Loan Prediction

Planning Analytics

Understand the status of the business

Introduction

- Loan eligibility is defined as a set of criteria basis which a financial institution evaluates to decide the eligibility of a customer for a particular loan
- Loans are the core business of banks and their main profit comes directly from the loan’s interest

Persona

Name: Kelvin
Age: 30s
Location: Yishun
Education: Computer Science Degree (NUS)
Job: Singapore-based Youtuber (Channel 'Kelvin Learns Investing')
Family: Married with 1 Kid
Working Exp: Former game developer turned full-time Youtuber

" How I wish I can buy a house with easy loan approval "

Goals (what to accomplish)
- get loan at the best possible rate and best deal

- able to secure loan within 1-2 days

Motivations (what motivates)
- to buy a house as soon as possible

Needs (unfulfilled today)
- having a peace of mind when applying loan

What is the business problem?

Loan eligibility is decided after long and intensive process of verification of documents and validation which takes a huge amount of time

Loan Application

Document Submission

Document Verification

Loan Approval

HMW Statement

How might we

Improve the housing loan application process

For

time-starved working professionals

So that

the process of securing the loan is faster and hassle-free

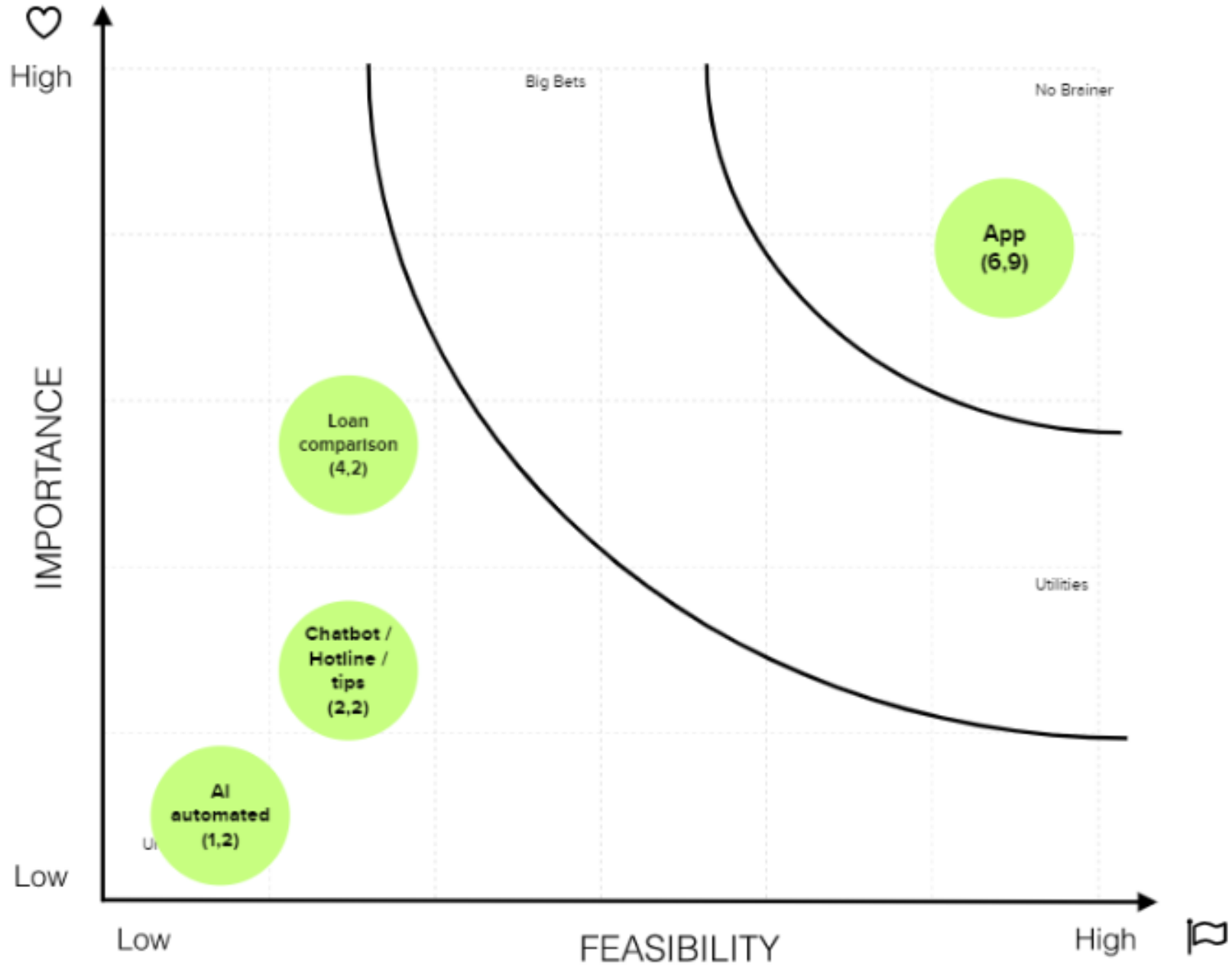
Pain points


Finding the process tedious and time consuming
Troublesome to submit all the details
Worried on not able to secure the loan in time
Frustrating to come up with alternative plan

What is the monetization model?

- Referral Fees
- Listing Fees From Lenders (eg banks, financial institutions)
