Loan Prediction

Contents

Project Overview

Loans allow for growth in the overall money supply in an economy and open up competition by lending to new businesses.1 Thus, with the growing pace of emerging businesses, speeding up of the approval process for potential customers becomes very important.

In this project, I attempted to build a Machine Learning model based on 7 features that is capable of predicting whether an individual should be eligible for loan approval.

Problem Statement

The goal is to create a classifier that can predict whether an individual should be approved for loan or not; the tasks involved are the following:

i) Download and pre-process the data.
ii) Gain insights about the data, engineer features (If necessary) and make it ready for the machine learning algorithms.
iii) Train the data and test for evaluation metrics.
iv) Fine tune the hyperparameters.
v) Make predictions on the testing set.

Metrics

The most common metric for a binary classifier is Accuracy but it is not as robust as some other metrics like Balanced Accuracy, AUC-ROC. Therefore, we can use the following metrics to judge the performance of this classifier:

i) Balanced Accuracy (BAC) = 0.5*(Sensitivity + Specificity)
ii) Recall (Sensitivity) = Tells us what % of True positives were recalled.
iii) AUC = Area under curve of a ROC curve.

Data pre-processing

The two Data pre-processing steps that we will apply to our data set are:

i) Handling Missing values – Filling the holes in our training and testing dataset.
ii) Discretization – Converting continuous values into discrete categories.

[1] Loan Definition (investopedia.com)

The missing values per feature is shown in the table as follows:

```
# Arranging columns by proportion of null values
get_df_info(finance_train).sort_values(by='null_percent').loc[:, ['column', 'null_count', 'null_percent']]
```

	column	null_count	null_percent
0	Loan_ID	0	0.000000
4	Education	0	0.000000
6	ApplicantIncome	0	0.000000
7	CoapplicantIncome	0	0.000000
11	Property_Area	0	0.000000
12	Loan_Status	0	0.000000
2	Married	3	0.488599
1	Gender	13	2.117264
9	Loan_Amount_Term	14	2.280130
3	Dependents	15	2.442997
8	LoanAmount	22	3.583062
5	Self_Employed	32	5.211726
10	Credit_History	50	8.143322

Innovation: There are a lot of ways to handle missing values but to decrease the probability of any random error leakage into the training data, can we find and drop all the rows which have at least two feature values missing? Let's see.

Innovation-1: Can we find all the rows that have at least two feature values as null?

```
# Extracting the null indices of each feature

m = finance_train[finance_train.Married.isna()].index.to_list()
g = finance_train[finance_train.Gender.isna()].index.to_list()
lat = finance_train[finance_train.Loan_Amount_Term.isna()].index.to_list()
d = finance_train[finance_train.Dependents.isna()].index.to_list()
la = finance_train[finance_train.LoanAmount.isna()].index.to_list()
se = finance_train[finance_train.Self_Employed.isna()].index.to_list()
ch = finance_train[finance_train.Credit_History.isna()].index.to_list()

list_null_index = [m, g, lat, d, la, se, ch]

# To check if a row with all null values exists
print(set(m) & set(g) & set(la) & set(lat) & set(d) & set(se) & set(ch))
```

```
set()
```

```
# Trying to find the existence of all the rows with at least two missing values.

miss_index = []

for i in list_null_index:
    for j in list_null_index:
        if i !=j:
            x = set(i) & set(j)
            if len(x) != 0:
                if x not in list_null_index:              # Memoization
                    miss_index.append(x)
                else:
                    continue
print(list(miss_index))
```

```
[{104, 435, 228}, {435}, {460}, {335}, {19}, {104, 435, 228}, {335}, {435, 102}, {435}, {435, 102}, {95}, {95}, {19}, {95}, {600, 236, 24, 411, 30, 95}, {460}, {95}, {600, 236, 24, 411, 30, 95}]
```

The code above helps to find a list of set of indices containing at least two missing feature values. Now, we need to extract all the unique values from the list of sets.

```
# Extracting the unique indices form the list of sets.

indices_remove = []

for i in miss_index:
    for j in i:
        if j not in indices_remove:
            indices_remove.append(j)
        else:
            continue

# Let's look at the indices that from the dataframe that contain at least two missing values.
indices_remove
```

```
[104, 435, 228, 460, 335, 19, 102, 95, 600, 236, 24, 411, 30]
```

The above are the unique indices with at least two missing values. We can have a look at the rows for verification and then drop these 13 rows.

```
[9]: # Printing the rows from the DataFrame with at least 2 missing values.

print(len(indices_remove))
finance_train.iloc[indices_remove,:]
```

13

[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
104	LP001357	Male	NaN	NaN	Graduate	No	3816	754.0	160.0	360.0	1.0	Urban	Y
435	LP002393	Female	NaN	NaN	Graduate	No	10047	0.0	NaN	240.0	1.0	Semiurban	Y
228	LP001760	Male	NaN	NaN	Graduate	No	4758	0.0	158.0	480.0	1.0	Semiurban	Y
460	LP002478	NaN	Yes	0	Graduate	Yes	2083	4083.0	160.0	360.0	NaN	Semiurban	Y
335	LP002106	Male	Yes	NaN	Graduate	Yes	5503	4490.0	70.0	NaN	1.0	Semiurban	Y
19	LP001041	Male	Yes	0	Graduate	NaN	2600	3500.0	115.0	NaN	1.0	Urban	Y
102	LP001350	Male	Yes	NaN	Graduate	No	13650	0.0	NaN	360.0	1.0	Urban	Y
95	LP001326	Male	No	0	Graduate	NaN	6782	0.0	NaN	360.0	NaN	Urban	N
600	LP002949	Female	No	3+	Graduate	NaN	416	41667.0	350.0	180.0	NaN	Urban	N
236	LP001786	Male	Yes	0	Graduate	NaN	5746	0.0	255.0	360.0	NaN	Urban	N
24	LP001052	Male	Yes	1	Graduate	NaN	3717	2925.0	151.0	360.0	NaN	Semiurban	N
411	LP002319	Male	Yes	0	Graduate	NaN	6256	0.0	160.0	360.0	NaN	Urban	Y
30	LP001091	Male	Yes	1	Graduate	NaN	4166	3369.0	201.0	360.0	NaN	Urban	N

