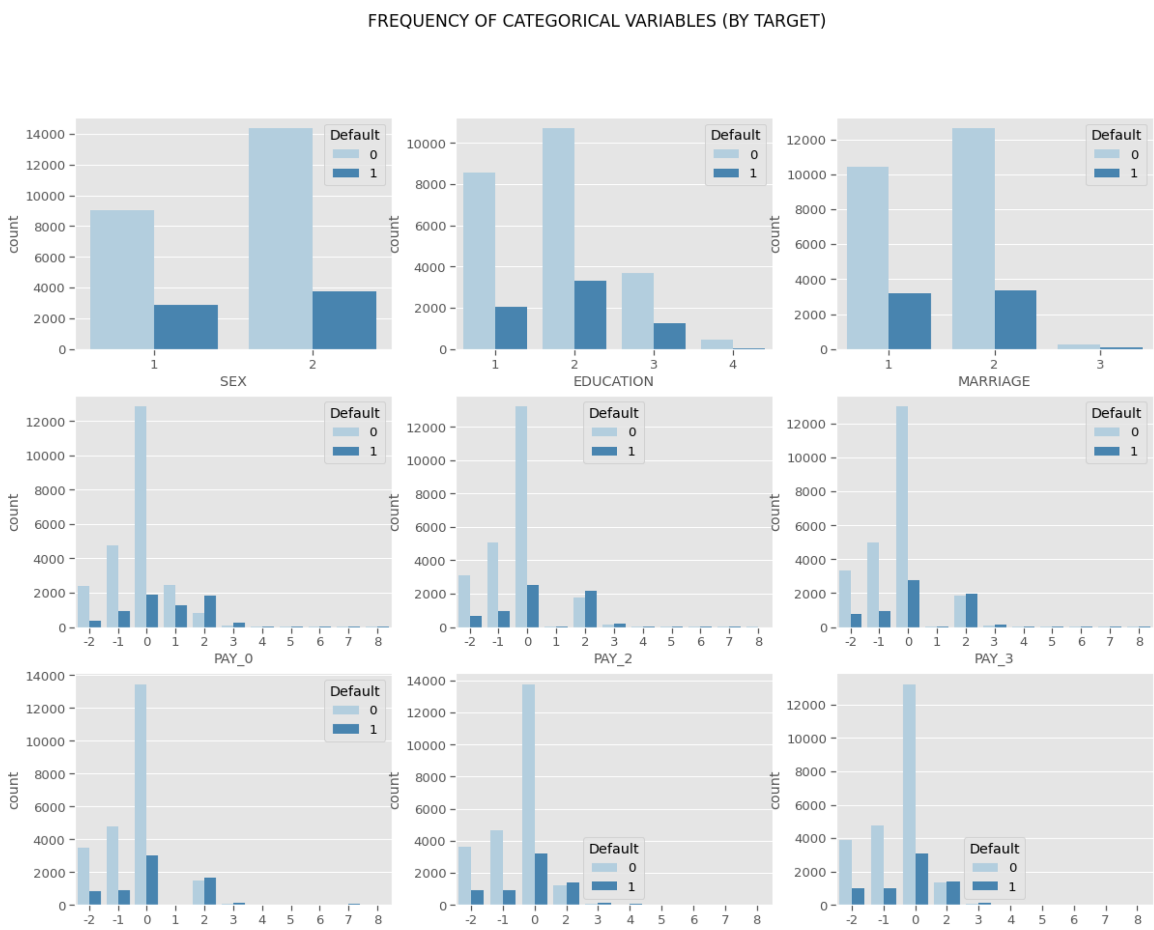
SUMMARY STATISTICS OF NUMERIC COLUMNS

	count	mean	std	min	25%	50%	75%	max
LIMIT_BAL	30000.0	167484.32	129747.66	10000.0	50000.00	140000.0	240000.00	1.00e+06
SEX	30000.0	1.60	0.49	1.0	1.00	2.0	2.00	2.00e+00
EDUCATION	30000.0	1.84	0.74	1.0	1.00	2.0	2.00	4.00e+00
MARRIAGE	30000.0	1.56	0.52	1.0	1.00	2.0	2.00	3.00e+00
AGE	30000.0	35.49	9.22	21.0	28.00	34.0	41.00	7.90e+01
PAY_0	30000.0	-0.02	1.12	-2.0	-1.00	0.0	0.00	8.00e+00
PAY_2	30000.0	-0.13	1.20	-2.0	-1.00	0.0	0.00	8.00e+00
PAY_3	30000.0	-0.17	1.20	-2.0	-1.00	0.0	0.00	8.00e+00
PAY_4	30000.0	-0.22	1.17	-2.0	-1.00	0.0	0.00	8.00e+00
PAY_5	30000.0	-0.27	1.13	-2.0	-1.00	0.0	0.00	8.00e+00
PAY_6	30000.0	-0.29	1.15	-2.0	-1.00	0.0	0.00	8.00e+00
BILL_AMT1	30000.0	51223.33	73635.86	-165580.0	3558.75	22381.5	67091.00	9.65e+05
BILL_AMT2	30000.0	49179.08	71173.77	-69777.0	2984.75	21200.0	64006.25	9.84e+05
BILL_AMT3	30000.0	47013.15	69349.39	-157264.0	2666.25	20088.5	60164.75	1.66e+06
BILL_AMT4	30000.0	43262.95	64332.86	-170000.0	2326.75	19052.0	54506.00	8.92e+05
BILL_AMT5	30000.0	40311.40	60797.16	-81334.0	1763.00	18104.5	50190.50	9.27e+05
BILL_AMT6	30000.0	38871.76	59554.11	-339603.0	1256.00	17071.0	49198.25	9.62e+05
PAY_AMT1	30000.0	5663.58	16563.28	0.0	1000.00	2100.0	5006.00	8.74e+05
PAY_AMT2	30000.0	5921.16	23040.87	0.0	833.00	2009.0	5000.00	1.68e+06
PAY_AMT3	30000.0	5225.68	17606.96	0.0	390.00	1800.0	4505.00	8.96e+05