- Advertising Fees (eg private developers, contractors, ID)

Prioritization Grid



Tools

- **Mural**
- **Jupyter Notebook**
- **Dataspell (Integrated Development Environment)**
- **Python (Programming Language)**
- **Tableau**
- **Github Cloud Storage**
- **IBM Watson**
- **Auto AI**
- **Google Colaboratory**

Libraries

- **Pandas** – Powerful/flexible library for Data Analysis, Manipulation, Filtering
- **Matplotlib** – Comprehensive library for creating visualizations
- **Seaborn** – Library used for statistical plotting (Built on top of Matplotlib)
- **Numpy** – Used to perform mathematical and logical operations on arrays
- **Scikit-Learn** – Features various classification, regression and clustering algorithms
- **IPython.display** – To display images in python
- **XGBoost** – XGBClassifier Library
- **AutoGluon** – Enables developers to write machine learning-based application

Why Python?

- Easy Structure and Syntax
- Succinct and readable code
- Extensible and portable programming language
- Provides various libraries which play a crucial role in data science, data visualization, data analytics etc

Why DataSpell?

- Intelligent coding assistance
- Able to run entire or part of Python scripts
- Built-in error debugger
- Ability to connect to multiple remote instances of Jupyter servers

Why Tableau?

- Powerful and fastest growing data visualization
- Most often used in Business Intelligence Industry
- Helps to simplify raw data in an easily understandable format
- Can be understood by professionals at any level in an organization

Data Source

Chosen

- https://github.com/ParthS007/Loan-Approval-Prediction
- Information on the bank loan data and python source in csv file