Exploratory Data Analysis

Since the number of features were small, we can begin the process of exploring the data feature by keeping these three questions in mind:

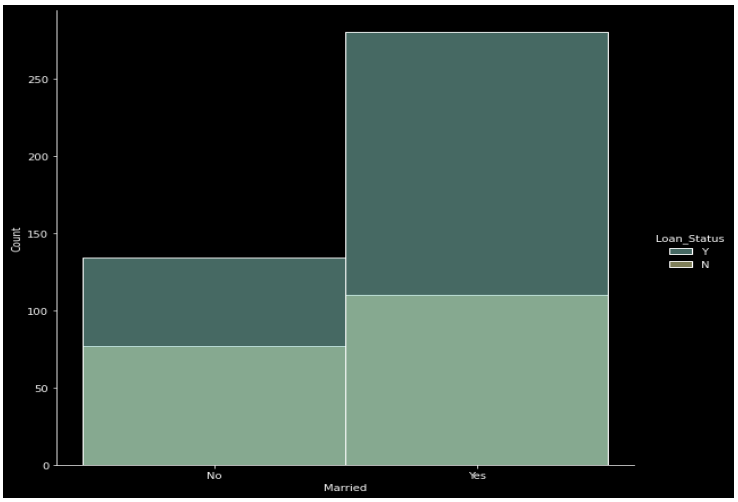
i) What does the distribution of each feature look like?
ii) If the variables are categorical, how much does each category impact the Loan status?
iii) If there are missing values, what should be the best strategy to impute them?

Feature-1: Loan ID:

This feature had all unique values and was thus should not be considered a part of our model building.

Feature-2: Married (Y/N)

i) There were a greater number of Married people in the training data.
ii) It's a categorical feature and on checking whether being married impacts the Loan status or not, we can observe this:

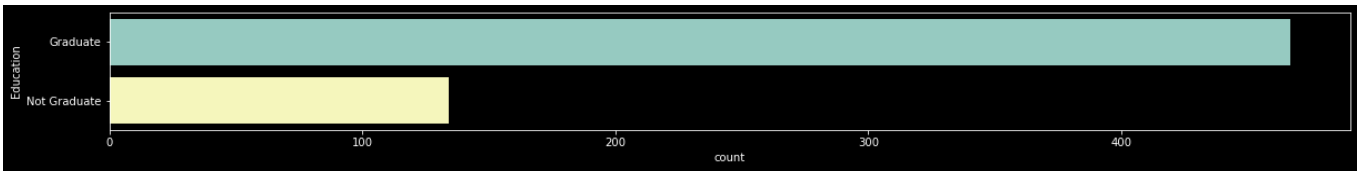


In hard numbers, 72% of the people who were **married** got approved whereas 65% people who were **not married** got approved indicating that being married has a significant impact on the loan status.

iii) There were no missing values in this feature.

Feature-3: Education (Graduate/Not-Graduate)

i) Majority of the applicants were graduates.



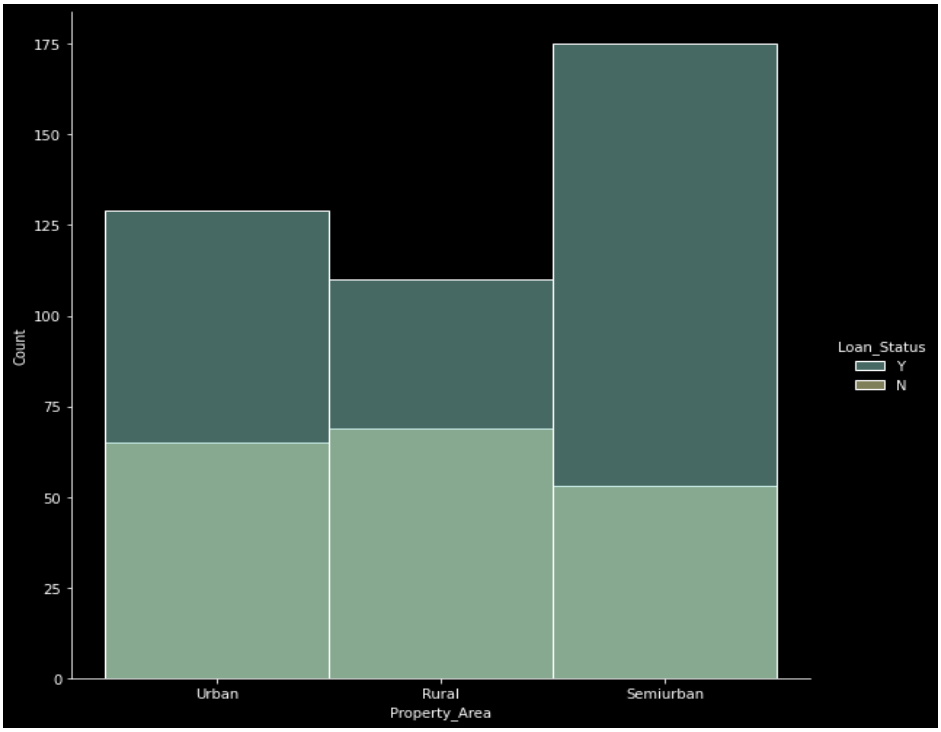
ii) This is a categorical feature as well and it also impacts the Loan approval status quite a bit.

iii) There were no missing values in this feature.