PAY_AMT4	30000.0	4826.08	15666.16	0.0	296.00	1500.0	4013.25	6.21e+05
PAY_AMT5	30000.0	4799.39	15278.31	0.0	252.50	1500.0	4031.50	4.27e+05
PAY_AMT6	30000.0	5215.50	17777.47	0.0	117.75	1500.0	4000.00	5.29e+05
Default	30000.0	0.22	0.42	0.0	0.00	0.0	0.00	1.00e+00

In [12]:

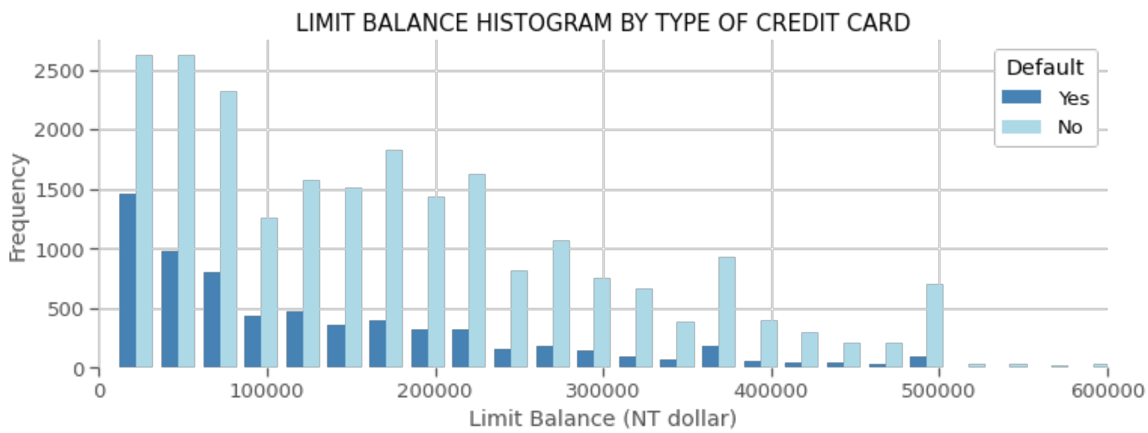
```
# Creating a new dataframe with categorical variables
subset = data[['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', '
PAY_4',
               'PAY_5', 'PAY_6', 'Default']]

f, axes = plt.subplots(3, 3, figsize=(20, 15), facecolor='white')
f.suptitle('FREQUENCY OF CATEGORICAL VARIABLES (BY TARGET)')
ax1 = sns.countplot(x="SEX", hue="Default", data=subset, palette="Blues", a
x=axes[0,0])
ax2 = sns.countplot(x="EDUCATION", hue="Default", data=subset, palette="Blu
es",ax=axes[0,1])
ax3 = sns.countplot(x="MARRIAGE", hue="Default", data=subset, palette="Blue
s",ax=axes[0,2])
ax4 = sns.countplot(x="PAY_0", hue="Default", data=subset, palette="Blues",
ax=axes[1,0])
ax5 = sns.countplot(x="PAY_2", hue="Default", data=subset, palette="Blues",
ax=axes[1,1])
ax6 = sns.countplot(x="PAY_3", hue="Default", data=subset, palette="Blues",
ax=axes[1,2])
ax7 = sns.countplot(x="PAY_4", hue="Default", data=subset, palette="Blues",
ax=axes[2,0])
ax8 = sns.countplot(x="PAY_5", hue="Default", data=subset, palette="Blues",
ax=axes[2,1])
ax9 = sns.countplot(x="PAY_6", hue="Default", data=subset, palette="Blues",
ax=axes[2,2]);
```



