1. Credit card applications project introduction

Commerical banks have enormous amount of applications for credit cards. These applications get evaluated based on many reasons, that include high loan balances, low income levels, or too many inquiries on an individual's credit report, for example. Manually analyzing these applications is mundane, error-prone, and time-consuming. The task of evaluating credit cards applications can be automated with the power of machine learning. Pretty much every commercial bank does so nowadays. In this notebook, we will build an automatic credit card approval predictor using machine learning techniques, just like the real banks do!

We are using UCI Credit Card Approval Dataset (http://archive.ics.uci.edu/ml/datasets/credit+approval). The structure of this project is as follows:

- Load and view the dataset
- Deal with missing values and object variables
- Preprocess datasets
- Exploratory data analysis
- Build ML model

In [42]:
```
# Load necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
```

2. Loading and exploring data

In [43]:
```
# Load dataset
cc_apps = pd.read_csv("cc_approvals.data", header=None)

# Inspect data
# ... YOUR CODE FOR TASK 1 ...
print(cc_apps.head())
```

```
   0      1      2  3  4  5  6     7  8  9   10 11 12     13   14 15
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  f  g  00202    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  f  g  00043  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  f  g  00280  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  t  g  00100    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  f  s  00120    0  +
```

The features have been anonymized to protect privacy, but this blog (http://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html) gives a pretty good idea of the probable features. The probable features in a typical credit card application are Gender, Age, Debt, Married, BankCustomer, EducationLevel, Ethnicity, YearsEmployed, PriorDefault, Employed, CreditScore, DriversLicense, Citizen, ZipCode, Income and finally the ApprovalStatus. This gives us a pretty good starting point, and we can map these features with respect to the columns in the output.

From looking at the first 5 rows of the dataset we can see that this data are a mixture of numeric and string data. We will fix that during the preprocessing phase.

In [44]:
```
#Summary Statistics + DIsplay if there are any columns with null values

def display_all(df):
    with pd.option_context("display.max_rows", 1000, "display.max_columns", 1000, "display.max_colwidth", 1000):
        display(df)

print(cc_apps.describe())
print("\n")
print(cc_apps.info())
print("\n")
cc_apps.tail(17)
```

```
                2           7          10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       690 non-null    object 
 1   1       690 non-null    object 
 2   2       690 non-null    float64
 3   3       690 non-null    object 
 4   4       690 non-null    object 
 5   5       690 non-null    object 
 6   6       690 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    object 
 13  13      690 non-null    object 
 14  14      690 non-null    int64  
 15  15      690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
None
```

Out[44]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
673	?	29.50	2.000	y	p	e	h	2.000	f	f	0	f	g	00256	17	-
674	a	37.33	2.500	u	g	i	h	0.210	f	f	0	f	g	00260	246	-
675	a	41.58	1.040	u	g	aa	v	0.665	f	f	0	f	g	00240	237	-
676	a	30.58	10.665	u	g	q	h	0.085	f	t	12	t	g	00129	3	-
677	b	19.42	7.250	u	g	m	v	0.040	f	t	1	f	g	00100	1	-
678	a	17.92	10.210	u	g	ff	ff	0.000	f	f	0	f	g	00000	50	-
679	a	20.08	1.250	u	g	c	v	0.000	f	f	0	f	g	00000	0	-
680	b	19.50	0.290	u	g	k	v	0.290	f	f	0	f	g	00280	364	-
681	b	27.83	1.000	y	p	d	h	3.000	f	f	0	f	g	00176	537	-
682	b	17.08	3.290	u	g	i	v	0.335	f	f	0	t	g	00140	2	-
683	b	36.42	0.750	y	p	d	v	0.585	f	f	0	f	g	00240	3	-
684	b	40.58	3.290	u	g	m	v	3.500	f	f	0	t	s	00400	0	-
685	b	21.08	10.085	y	p	e	h	1.250	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.000	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.000	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.040	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.290	f	f	0	t	g	00000	0	-

3. Missing values

We've uncovered some issues that will affect the performance of our machine learning model(s) if they go unchanged:

Our dataset contains both numeric and non-numeric data (specifically data that are of float64, int64 and object types). Specifically, the features 2, 7, 10 and 14 contain numeric values (of types float64, float64, int64 and int64 respectively) and all the other features contain non-numeric values. The dataset also contains values from several ranges. Some features have a value range of 0 - 28, some have a range of 2 - 67, and some have a range of 1017 - 100000. Apart from these, we can get useful statistical information (like mean, max, and min) about the features that have numerical values. Finally, the dataset has missing values, which we'll take care of in this section. The missing values in the dataset are labeled with '?', which can be seen in the last cell's output.