Feature-4: Property-Area (Urban/ Semi-Urban/ Rural)

i) Its distribution is somewhat linked to that of Education as people from Rural areas tend to have a lower chance of being Graduated. Rural and Urban populations did not have a huge difference whereas the Semi-urban population was the highest.

ii) This is again a categorical feature and its relation to the loan being approved is quite interesting. We would expect people from the Urban population to have the highest chance of Loan approval but the training data showed this:

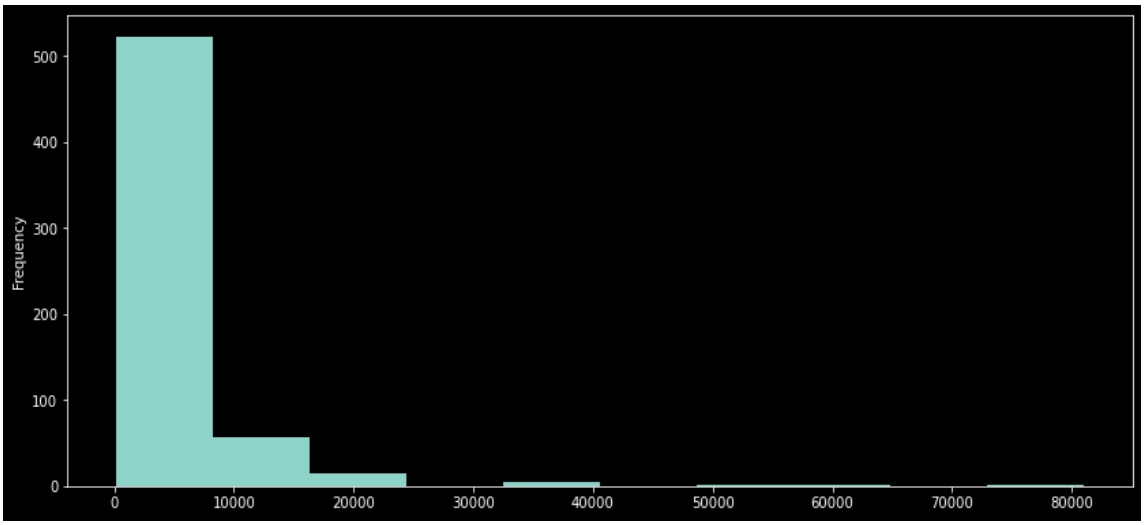


Therefore, we observe that the semi-urban population were the ones with the highest chances of being approved. Quite an interesting insight.

iii) There were no missing values in this feature.

Feature-5: Applicant Income

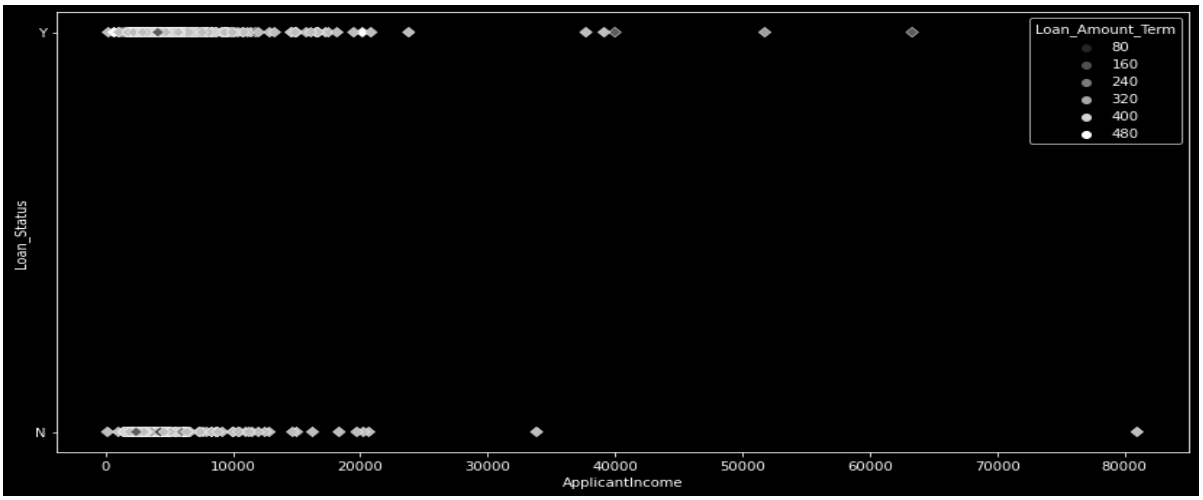
i) The distribution of income was somewhat right skewed.



On seeing hard numbers, Income appears to have some outliers. 75% of the income is below 5780.

```
[29]: finance_train.ApplicantIncome.describe()

[29]: count      601.000000
      mean      5404.632280
      std       6155.485943
      min        150.000000
      25%       2876.000000
      50%       3775.000000
      75%       5780.000000
      max      81000.000000
      Name: ApplicantIncome, dtype: float64
```

ii) It's impact on the Loan Status is shown below



iii) There were no missing values in the Applicant Income.