In [13]:
```
x1 = list(data[data['Default'] == 1]['LIMIT_BAL'])
x2 = list(data[data['Default'] == 0]['LIMIT_BAL'])

plt.figure(figsize=(12,4))
sns.set_context('notebook', font_scale=1.2)
#sns.set_color_codes("pastel")
plt.hist([x1, x2], bins = 40, density=False, color=['steelblue', 'lightblue
'])
plt.xlim([0,600000])
plt.legend(['Yes', 'No'], title = 'Default', loc='upper right', facecolor='
white')
plt.xlabel('Limit Balance (NT dollar)')
plt.ylabel('Frequency')
plt.title('LIMIT BALANCE HISTOGRAM BY TYPE OF CREDIT CARD', SIZE=15)
plt.box(False)
plt.savefig('ImageName', format='png', dpi=200, transparent=True);
```

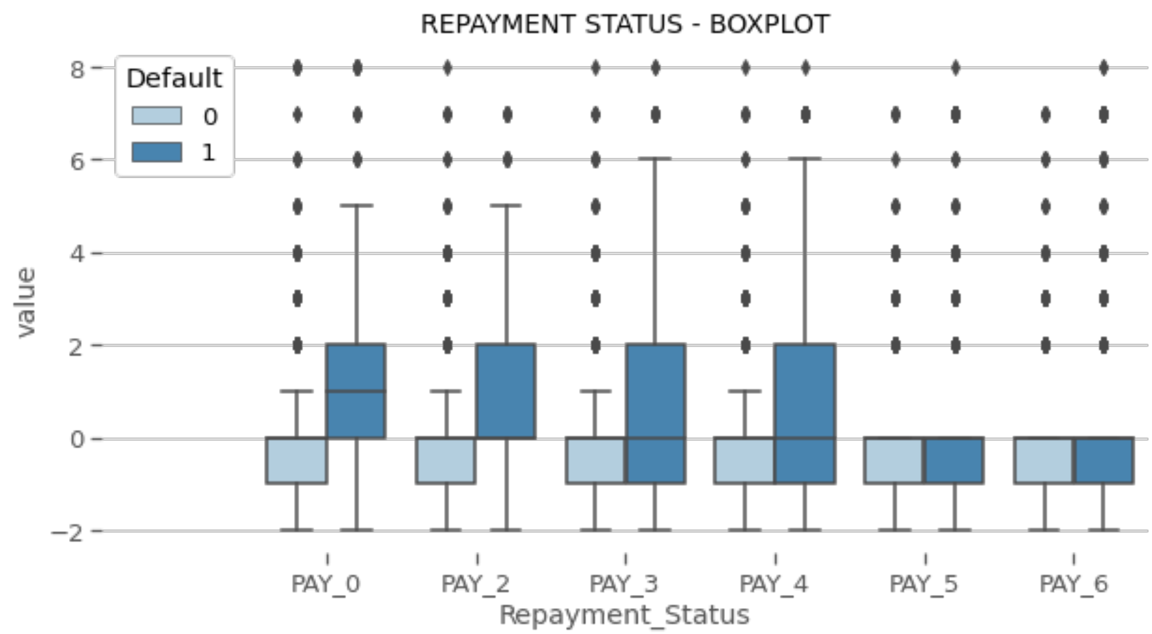


In [14]:

```
Repayment = data[['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']]

Repayment = pd.concat([y,Repayment],axis=1)