In [47]:
```
import numpy as np

cc_apps = cc_apps.replace('?', np.NaN)
print(cc_apps.tail(17))
print('\n')
sum(cc_apps.isna().sum(axis = 0))
```

```
      0      1       2  3  4   5   6      7  8  9   10 11 12     13   1
4 15
673  NaN  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256   1
7  -
674    a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260  24
6  -
675    a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240  23
7  -
676    a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129
3  -
677    b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100
1  -
678    a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000   5
0  -
679    a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000
0  -
680    b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280  36
4  -
681    b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176  53
7  -
682    b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140
2  -
683    b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240
3  -
684    b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400
0  -
685    b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260
0  -
686    a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200  39
4  -
687    a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200
1  -
688    b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280  75
0  -
689    b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000
0  -
```

Out[47]: 67

We are going to impute missing values with median of the feature and also crate additional columns that will let the model know if a value was na.

In [48]:
```
import pandas.api.types as ptypes

for n, c in cc_apps.items():
    if ptypes.is_numeric_dtype(c) and c.isna().sum():
        cc_apps[str(n) + '_na'] = c.isna()
        cc_apps[n] = cc_apps[n].fillna(c.median())

display_all(cc_apps.head())
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

We have successfully taken care of the missing values present in the numeric columns. There are still some missing values to be imputed for columns 0, 1, 3, 4, 5, 6 and 13. All of these columns contain non-numeric data and this why the median imputation strategy would not work here.

We are going to impute these missing values with the most frequent values as present in the respective columns.

In [49]:
```
for n, c in cc_apps.items():
    if ptypes.is_object_dtype(c):
        if c.isna().sum():
            cc_apps[str(n) + '_na'] = c.isna()
            cc_apps[n] = cc_apps[n].fillna(cc_apps[n].value_counts().ind
ex[0])
        cc_apps[n] = pd.Categorical(c, ordered = True)
        cc_apps[n] = cc_apps[n].cat.codes

display_all(cc_apps.head())
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0_na	1_na	3_na	4_na	
0	1	156	0.000	1	0	12	7	1.25	1	1	1	0	0	68	0	0	False	False	False	False	F
1	0	328	4.460	1	0	10	3	3.04	1	1	6	0	0	11	560	0	False	False	False	False	F
2	0	89	0.500	1	0	10	3	1.50	1	0	0	0	0	96	824	0	False	False	False	False	F
3	1	125	1.540	1	0	12	7	3.75	1	1	5	1	0	31	3	0	False	False	False	False	F
4	1	43	5.625	1	0	12	7	1.71	1	0	0	0	2	37	0	0	False	False	False	False	F

We have successfully converted all non-numeric columns to numeric and imputed all missing values with medians and most frequent values of respective columns.

Now, let's drop Driver's License and ZipCode columns as they have little to no value in predicting credit card approvals and rescale columns to be all in the range from 0 to 1.

In [62]:
```
y = X[15]
X = cc_apps.drop([11,13,15,'13_na'], axis = 1)
```

In [63]:
```
# Instantiate MinMaxScaler and use it to rescale
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
rescaled_X = pd.DataFrame(scaler.fit_transform(X))
```

4. Split the dataset into train and test sets

Now that we have our data in a machine learning modeling-friendly shape, we are really ready to proceed towards creating a machine learning model to predict which credit card applications will be accepted and which will be rejected.

First, we will split our data into train set and test set to prepare our data for two different phases of machine learning modeling: training and testing.

In [72]:
```
from sklearn.model_selection import train_test_split

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(rescaled_X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=101)
```

5. Fitting Logistic Regression and Random Forest Classifier

In [73]:
```
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter val
ues
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(X_train,y_train)
```

Out[73]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbo
se=0,
                   warm_start=False)
```

In [74]:
```
from sklearn.ensemble import RandomForestClassifier

# Instantiate a RandomForestClassifier with default parameter values
m_rf = RandomForestClassifier()

# Fit rf to the train set
m_rf.fit(X_train,y_train)
```

Out[74]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

6. Evaluating Models

We will now evaluate our model on the test set with respect to classification accuracy. But we will also take a look the model's confusion matrix. In the case of predicting credit card applications, it is equally important to see if our machine learning model is able to predict the approval status of the applications as denied that originally got denied. If our model is not performing well in this aspect, then it might end up approving the application that should have been approved. The confusion matrix helps us to view our model's performance from these aspects.

In [75]:
```
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set
y_pred_log = logreg.predict(X_test)