Rejected

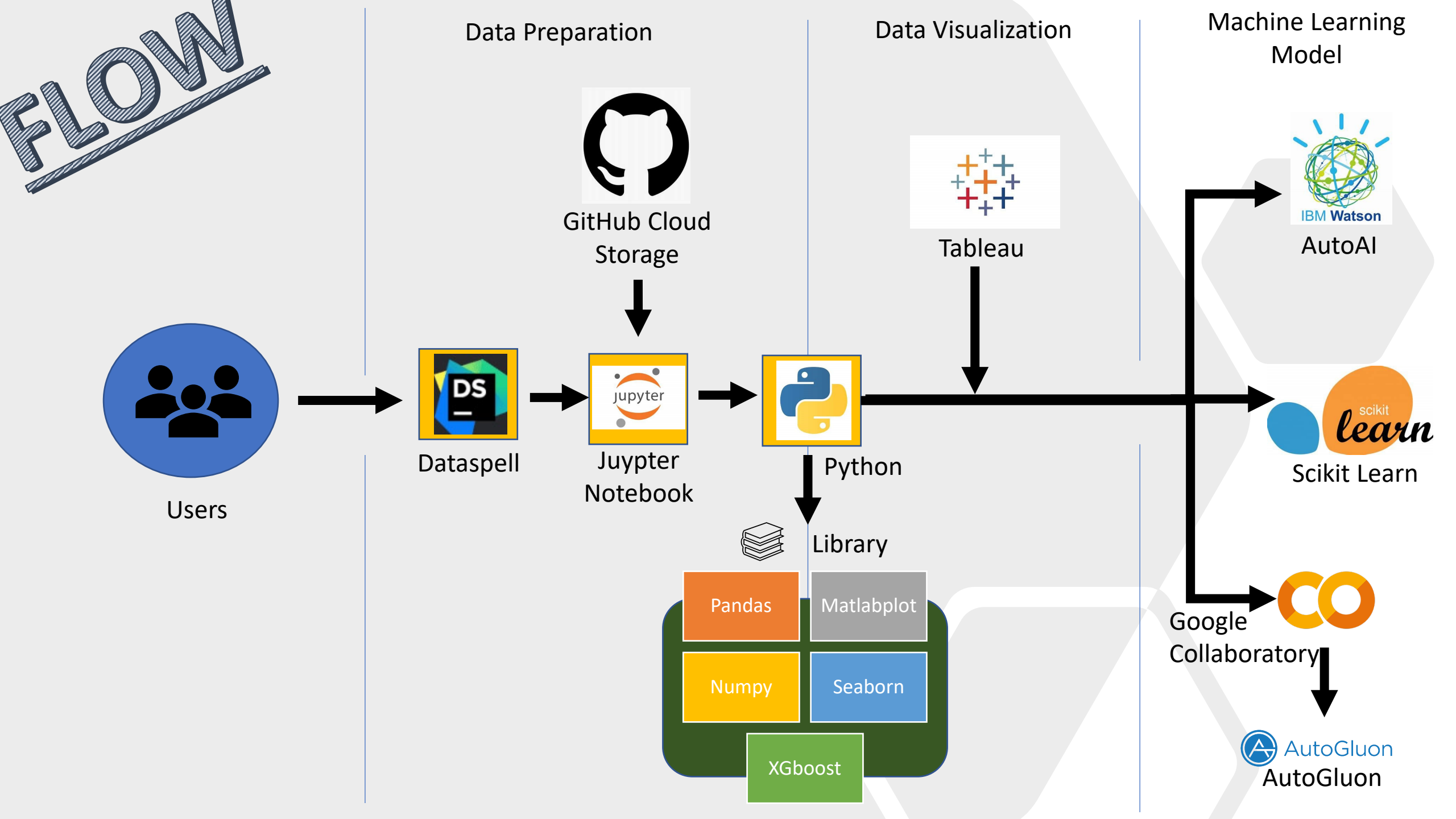
- https://www.kaggle.com/matthew2001/bank-loan-data?select=loan_table.csv
- It doesn't have the loan amount borrowed which will hinder our prediction

Rejected

- https://www.kaggle.com/caiocampiao/loan-approval-systemlas
- The columns are not in English after further investigation

Descriptive Analytics

Cleaning, Relating, Summarize, Visualization


FLOW
Data Preparation
Data Visualization
Machine Learning Model
GitHub Cloud Storage
Tableau
IBM Watson
AutoAI
Users
DS
Dataspell
jupyter
Juypter Notebook
Python
scikit learn
Scikit Learn
Library
Pandas
Matlabplot
Numpy
Seaborn
XGboost
Google Collaboratory
AutoGluon
AutoGluon

Data Preparation

Importing the Dependencies

```
import collections
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, accuracy_score, confusion_matrix, classification_report
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Data Preparation

Reading train.csv & test.csv through Github Cloud Storage

```
train = pd.read_csv("https://github.com/Lightgamers/Loan/raw/main/train.csv")
train.head()
```

```
test = pd.read_csv("https://github.com/Lightgamers/Loan/raw/main/test.csv")
test.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
0	LP001002	Male	No	0	Graduate	No
1	LP001003	Male	Yes	1	Graduate	No
2	LP001005	Male	Yes	0	Graduate	Yes
3	LP001006	Male	Yes	0	Not Graduate	No
4	LP001008	Male	No	0	Graduate	No

ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
5849	0.0	NaN	360.0	1.0	Urban	Y
4583	1508.0	128.0	360.0	1.0	Rural	N
3000	0.0	66.0	360.0	1.0	Urban	Y
2583	2358.0	120.0	360.0	1.0	Urban	Y
6000	0.0	141.0	360.0	1.0	Urban	Y