Feature-6: Co-applicant Income

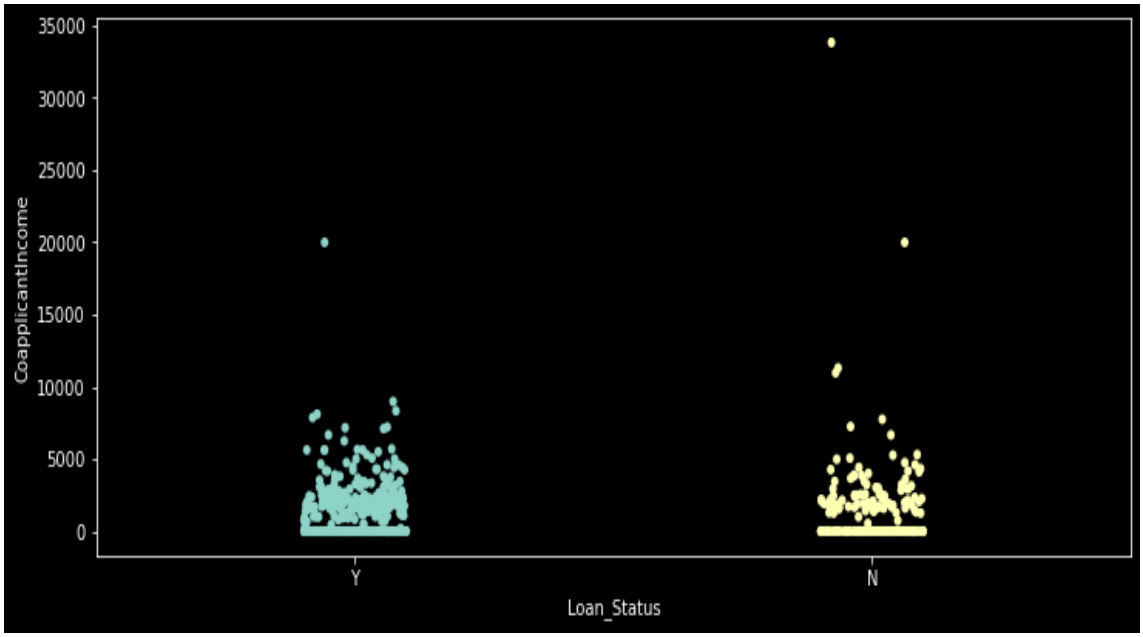
i) There were a lot of unique values in this feature.

ii) It's a continuous in nature. Let's visualize its distribution in terms of proportion.

```
finance_train.CoapplicantIncome.value_counts(normalize=True).head(10)
```

```
0.0       0.444260
1666.0    0.008319
2083.0    0.008319
2500.0    0.008319
1750.0    0.004992
2333.0    0.004992
1625.0    0.004992
5625.0    0.004992
1800.0    0.004992
1459.0    0.004992
Name: CoapplicantIncome, dtype: float64
```

We observe that about <u>44% of the co-applicants have a 0 income</u>. So, the next sensible thing to do would be to check how various co-applicant incomes impact the Loan approval status.



Therefore, <u>there was no visible change in the approval status with the increase in co-applicant income.</u> We can use this insight to engineer a feature which is covered in the next section.

iii) There were no missing values in this feature.

Feature-7: Gender (Male/Female)

i) About 80% of the applicants were male.

ii) <u>Gender didn't impact the loan status</u>.

```
male_approve, female_approve = feature_approval_proportion(df_finance, col='Male')
print("Males approved: {} %\nFemales  approved: {} %".format(male_approve, female_approve))
```

```
Males approved: 69.05737704918032 %
Females  approved: 68.14159292035397 %
```

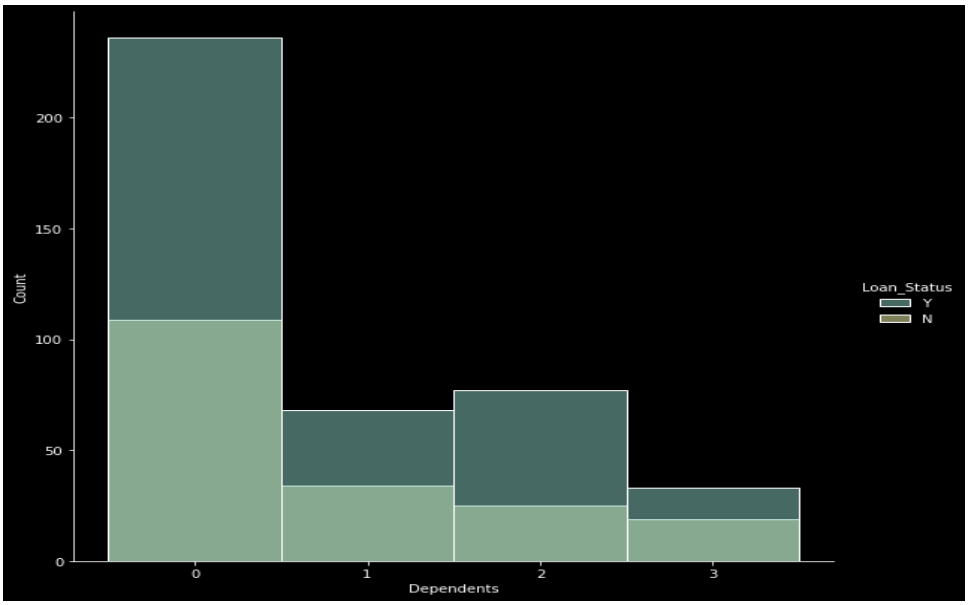
Therefore, Loan_Status is not very dependent on Gender. 'Advantage:' Even if the actual null values are in a different proportion from what we imputed it wouldn't make much of a difference.

iii) Gender had some missing values. Since it's a categorical feature which is highly skewed towards the male population, imputing the missing values in the 80-20 male-female ratio appears to be the best way to proceed.