# Get the accuracy score of logreg model and print it
print("Accuracy of logistic regression classifier: ", logreg.score(X_tes
t, y_test))

# Print the confusion matrix of the logreg model
print(confusion_matrix(y_test, y_pred_log))



# Use m_rf to predict instances from the test set
y_pred_rf = m_rf.predict(X_test)

# Get the accuracy score of random forest model and print it
print("Accuracy of random forest regression classifier: ", m_rf.score(X_
test, y_test))

# Print the confusion matrix of the random forest model
print(confusion_matrix(y_test, y_pred_rf))
```

```
Accuracy of logistic regression classifier:  0.8464912280701754
[[ 87  12]
 [ 23 106]]
Accuracy of random forest regression classifier:  0.8464912280701754
[[ 78  21]
 [ 14 115]]
```

7. Grid searching and making models perform better

Our models are performing relatively good. They were able to yield accuracy scores of almost 85%.

For the confusion matrix, the first element of the of the first row of the confusion matrix denotes the true negatives meaning the number of negative instances (denied applications) predicted by the model correctly. And the last element of the second row of the confusion matrix denotes the true positives meaning the number of positive instances (approved applications) predicted by the model correctly.

Let's see if we can do better. We can perform a grid search of the model parameters to improve the model's ability to predict credit card approvals.

scikit-learn's implementation of logistic regression consists of different hyperparameters but we will grid search over the following two:

- tol
- max_iter

scikit-learn's implementation of random forest classifier consists of different hyperparameters but we will grid search over the following two:

- min_samples_leaf
- max_features

In [78]:
```
from sklearn.model_selection import GridSearchCV

# Define the grid of values for tol and max_iter
tol = [0.01, 0.001, 0.0001]
max_iter = [100, 150, 200]

# Create a dictionary where tol and max_iter are keys and the lists of t
heir values are corresponding values
param_grid_log = dict(tol=tol, max_iter=max_iter)
print(param_grid_log)
```

```
{'tol': [0.01, 0.001, 0.0001], 'max_iter': [100, 150, 200]}
```

In [76]:
```
from sklearn.model_selection import GridSearchCV

# Define the grid of values for tol and max_iter
min_samples_leaf = [1, 3, 5, 7]
max_features = [0.1, 0.3, 0.5, 1]

# Create a dictionary where tol and max_iter are keys and the lists of t
heir values are corresponding values
param_grid_rf = dict(min_samples_leaf=min_samples_leaf, max_features=max
_features)
print(param_grid_rf)
```

```
{'min_samples_leaf': [1, 3, 5, 7], 'max_features': [0.1, 0.3, 0.5, 1]}
```

8. Best model selection

We have defined the grid of hyperparameter values and converted them into a single dictionary format which GridSearchCV() expects as one of its parameters. Now, we will begin the grid search to see which values perform best.

We will instantiate GridSearchCV() with our earlier logreg model with all the data we have. Instead of passing train and test sets, we will supply rescaledX and y. We will also instruct GridSearchCV() to perform a cross-validation of five folds.

We'll end the notebook by storing the best-achieved score and the respective best parameters.

While building this credit card predictor, we tackled some of the most widely-known preprocessing steps such as scaling, label encoding, and missing value imputation. We finished with some machine learning to predict if a person's application for a credit card would get approved or not given some information about that person.

In [80]:
```
# Instantiate GridSearchCV with the required parameters
grid_model_log = GridSearchCV(estimator=logreg, param_grid=param_grid_lo
g, cv=5)

# Fit data to grid_model
grid_model_result_log = grid_model_log.fit(rescaled_X, y)

# Summarize results
best_score, best_params = grid_model_result_log.best_score_,grid_model_r
esult_log.best_params_
print("Best: %f using %s" % (best_score, best_params))
```

```
Best: 0.850725 using {'max_iter': 100, 'tol': 0.01}
```

In [81]:
```
# Instantiate GridSearchCV with the required parameters
grid_model_rf = GridSearchCV(estimator=m_rf, param_grid=param_grid_rf, c
v=5)

# Fit data to grid_model
grid_model_result_rf = grid_model_rf.fit(rescaled_X, y)

# Summarize results
best_score, best_params = grid_model_result_rf.best_score_,grid_model_re
sult_rf.best_params_
print("Best: %f using %s" % (best_score, best_params))
```

```
Best: 0.853623 using {'max_features': 0.1, 'min_samples_leaf': 5}
```

In [ ]:
```

```