Data Preparation

Checking if there is any identical Loan_ID appears in both train and test dataframe

```
#Checking if theres any with same load_id in both train and test
train_id = list(train['Loan_ID'])
test_id = list(test['Loan_ID'])

if collections.Counter(train_id) == collections.Counter(test_id):
    print("There are records with similar ID")
else:
    print("Different sets of load ids are in both of the datasets")
```

```
Different sets of load ids are in both of the datasets
```

Data Preparation

Checking if there is any duplicates of Loan_ID in either train or test dataset

```
#Checking for duplicates of Loan_ID in train dataset
duplicate_check_train = train.shape[0] - train.Loan_ID.nunique()

print(f'There are {duplicate_check_train} duplicates for load id in the train dataset')
```

```
There are 0 duplicates for load id in the train dataset
```

```
#Checking for duplicates of Loan_ID in test dataset
duplicate_check_test = test.shape[0] - test.Loan_ID.nunique()

print(f'There are {duplicate_check_test} duplicates for load id in the test dataset')
```

```
There are 0 duplicates for load id in the test dataset
```

Data Preparation

Comparing differences between the two datasets

12 independent variables and 1 dependent variable ‘Loan_Status’

```
train.columns
```

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

13 columns

```
test.columns
```

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

12 columns

Data Preparation

Checking the number of rows and columns for each datasets

```
train.shape
```

```
(614, 13)
```

```
test.shape
```

```
(367, 12)
```

Train - 614 rows, 13 columns

Test – 367 rows, 12 columns

Data Preparation

To find out the Descriptive Statistics that summarize the central tendency, dispersion and shape of the dataset

Count

Mean

```
train.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

8 rows × 5 columns Open in new tab

Standard Deviation

Minimum

Maximum

Data Preparation

Checking the data types format of each variables

3 formats of the datatype used

object = String, Boolean Variables (Categorical , Nominal/Ordinal)

int64 = Integer Variables (Numerical)

float64 = Float Variables (Numerical)

