Predicting House Mortgage Eligibility

Task

Develop a system for predicting house mortgage eligibility in order to automate the process of targeting the right applicants. You are given a set of features describing the applicant and you are asked to decide if this is the right applicant or not. It is up to you to analyze the significance of given features and decide on which to utilize in your solution. You are allowed to use whatever programming language/library you feel the most comfortable with, but Python (and the typical pandas/numpy/scikit-learn/jupyter stack) is preferred. Briefly describe your choices in terms of data preprocessing, feature extraction, algorithms and similarity metrics being used.

Additional questions:

1. How would you assess the performances of your system?
2. How would you treat missing data and/or outliers (if any).
3. How could your system assure that the applicant is able to repay the mortgage with no difficulties?

Data info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

Based on the raw_data.info(), we can see that our dataset consists of 13 columns (12 features and 1 dependent variable) and 614 entries/rows.

We will change the style writing of those columns that are not snake case, and convert everything to be lower-case.

Preprocessing

	loan_id	gender	married	dependents	education	self_employed	applicant_income	coapplicantincome	loan_amount	loan_amount_term	credit_history	property_area	loan_status
count	614	601	611	599	614	582	614.000000	614.000000	592.000000	600.00000	564.000000	614	614
unique	614	2	2	4	2	2	NaN	NaN	NaN	NaN	NaN	3	2
top	ID342	Male	Yes	0	Graduate	No	NaN	NaN	NaN	NaN	NaN	Semiurban	Y
freq	1	489	398	345	480	500	NaN	NaN	NaN	NaN	NaN	233	422
mean	NaN	NaN	NaN	NaN	NaN	NaN	5403.459283	1621.245798	146.412162	342.00000	0.842199	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN	6109.041673	2926.248369	85.587325	65.12041	0.364878	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN	150.000000	0.000000	9.000000	12.00000	0.000000	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN	2877.500000	0.000000	100.000000	360.00000	1.000000	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN	3812.500000	1188.500000	128.000000	360.00000	1.000000	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN	5795.000000	2297.250000	168.000000	360.00000	1.000000	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN	81000.000000	41667.000000	700.000000	480.00000	1.000000	NaN	NaN

Based on the raw_data.describe(), we see that some variables have different number of observations, which implies that there are some missing values. Also, *gender* has 489 male entries, which is almost 80% of the data. This feature probably won't be useful, not only because of the fact that majority is male, but also because this doesn't affect the outcome at all. Gender is not important category when allowing a mortgage.

Let's take a look at the null-values.

```
married               3
dependents           15
education             0
self_employed        32
applicant_income      0
coapplicantincome     0
loan_amount          22
loan_amount_term     14
credit_history       50
property_area         0
loan_status           0
dtype: int64
```

In total, we have 136 null-values. Since this is not 5% of the dataset, we cannot invoke the rule of thumb (If you are removing <5% of the features, you are free to just remove all that have missing values), which means that we will have to omit some of them, and convert others to string/numeric values.

The first column that appears is *married* column. This part is considered as important when taking a request for a mortgage. We will se whether the values from dependents where married.isnull() have non-null values and based on that we will decide which values we will replace married.isnull() with. For example, if the values in the *dependents* column are not null and are not zero, we will replace NaN in the married column with a *Yes*. Otherwise, it will be *No.*

Since the dependents column is important for the outcome, we will drop these, because, replacing null values with values 0,1,2 or 3+ can affect the outcome.

In self_employed column, most of the values are a "No", so we will replace all null-values with a "No"

For the *loan_amount*, *loan_amount_term* and *credit_history* we will replace null values with the median value of the columns.

Encoding Categorical Data

There are multiple ways to encode the data: find and replace method, Label Encoding, One Hot Encoding, Custom Binary Encoding, Backward Difference encoding, etc.

Here, we will use find and replace method.

1. *married* has two unique values, 'Yes' and 'No'. We will replace 'Yes' with a 1 and 'No' with a 0
2. *dependents* has three unique values, '0', '1', '2' and '3+'. We will replace '0' with a 0, 1 with a '1', '2' with a 2 and '3+' with a 3.
3. *self_employed* has two unique values, 'Yes' and 'No'. We will replace 'Yes' with a 1 and 'No' with a 0
4. *property_area* has three unique values, '*Rural*', '*Semiurban*' and '*Urban*'. We will replace '*Rural*' with a 0, '*Semiurban*' with with a 1 and '*Urban*' with a 2
5. *loan_status* has two unique values, 'Y' and 'N'. We will replace 'Y' with a 1 and 'N' with a 0

We will aggregate these columns into an *obj_df*, create *cleanup_nums*, where we will do the encoding, replace columns from the dataframe with *cleanup_nums*, and the assign columns from the *data_no_mv* dataset to columns in the *obj_df.*

Last, we will create a matrix of features and a matrix of dependent variable (X and y). Then, we will split the data using the train_test_split, with the 80:20 ratio.

Feature Scaling

To get better accuracy score, we will apply feature scaling to the *applicant_income*, *co_applicant_income*, *loan_amount* and *loan_amount_term*.