Repayment = pd.melt(Repayment,id_vars="Default",
                    var_name="Repayment_Status",
                    value_name='value')

plt.figure(figsize=(10,5))
sns.set_context('notebook', font_scale=1.2)
sns.boxplot(y="value", x="Repayment_Status", hue="Default", data=Repayment,
palette='Blues')
plt.legend(loc='best', title= 'Default', facecolor='white')
plt.xlim([-1.5,5.5])
plt.title('REPAYMENT STATUS - BOXPLOT', size=14)
plt.box(False)
plt.savefig('ImageName', format='png', dpi=200);
```



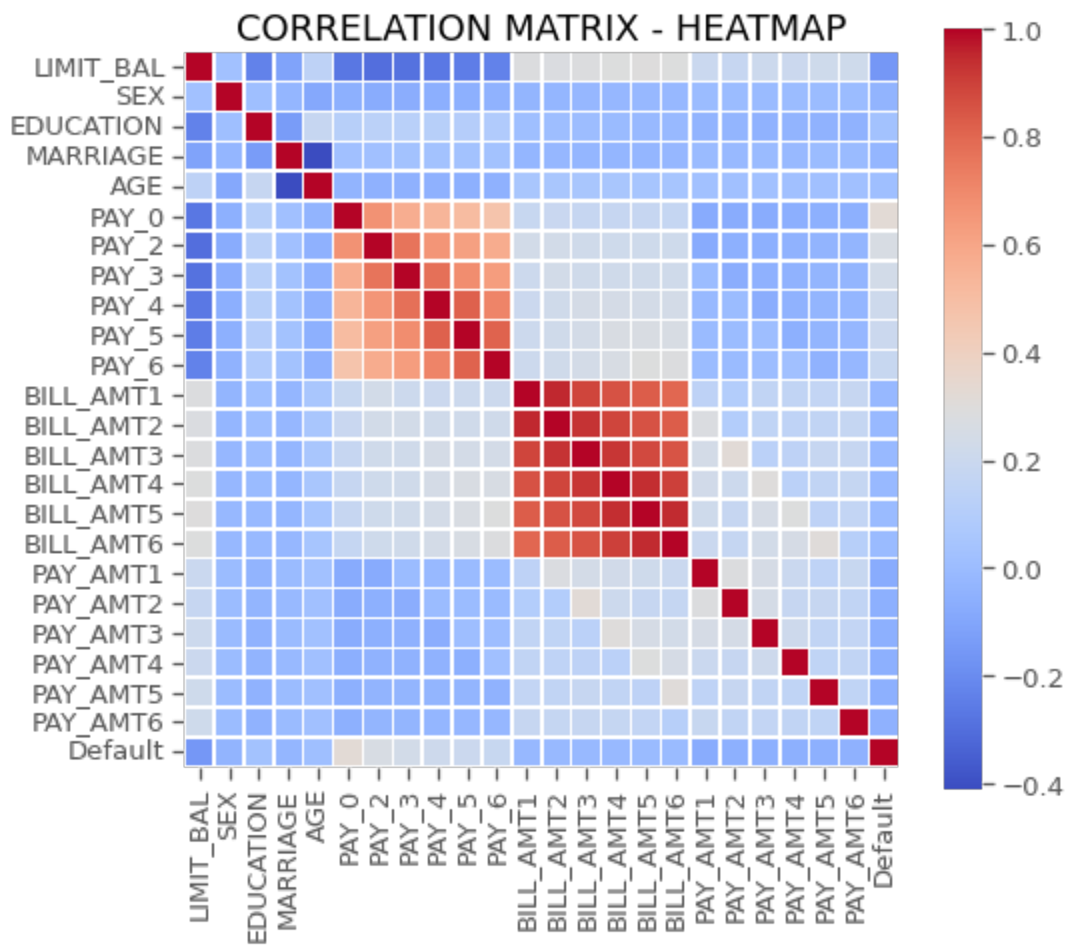
In [15]:

```
stdX = (features - features.mean()) / (features.std())             # stand
ardization
data_st = pd.concat([y,stdX.iloc[:,:]],axis=1)
data_st = pd.melt(data_st,id_vars="Default",
                    var_name="features",
                    value_name='value')
```