```
1 train.dtypes
```

	data
Loan_ID	object
Gender	object
Married	object
Dependents	object
Education	object
Self_Employed	object
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	object
Loan_Status	object

Length: 13, dtype: object

Attributes

Variable	Description	Data Type
Loan_ID	Unique Loan ID identifier	String
Gender	Male/Female	String
Married	Applicant married (Y/N)	String
Dependents	Number of dependents	String
Education	Applicant Education (Graduate/Under Graduate)	String
Self_Employed	Self employed (Y/N)	String
ApplicantIncome	Applicant's Income	Integer
CoapplicantIncome	Co-Applicant's Income	Float
LoanAmount	Loan amount in thousands	Float
Loan_Amount_Term	Term of loan in months	Float
Credit_History	credit history meets guidelines	Float
Property_Area	Urban/Semi Urban/Rural	String
Loan_Status	Loan approved (Y/N)	String

About Data Types

Grouping variables according to Data Type

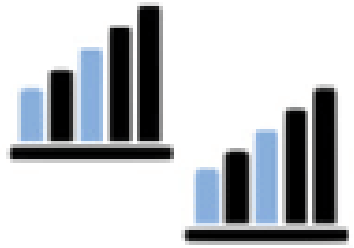
Nominal (Categorical)	Ordinal (Categorical)	Numerical
• Gender • Married • Self Employed • Credit History	• Dependents • Property Area • Education	• Applicant Income • Co-applicant Income • Loan Amount • Loan Amount Term

Exploratory Data Analysis(EDA)

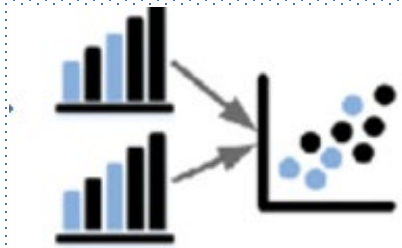
Load Data

STEP 1:
Distinguish Attributes

STEP 2:
Univariate Analysis



Univariate analysis explores variables (attributes) one by one