Logistic Regression

We will build Logistic Regression using the stats model library.

First, we must add a constant. Then, we will create a regression model:

1. Declare a regression variable called *reg_log*. For calculating the regression, we will use *Logit* that takes independent variable as the first argument and the dependent variable as the second.
2. Declare a variable where we will fit the regression: *results_log*.

Here is our message:

```
Optimization terminated successfully.
         Current function value: 0.465782
         Iterations 7
```

Optimization terminated successfully – this means that we managed to fit the regression. It took 7 iterations and current function value is 0.465782 - this refers to the idea that SM uses a machine learning algorithm to fit the regression. The function value shows the value of the 'objective function' at the 7th iteration. The reason of this message is that, after a certain number of iterations, there's always a possibility that the model won't learn the relationship. Therefore, it cannot optimize the optimization function. In SM, the maximum number of observations is 35. After that, it will stop trying.

Let's take a look at the summary.

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	479
Model:	Logit	**Df Residuals:**	468
Method:	MLE	**Df Model:**	10
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.2425
Time:	12:32:56	**Log-Likelihood:**	-223.11
converged:	True	**LL-Null:**	-294.53
Covariance Type:	nonrobust	**LLR p-value:**	1.106e-25

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.8963	0.923	-3.139	0.002	-4.705	-1.088
x1	0.5416	0.265	2.044	0.041	0.022	1.061
x2	-0.0120	0.132	-0.091	0.928	-0.272	0.248
x3	0.3647	0.295	1.238	0.216	-0.213	0.942
x4	-0.0900	0.351	-0.256	0.798	-0.778	0.598
x5	1.816e-05	2.47e-05	0.736	0.462	-3.02e-05	6.65e-05
x6	-3.841e-05	4.02e-05	-0.957	0.339	-0.000	4.03e-05
x7	-0.0008	0.002	-0.449	0.653	-0.004	0.003
x8	-0.0009	0.002	-0.419	0.675	-0.005	0.003
x9	3.9284	0.493	7.969	0.000	2.962	4.895
x10	0.0949	0.155	0.611	0.541	-0.210	0.400

x1 – married
x2 – dependents
x3 – education
x4 – self_employed
x5 – applicant_income
x6 – co_applicant_income
x7 – loan_amount
x8 – loan_amount_term
x9 – credit_history
x10 – property_area

MLE – Maximum likelihood estimation

- Based on the likelihood function
  - Likelihood function – a function which estimates how likely it is that the model at hand describes the real underlying relationships of the variables.
  - The bigger the likelihood function, the higher the probability that our model is correct
- MLE tries to maximize the likelihood function. The computer is going through different values, until it finds a model for which the likelihood is the highest. When it can no longer improve it, it will just stop the optimization

Log-Likelihood

- The value is almost but not always negative

- More popular metric

LL-Null

- The log-likelihood of a model which has no independent variables.

LLR

- Based on the log-likelihood of the model and the LL-Null. It measures if our model is statistically different from the LL-null (useless model)
- Based on our summary table, LLR p-value is 0.000000000000000000000000001106, which means that it is very low, around ~0.000, so, we can conclude that, our model is **significant**

Pseudo R-squared

- Since there is no such thing as a clearly defined R-squared for the logistic regression. There are several propositions (AIC, BIC and McFadden's R-squared) which have the similar meaning to the R-squared, but none of them is the same as the R-squared in linear regression.
- Here, we have the McFadden's R-squared. According to McFadden, 'A good Pseudo R-squared is somewhere between 0.2 and 0.4'. Here, we have the value of 0.2425

Accuracy

Using the results_log.predict(), we will get all the predicted values by the model.

```
array([0.83, 0.85, 0.86, 0.77, 0.71, 0.85, 0.73, 0.67, 0.06, 0.83, 0.83,
       0.85, 0.75, 0.88, 0.85, 0.38, 0.78, 0.77, 0.78, 0.83, 0.72, 0.81,
       0.79, 0.76, 0.85, 0.82, 0.86, 0.73, 0.62, 0.83, 0.80, 0.83, 0.83,
       0.78, 0.68, 0.81, 0.87, 0.87, 0.84, 0.81, 0.76, 0.78, 0.78, 0.08,
       0.07, 0.82, 0.74, 0.86, 0.77, 0.88, 0.83, 0.84, 0.83, 0.84, 0.89,
       0.06, 0.81, 0.06, 0.71, 0.78, 0.08, 0.86, 0.81, 0.78, 0.75, 0.77,
       0.78, 0.73, 0.69, 0.84, 0.77, 0.86, 0.83, 0.76, 0.84, 0.71, 0.84,
       0.86, 0.78, 0.75, 0.76, 0.04, 0.75, 0.79, 0.05, 0.67, 0.06, 0.81,
       0.82, 0.81, 0.79, 0.79, 0.82, 0.87, 0.80, 0.76, 0.84, 0.75, 0.83,
       0.06, 0.85, 0.77, 0.79, 0.75, 0.86, 0.78, 0.79, 0.08, 0.83, 0.85,
       0.73, 0.09, 0.07, 0.81, 0.86, 0.85, 0.84, 0.87, 0.83, 0.78, 0.90,
       0.71, 0.05, 0.75, 0.77, 0.06, 0.07, 0.71, 0.76, 0.85, 0.75, 0.76,
       0.87, 0.75, 0.84, 0.76, 0.81, 0.83, 0.08, 0.85, 0.73, 0.85, 0.78,
       0.10, 0.70, 0.66, 0.85, 0.85, 0.77, 0.70, 0.67, 0.07, 0.75, 0.81,
```