Feature-8: Dependents (0,1,2,3+)

i) Majority of individuals applying for Loan had 0 dependents.

ii) Individuals with 3+ dependents had the least chance of getting their loans approved. Let's see the result in hard numbers.



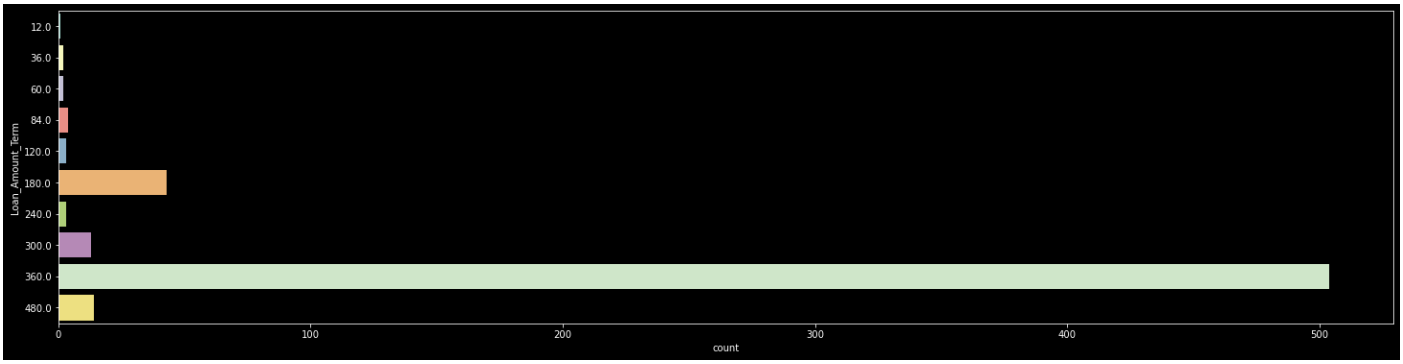
About 75% people with exactly 1 dependent got approved, which was the highest. This may indicate that having 1 dependent is ideal to be granted a loan.

```
zero, two  = finance_train.Dependents.value_counts()[0], finance_train.Dependents.value_counts()[1]
one, three = finance_train.Dependents.value_counts()[2], finance_train.Dependents.value_counts()[3]

zero_p = (finance_train[(finance_train.Dependents == '0')]['Loan_Status'] == 'Y').sum() / zero
two_p = (finance_train[(finance_train.Dependents == '1')]['Loan_Status'] == 'Y').sum() / two
one_p = (finance_train[(finance_train.Dependents == '2')]['Loan_Status'] == 'Y').sum() / one
three_p = (finance_train[(finance_train.Dependents == '3')]['Loan_Status'] == 'Y').sum() / three

print("zero dependents approved:  {}\nOne dependent approved:    {}\nTwo Dependents approved:   {}\nThree dependents approved: {}"
```

```
zero dependents approved:  0.6840579710144927
One dependent approved:    0.7549019607843137
Two Dependents approved:   0.6666666666666666
Three dependents approved: 0.6346153846153846
```

iii) This is again a categorical feature and missing values can be imputed by replacing them with values of Dependents in the same proportions as they appear in the training set.

Feature-9: Loan Amount Term



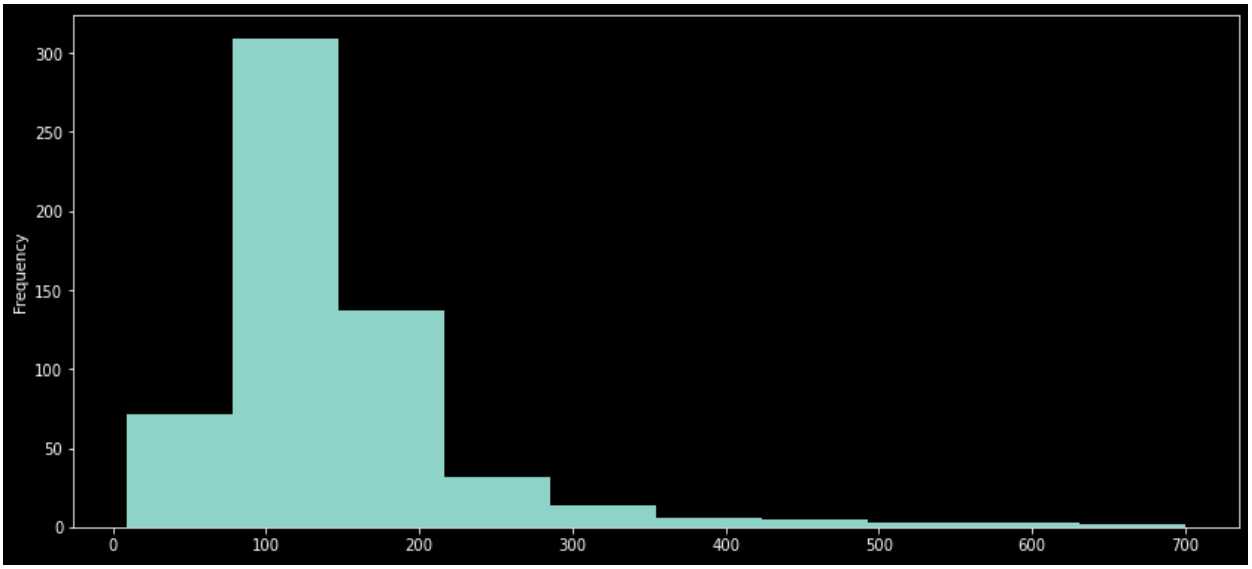
i) Loan Amount term has got 10 unique values with about 80% of the values skewed towards 360 months.

ii) There does not seem to be any significant difference between receiving the approval status if to loans are taken for a different tenure. Therefore, this may not be an important feature.

iii) This is a categorical feature as well and the missing values can be randomly filled with top 2 most frequently occurring values in the ratio 80:20.