Bivariate Analysis

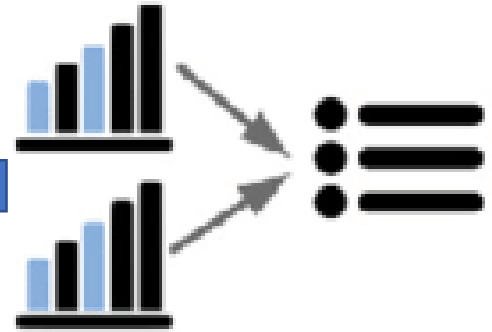
Explores the concept of relationship between two variables, whether there exists an association and the strength of this association. Or whether there are differences between two variables

STEP 3:
Detect Interactions Among Attributes

STEP 4:
Detect Abnormal & Missing Values

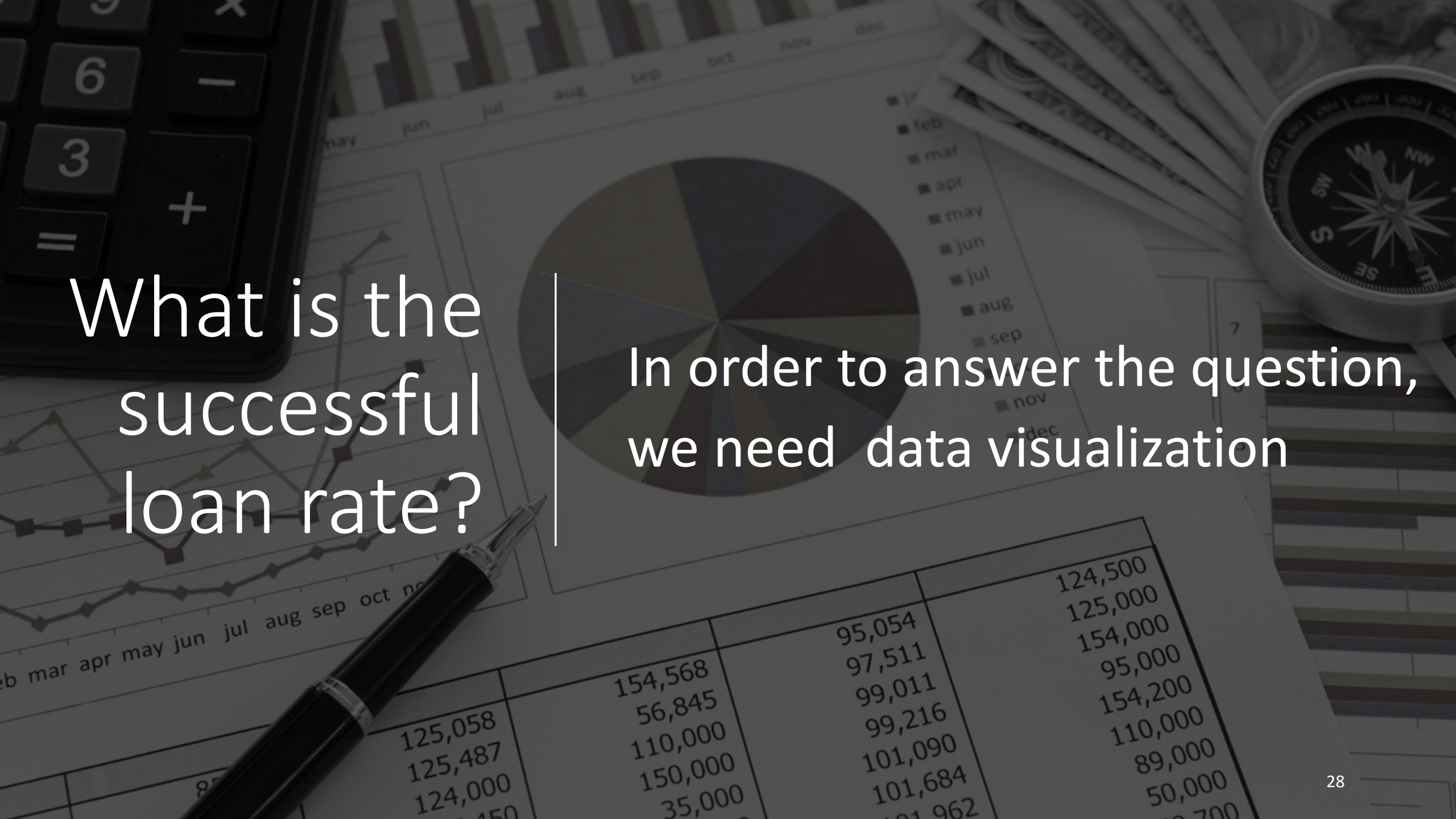
An outlier is a value or point that differs substantially from the rest of the data

STEP 5:
Detect Outliers



STEP 6:
Feature Engineering

Collect insight

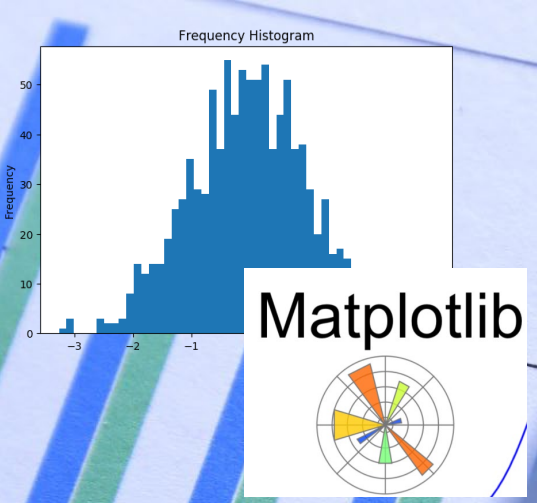


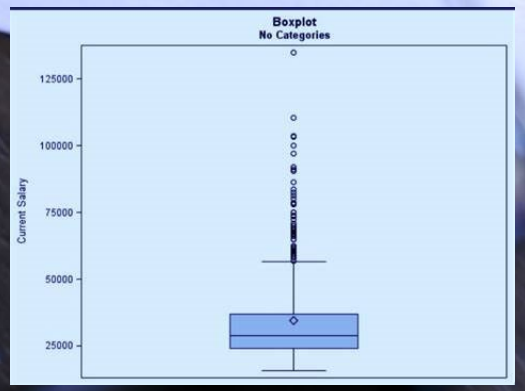
What is the successful loan rate?