These values represent probabilities. In the model, these are the values of probability of mortgage being approved. Ultimately, values below 0.5 mean that there is a less than 50% chance of approving a mortgage, so we would round down (0). Alternatively, values above 0.5 would be rounded up(1)

Using this simplification, we can compare the actual values we observed with those predicted by the model.

```
array([1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
```

These are a lot of values to compare. We will summarize them in a table, using the method pred_table (results_log.pred_table()).

	Predicted 0	Predicted 1
Actual 0	62.0	84.0
Actual 1	5.0	328.0

The table represents the confusion matrix:

- For 62 observations, the model predicted 0 and the true value was 0
- For 328 observations, the model predicted 1 and the true value was 1
- For 5 observations, the model predicted 0 while the outcome was 1
- For 84 observations, the model predicted 1 while the actual was 0

The model made an accurate prediction in 390 out of the 479 cases. That gives us the accuracy 81.42%

Testing the Model

We will use our model to make predictions based on the test data, we will compare those with the actual outcome, calculate the accuracy and create a confusion matrix.

We will declare a new variable, x_test and add X_test as constant. Then, we will create confusion matrix. SM doesn’t provide this functionality, unlike scikit learn, so we will manually create a function confusion_matrix that has three values – data, actual_values and the model.

```
def confusion_matrix(data,actual_values,model):

        pred_values = model.predict(data)
        bins=np.array([0,0.5,1])
        cm = np.histogram2d(actual_values, pred_values, bins=bins)[0]
        accuracy = (cm[0,0]+cm[1,1])/cm.sum()
        return cm, accuracy
```

Our function will use the already created regression model to make predictions based on the data. Then, we will specify the bins and create a histogram where if values are between 0 and 0.5 will be considered 0. If they are between 0.5 and 1, they will be considered 1.

Then, it will summarize the values in a table. Finally, we will calculate the accuracy.

```
(array([[17.00, 23.00],
        [2.00, 78.00]]), 0.7916666666666666)
```

We see that, the accuracy is 79.16%. the test accuracy is the figure we use when we refer to overall accuracy. Almost always, the training accuracy is higher than the test accuracy. That's because of the overfitting, the regression fitted the training data the best as possible, but that doesn't mean that the prediction is true for all the values from the population.

	Predicted 0	Predicted 1
Actual 0	17.0	23.0
Actual 1	2.0	78.0

The opposite of accuracy is misclassification rate.

$$misslassification\ rate = \frac{\#misslassified}{\#all\ elements}$$

```
[181] print ('Missclassification rate: %.2f' %(((25/(17+23+2+78))*100)), "%")
```

```
Missclassification rate: 20.83 %
```

Our misclassification rate is 20.83%. This means that, 20.83% of the observations were incorrect.

Logistic Regression and K-fold cross validation

Let's see the accuracy using Logistic Regression and K-fold-cross validation.

First, we will train Logistic Regression on the training set. Then, we will predict the result and make a confusion matrix.

```
[186] y_pred = classifier.predict(X_test)
      cm = confusion_matrix(y_test, y_pred)
      a_s=accuracy_score(y_test, y_pred)
      a_s*=100
      print("Accuracy score: %.2f"%a_s)
```

```
Accuracy score: 79.17
```

We see that, our accuracy score is 79.17%, which is lower than accuracy we got using the logit function from the SM.

We will now use K-fold-cross validation to estimate the skill of the model. We will choose 10 for the value of k. So, we will have 10 train test folds ending up with 10 accuracies. We will use these accuracies to compute their average value.

```
accuracies = cross_val_score(estimator = classifier, X=X_train, y=y_train, cv=10)
print("Accuracy  (K-fold-cross validation): {:.2f}%".format(accuracies.mean()*100))
```

```
Accuracy  (K-fold-cross validation): 80.80%
```

Our new accuracy is slightly bigger than the previous one, but still smaller than the accuracy we got using the SM.

1.How would you assess the performances of your system?

Prediction accuracy is 81.42%. There are a lot of factors that affect the accuracy: number of features, how we deal with missing values, how do we clean the data, etc.

Expectations regarding predictive accuracy are a function of many factors including its importance as a unit of measure, the tactical objectives of the model, the information under analysis, feature measurement, and more.

The performances would be better if we used neural networks because neural networks models are more flexible.

2. How would you treat missing data and/or outliers (if any).

Entries where dependents.isnull() are dropped. The others are renamed according to the most common values and according to the median value of the column.

Outliers are found using the IQR. Everything that’s above upper bound is replaced with the mean value of the column.