Feature-10: Loan Amount

i) This is a continuous feature with the following distribution.



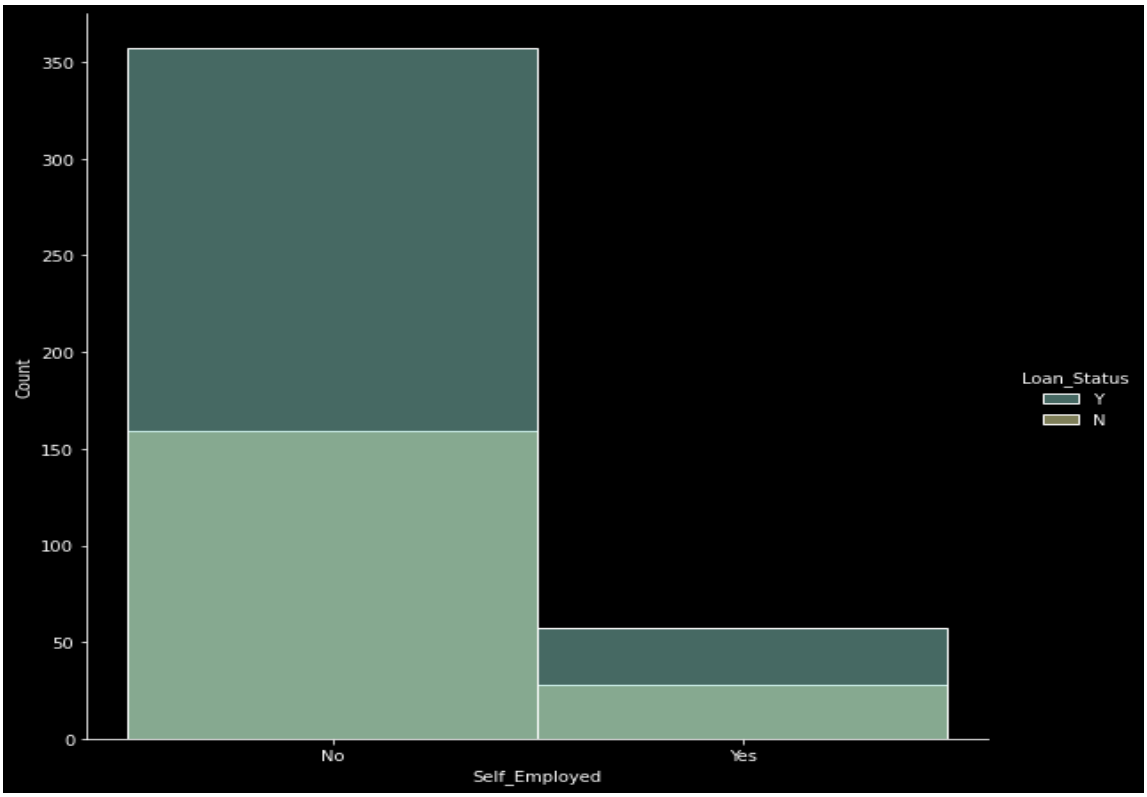
ii) The Loan Amount is almost Normal with some outliers indicated by the long tail. We need to handle this in the data wrangling part.

iii) The null values could be replaced by the median of the distribution as outliers don't impact the median.

Feature-11: Self Employed

i) Only about 14% of the applicants are self-employed.

ii) Visualizing the impact of Loan approval status on being self-employed.



The visualisation is unclear. Therefore, let's see the hard numbers.

```
# Let's see the proportion of Self_employed people whose loan status was approved.

self_approve, not_self_approve = feature_approval_proportion(df_finance, col='Self_Employed')
print("{} % people who were Self-Employed were approved\n{} % people who were not Self-Employed were approved".format(self_approve
```

```
67.05882352941175 % people who were Self-Employed were approved
69.18604651162791 % people who were not Self-Employed were approved
```

<u>There was not much difference between approval being granted based on the Self-employed status</u>. Therefore, this may not be an important feature.

iii) This is a categorical variable and the missing values can be imputed in the same proportion as they appear in the training set.

Feature-12: Credit History

i) Majority of values were 1 meaning that a credible credit history was available

```
finance_train.Credit_History.value_counts(dropna=False)

1.0    469
0.0     89
NaN     43
Name: Credit_History, dtype: int64
```

ii) From the description, this looks like the most important feature of all but this was also the feature with the most missing values. One way to approximate the dependency of Credit History on Loan Status is to impute the missing values once by 0 , once by 1 and compute the correlation coefficient for both the imputations.

```
print("After filling with 0: ",np.corrcoef(finance_train.Credit_History.fillna(0), df_finance.Loan_Status)[0][1].round(4))

print("After filling with 1: ",np.corrcoef(finance_train.Credit_History.fillna(1), df_finance.Loan_Status)[0][1].round(4))

After filling with 0:  0.4248
After filling with 1:  0.5495
```

iii) The above piece of code shows that the correlation between Loan Status and Credit History cannot be ignored and hence imputing the missing values with random values can possibly mess up with the pattern. Therefore, the best technique here would be to drop the rows which has missing values of Credit History.