Correlation Heatmap of the Customer transaction data Fields

> The correlation matrix provides us with an indication of how well (or not so well) each feature is correlated with each other. The returned value will be between -1 and +1, with higher correlations tending toward these endpoints, and poorer correlations tending towards 0.

In [16]:
```
#  looking at correlations matrix, defined via Pearson function
corr = data.corr() # .corr is used to find corelation
f,ax = plt.subplots(figsize=(8, 7))
sns.heatmap(corr, cbar = True,  square = True, annot = False, fmt= '.1f',
            xticklabels= True, yticklabels= True
            ,cmap="coolwarm", linewidths=.5, ax=ax)
plt.title('CORRELATION MATRIX - HEATMAP', size=18);
```



Step 4: Preparing for Machine Learning: Train/Test Split

With the data prepared, split it into a train and test set.

In [17]:
```
# Original dataset
X = data.drop('Default', axis=1)
y = data['Default']

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, str
atify=y, random_state=42)
```

In [18]:
```
# Dataset with standardized features
Xstd_train, Xstd_test, ystd_train, ystd_test = train_test_split(stdX,y, tes
t_size=0.2, stratify=y,
                                                                 random_stat
e=42)
```

In [19]:
```
# Dataset with three most important features
Ximp = stdX[['PAY_0', 'BILL_AMT1', 'PAY_AMT2']]
X_tr, X_t, y_tr, y_t = train_test_split(Ximp,y, test_size=0.2, stratify=y,
random_state=42)
```

In [20]:
```
# Printing out last 15 lines of the dataset
data.tail(15)
```

Out[20]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
29985	240000.0	1	1	2	30	-2	-2	-2	-2
29986	360000.0	1	1	2	35	-1	-1	-2	-2
29987	130000.0	1	1	2	34	0	0	0	0
29988	250000.0	1	1	1	34	0	0	0	0
29989	150000.0	1	1	2	35	-1	-1	-1	-1
29990	140000.0	1	2	1	41	0	0	0	0
29991	210000.0	1	2	1	34	3	2	2	2
29992	10000.0	1	3	1	43	0	0	0	-2
29993	100000.0	1	1	2	38	0	-1	-1	0
29994	80000.0	1	2	2	34	2	2	2	2
29995	220000.0	1	3	1	39	0	0	0	0
29996	150000.0	1	3	2	43	-1	-1	-1	-1
29997	30000.0	1	2	2	37	4	3	2	-1
29998	80000.0	1	3	1	41	1	-1	0	0
29999	50000.0	1	2	1	46	0	0	0	0

15 rows × 24 columns

Step 6: Creating a Random Forest Classifier and feeding it the prepared data

With the data prepared, our model is now ready to learn the patterns.

In [21]:
```
Ran = RandomForestClassifier(criterion= 'gini', max_depth= 6,
                                     max_features= 5, n_estimators= 150,
                                     random_state=0)
Ran.fit(X_train, y_train)
y_pred = Ran.predict(X_test)
print('Accuracy:', metrics.accuracy_score(y_pred,y_test))

## 5-fold cross-validation
cv_scores =cross_val_score(Ran, X, y, cv=5)

# Print the 5-fold cross-validation scores
print()
print(classification_report(y_test, y_pred))
print()
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)),
      ", Standard deviation: {}".format(round(np.std(cv_scores),4)))

plt.figure(figsize=(4,3))
ConfMatrix = confusion_matrix(y_test,Ran.predict(X_test))
sns.heatmap(ConfMatrix,annot=True, cmap="Blues", fmt="d",
            xticklabels = ['Non-default', 'Default'],
            yticklabels = ['Non-default', 'Default'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion Matrix - Random Forest");
```

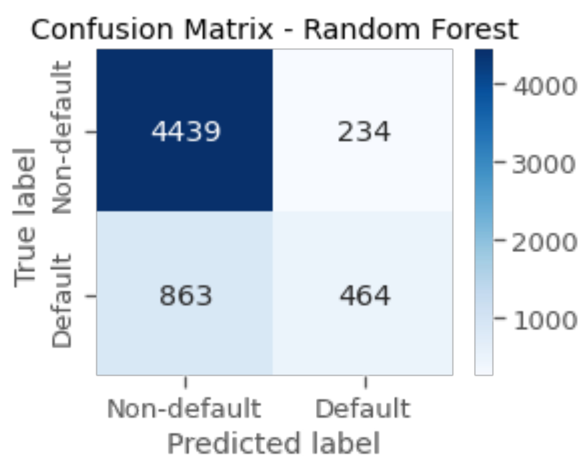
```
Accuracy: 0.8171666666666667

              precision    recall  f1-score   support

           0       0.84      0.95      0.89      4673
           1       0.66      0.35      0.46      1327

    accuracy                           0.82      6000
   macro avg       0.75      0.65      0.67      6000
weighted avg       0.80      0.82      0.79      6000


Average 5-Fold CV Score: 0.8203 , Standard deviation: 0.0093
```