Data Wrangling and Feature Engineering

1. The data set has 8 categorical features and 3 continuous features.

Variables Description

Loan_ID	: Unique Loan ID
Gender	: Applicant is Male/Female
Married	: Applicant married - Yes/No
Dependents	: Number of dependents
Education	: Applicant Education - Graduate/ Under Graduate
Self_Employed	: Self employed - Yes/No
ApplicantIncome	: Applicants total income
CoapplicantIncome	: Coapplicants total income
LoanAmount	: Loan amount
Loan_Amount_Term	: Term of loan in months
Credit_History	: Credit history meets guidelines - 1(Yes)/ 0(No)
Property_Area	: Urban/ Semi Urban/ Rural
Loan_Status	: Loan approved - Yes/No

2. Binary features like Gender, Married, Education, Self-employed, Credit-History, Loan status were all One Hot encoded. Example: Gender was encoded 1 if the applicant was male, 0 otherwise.
3. A person with 1 dependent had the highest chance of getting their loan approved. Therefore, while encoding maximum weight was given to applicants with 1 dependent.

Therefore, we need to encode the Dependents accordingly giving most weight to 1 dependent followed by 0, 2 and 3.

```
df_finance['Dependents'] = finance_train['Dependents']
df_finance['Dependents'] = df_finance.Dependents.replace({'1': 3, '0': 2, '2': 1, '3': 0})
df_finance.head()
```

	Loan_Status	Married	Graduate	Property_Area	Applicant_Income_Category	Coapplicant_Income_exists	Male	Dependents
0	1	0	1	2	4	0	1	2
1	0	1	1	1	4	1	1	3
2	1	1	1	2	2	0	1	2

4. Applicant Income had outliers. So, discretization of Applicant Income, based on their sample quantiles, into 5 categories was a valid solution.

```
[29]: finance_train.ApplicantIncome.describe()

[29]: count      601.000000
      mean      5404.632280
      std       6155.485943
      min        150.000000
      25%       2876.000000
      50%       3775.000000
      75%       5780.000000
      max      81000.000000
      Name: ApplicantIncome, dtype: float64
```

75% of the Applicants have an income <5780. The maximum income is 81000 indicating a possibility of outliers with respect to the Loan_Status. Therefore, we can discretize the ApplicantIncome by dividing it into 5 categories where Category 1 is the lowest income group and Category 5 is the highest income group.

```
[30]: # Discretizing the Applicant_Income into 5 categories and adding it to our new DataFrame

      df_finance['Applicant_Income_Category'] = pd.qcut(finance_train.ApplicantIncome, 5, labels=[1,2,3,4,5])
      df_finance.head()
```

5. Loan Amount was almost Normally distributed but it had few outliers which we had to deal with. Therefore, after filling the null spaces with the median of the distribution, the feature Loan Amount was also, based on the sample quantiles, discretized into 5 categories.

6. Loan Amount term had only 10 unique values in the form of the number of years. Therefore, it was label encoded.

Label Encoding the Loan_Term

```
[53]: encode = LabelEncoder()
      df_finance['Loan_Term_Years_encoded'] = pd.Series(encode.fit_transform(finance_train['Loan_Term_Years']))
      df_finance.head(3)
```

[53]:

	Loan_Status	Married	Graduate	Property_Area	Applicant_Income_Category	Coapplicant_Income_exists	Male	Dependents	Loan_Term_Years_encoded
0	1	0	1	2	4	0	1	2	8
1	0	1	1	1	4	1	1	3	8
2	1	1	1	2	2	0	1	2	8

7. EDA of Property Area showed that individuals living in semi-urban areas were more likely to get their loans approved. Therefore, most weight was given to the applicants coming from semi-urban areas.

From the above graph, we observe that people from Semiurban area have a greater chance of Loan_approval. Therefore, while encoding Property_Area into numbers, we can give the most weight to Semiurban.

[26]:
```
# Encoding 1 for Rural, 2 for Urban and 3 for Semiurban and adding it to the new DataFrame

df_finance['Property_Area'] = finance_train['Property_Area'].replace({'Semiurban': 3,
                                                                      'Urban': 2,
                                                                      'Rural': 1})
df_finance.head()
```

[26]:

	Loan_Status	Married	Graduate	Property_Area
0	1	0	1	2
1	0	1	1	1
2	1	1	1	2
3	1	1	0	2
4	1	0	1	2

8. Finally, the last feature i.e., Co-applicant Income. This feature provided some pretty interesting insights as mentioned in the EDA part. Because the increase in co-applicant income did not play a big role in increasing the chances of approval of an applicant combined with the fact that 44% of the values were exactly 0, **we can engineer a new feature and call it "*co-applicant income exists*"** which is 1 if the co-applicant income is non-zero, 0 otherwise.

Feature Engineering : There doesn't seem to be much increase in the probability of Loan getting approved as the CoapplicantIncome increases. Therefore, we create a new feature which is 1 if CoapplicantIncome is non-zero, 0 otherwise.

[35]:
```
df_finance['Coapplicant_Income_exists'] = finance_train['CoapplicantIncome']
df_finance['Coapplicant_Income_exists'] = np.where(df_finance.Coapplicant_Income_exists == 0, 0, 1)
df_finance.head()
```

[35]:

	Loan_Status	Married	Graduate	Property_Area	Applicant_Income_Category	Coapplicant_Income_exists
0	1	0	1	2	4	0
1	0	1	1	1	4	1
2	1	1	1	2	2	0
3	1	1	0	2	1	1
4	1	0	1	2	4	0

Algorithms

The algorithms used in this project are the following:

i) Random Forests
ii) Logistic Regression
iii) Support Vector Classifier
iv) Gaussian Naïve Bayes[2]

[2] The Optimality of Naive Bayes (aaai.org)

The number of observations in the training set were pretty small so it made more sense to choose **algorithms with a high bias and low variance** (ii, iii, iv)

Random Forests serve as a comparison model to see how they perform against the other three.