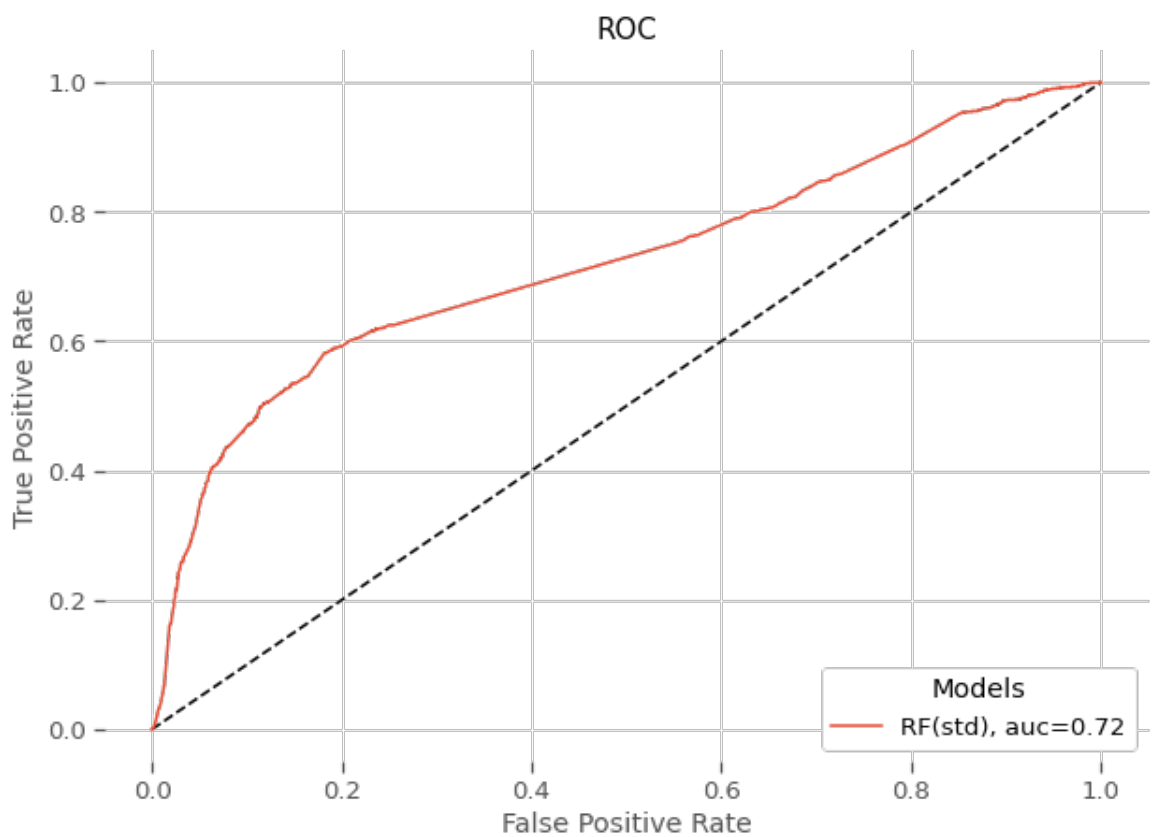
Step 7: Score the Model

What is the accuracy and how well is our model doing? We can find this out using “Area Under the Curve” (AUC) of the “Receiver Operating Characteristic” (ROC) plots.

A ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR). The true positive rate is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)). Similarly, the false positive rate is the proportion of observations that are incorrectly predicted to be positive out of all negative observations (FP/(TN + FP)).

In [22]:
```
y_pred_proba_RF = Ran.predict_proba(Xstd_test)[::,1]
fpr4, tpr4, _ = metrics.roc_curve(ystd_test,  y_pred_proba_RF)
auc4 = metrics.roc_auc_score(ystd_test, y_pred_proba_RF)

plt.figure(figsize=(10,7))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr4,tpr4,label="RF(std), auc="+str(round(auc4,2)))
plt.legend(loc=4, title='Models', facecolor='white')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC', size=15)
plt.box(False)
plt.savefig('ImageName', format='png', dpi=200, transparent=True);
```



Step 8: Generate Predictions for new customer data based on our Random Forest Model.

The RF_Prediction Function is used to view the prediction from the Random Forest Model.

In [23]:
```
def RF_Prediction(creditCardData):

     crd=pd.DataFrame(creditCardData)

     results = Ran.predict(crd)

     print("The predicted Credit  status is: $", results)
     print("")
     print("Here is how to interpret the credit default status of a custome
r:")
     print("")
     print("An outcome of  '1' indicates that there is high likely hood of
a default on their credit card debt")
     print("")
     print("An outcome of '0' indicates that the customer has high probabil
ity of paying their credit card debt" )
     return
```

Prepare a Random set of new Customer Credit Card data for prediction through our Random Forest model based classifier

In [24]:
```
newCustomerCreditCardData=data[29991:]
newCustomerCreditCardData= newCustomerCreditCardData.drop("Default", axis=
1)
```

Generate some predictions

In [25]:
```
RF_Prediction(newCustomerCreditCardData)
```

```
The predicted Credit  status is: $ [1 0 0 1 0 0 1 0 0]

Here is how to interpret the credit default status of a customer:

An outcome of  '1' indicates that there is high likely hood of a default on
their credit card debt

An outcome of '0' indicates that the customer has high probability of payin
g their credit card debt
```

Result Summary:

> **We can see that the Random Forest model is fairly accurate. It got two predictions wrong which is in line with the accuracy rate on test data and ROC characteristics**

Findings and Actionable Insights:

> 1. The objective of this Machine Learning project was to build a classification model to predict whether a customer is likely to default on his or her credit card debt based 22 attributes found in their transaction data.

Next Steps in the Random Forest Classifier Model Enhancement:

> 1. Fine Tuning the model based on domain knowledge and feature importance results
> 2. We could reduce the dimensions/features further to tune the model.
> 3. We could try other algorithms like XGBoost as a next step in improving the performance of the current Random Forest based model
> 4. Based on what I have read in Medium and in youtube videos, advanced techniques like Neural Networks/Autoencoders and LSTM based model seems to be well suited for classification problems. This way you do not have to label the data. However these techniques are well beyond my current abilities, as I recently learned about machine learning. Hopefully some time in the near future :-)