- Initially, all 11 features were used to evaluate the performance of the algorithms.
- Once, we got the top 2 performing models from the first iteration, their feature importance values were used to create a second feature set

[82]:

features	importance_magnitude_lr
Credit_History	3.2995
Property_Area	0.4091
Coapplicant_Income_exists	0.3981
Graduate	0.3312
Married	0.2540
Dependents	0.2249
Loan_Term_Years_encoded	0.1444
Male	0.1077
Loan_Amount	0.0465
Self_Employed	0.0299
Applicant_Income_Category	0.0268

[83]:
```
# Choosing the Top 7 features

feature_set_2 = lr_feature_imp.index[0:7].to_list()
feature_set_2
```

[83]:
```
['Credit_History',
 'Property_Area',
 'Coapplicant_Income_exists',
 'Graduate',
 'Married',
 'Dependents',
 'Loan_Term_Years_encoded']
```

.

- Finally, a third feature set was created based on the correlation analysis of all 11 features. The features that were least corelated with Loan status were dropped and the ones that were highly corelated among themselves were also taken care of.

[85]: <AxesSubplot:>

[86]:
```
# Removing less correlated and non-linear features
feature_set_3 = df_finance_train.drop(columns=['Loan_Term_Years_encoded', 'Married', 'Male', 'Dependents']).columns.to_list()
feature_set_3
```

[86]:
```
['Graduate',
 'Property_Area',
 'Applicant_Income_Category',
 'Coapplicant_Income_exists',
 'Loan_Amount',
 'Self_Employed',
 'Credit_History']
```

Hyperparameter tuning

After experimenting with 4 algorithms on 3 different feature sets, the following table evaluation table was obtained.

	model_name	feature_count	feature_names	ACC	BAC	REC	AUC
9	RFC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.783	0.724	0.889	0.724
6	LRC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.815	0.723	0.982	0.723
1	LRC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.814	0.721	0.979	0.721
3	SVC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.814	0.720	0.982	0.720
5	GNB_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
7	LRC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
10	SVC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.814	0.720	0.982	0.720
11	SVC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
0	GNB_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.808	0.716	0.974	0.716
4	GNB_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.808	0.716	0.974	0.716
8	RFC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.787	0.703	0.937	0.703
2	RFC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.765	0.697	0.887	0.697

The Top-3 models, when arranged in descending order of their balanced accuracies are:

- Random Forests with feature set 3
- Logistic Regression with feature set 2
- Logistic Regression with feature set 1

The third model requires all 11 features whereas the top 2 require only 7. Therefore, we can safely ignore Logistic Regression with all features and move towards tuning the hyperparameters of the top two models.

We can use **GridSearchCv** to fine tune the models. The following were the parameter grids for both the models.

i) Logistic Regression

```
param_grid_lr = { 'C': [0.001, 0.01, 0.1, 0.5, 1, 5, 10],
                  'class_weight': [None, 'balanced'],
                  'solver': ['liblinear', 'newton-cg']
}

grid_search_lr = GridSearchCV(LR(random_state=23), param_grid=param_grid_lr, cv=10, scoring='balanced_accuracy')
grid_search_lr.fit(df_finance_train[feature_set_2], label)
```

ii) Random Forest

```
param_grid_rfc = {'criterion': ['gini', 'entropy'],
                  'bootstrap': [True, False],
                  'class_weight': [None, 'balanced'],
                  'max_depth': [None, 10, 50, 100, 500],
                  'min_samples_split': [3,4,5,6],
}

grid_search_rfc = GridSearchCV(RFC(random_state=23), param_grid=param_grid_rfc, cv=10, scoring='balanced_accuracy')
grid_search_rfc.fit(df_finance_train[feature_set_3], label)
grid_search_rfc.best_params_
```

```
{'bootstrap': True,
 'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': None,
 'min_samples_split': 3}
```

After getting the tuned hyperparameters that fit the training data well, we can test the evaluation scores again to check for the improved values of our performance metrics

	model_name	feature_count	feature_names	ACC	BAC	REC	AUC
13	RFC_f3_tuned	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.792	0.742	0.881	0.742
12	LR_f2_tuned	7	[Credit_History, Property_Area, Coapplicant_In...	0.789	0.732	0.889	0.732
9	RFC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.783	0.724	0.889	0.724
6	LRC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.815	0.723	0.982	0.723
1	LRC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.814	0.721	0.979	0.721
3	SVC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.814	0.720	0.982	0.720
5	GNB_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
7	LRC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
10	SVC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.814	0.720	0.982	0.720
11	SVC_f3	7	[Graduate, Property_Area, Applicant_Income_Cat...	0.814	0.720	0.982	0.720
0	GNB_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.808	0.716	0.974	0.716
4	GNB_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.808	0.716	0.974	0.716
8	RFC_f2	7	[Credit_History, Property_Area, Coapplicant_In...	0.787	0.703	0.937	0.703
2	RFC_f1	11	[Married, Graduate, Property_Area, Applicant_I...	0.765	0.697	0.887	0.697

So, our GridSearch actually managed to increase the Balanced Accuracy of the Top-2 models but that happened at the cost of Recall scores.

Result

Judgement: Both Recall and Balanced accuracy are our evaluation metrics. Looking at the final table, I would choose LRC_f2 or Logistic Regression with feature set 2 to be the model for making predictions on the testing data.

Reason: The balanced accuracy score of the topmost model is **0.742** which is higher when compared to **0.723**; the balanced accuracy score of the model of my choice. But the increase is a mere **2%** and for a small testing set of size 347, this increase is pretty insignificant particularly when we factor in the **dip in Recall score values which is about 11%.**

Therefore, I made the predictions on the testing set using a **logistic regression model with feature set 2**.