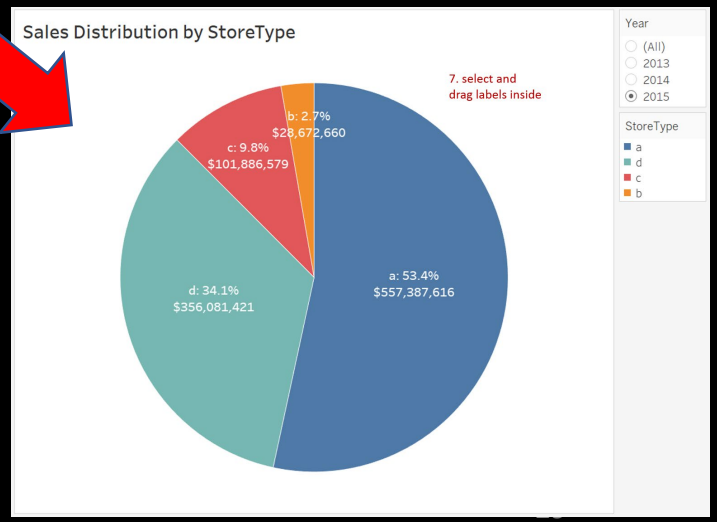
In order to answer the question, we need data visualization

Why Data Visualization?

- Explore data structure
- Detect outliers and unusual group
- Identifying trends and clusters
- Spotting patterns
- Evaluating modelling output
- Presenting results





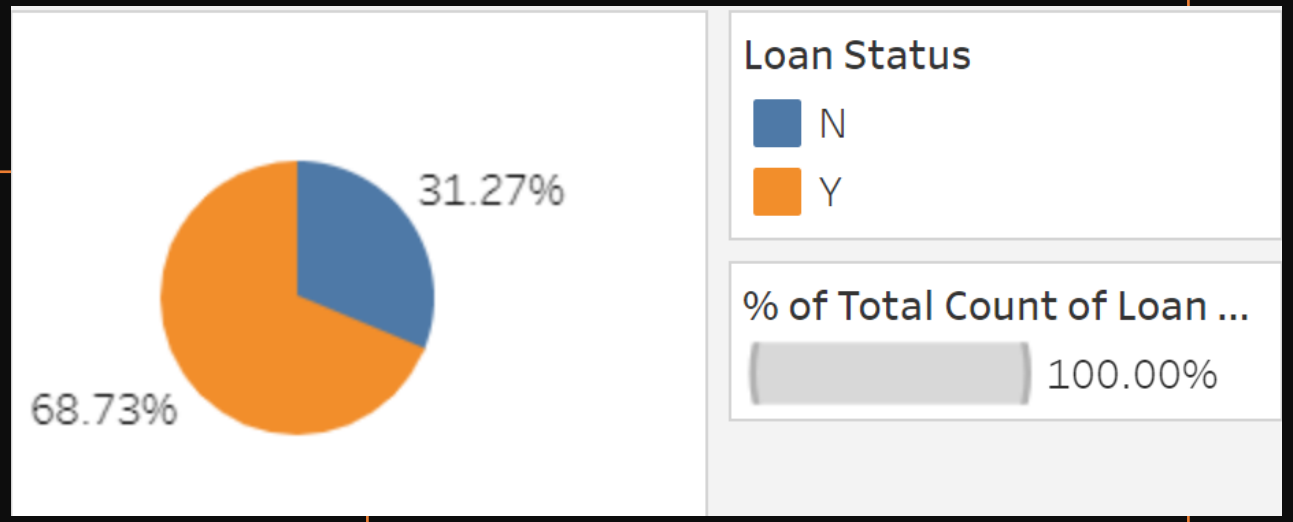


Data Exploration for Loan Status

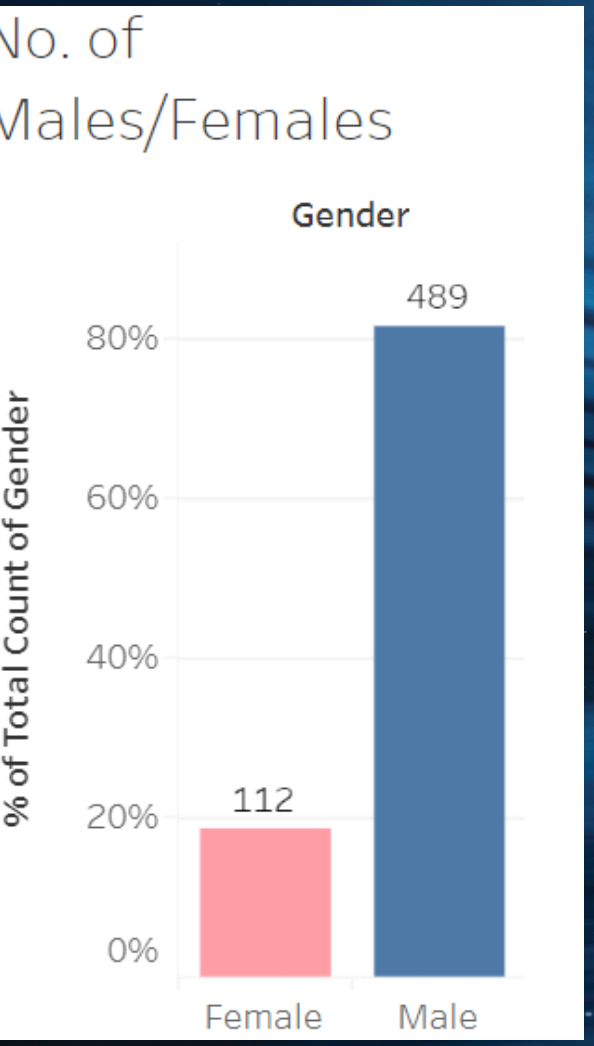
What is the successful loan rate?

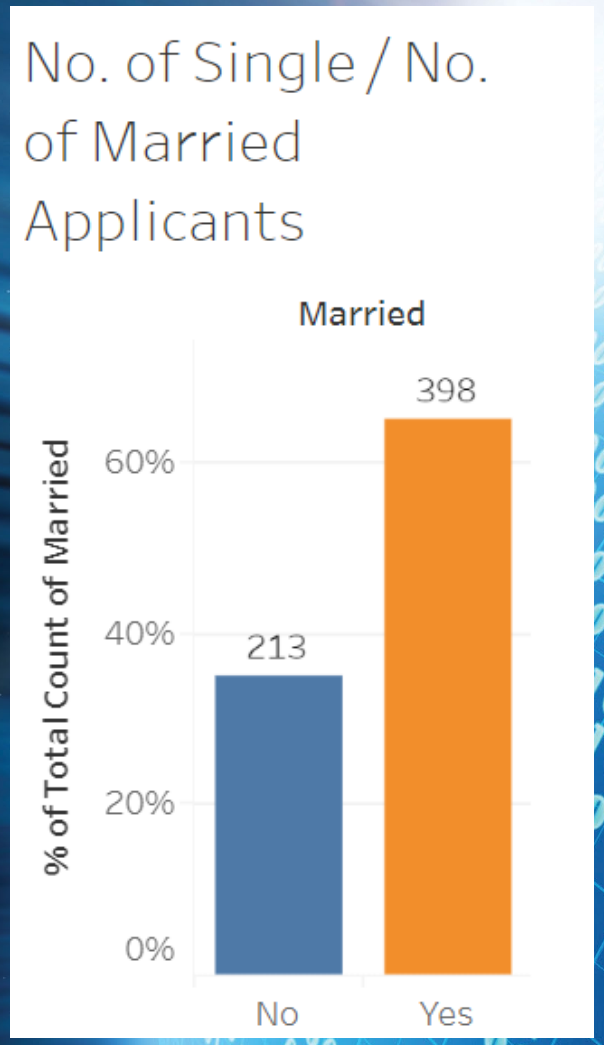
By using Tableau, we are able to see that out of the total of 614 applicants, 422(68.73%) are able to successfully take a loan

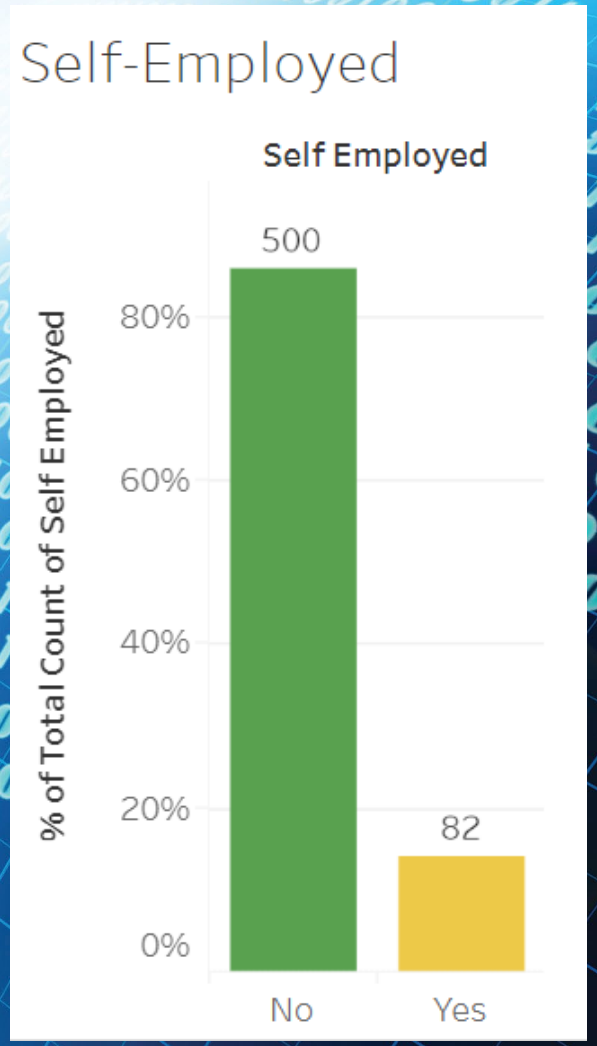
Loan Status	
N	192
Y	422
Grand Total	614

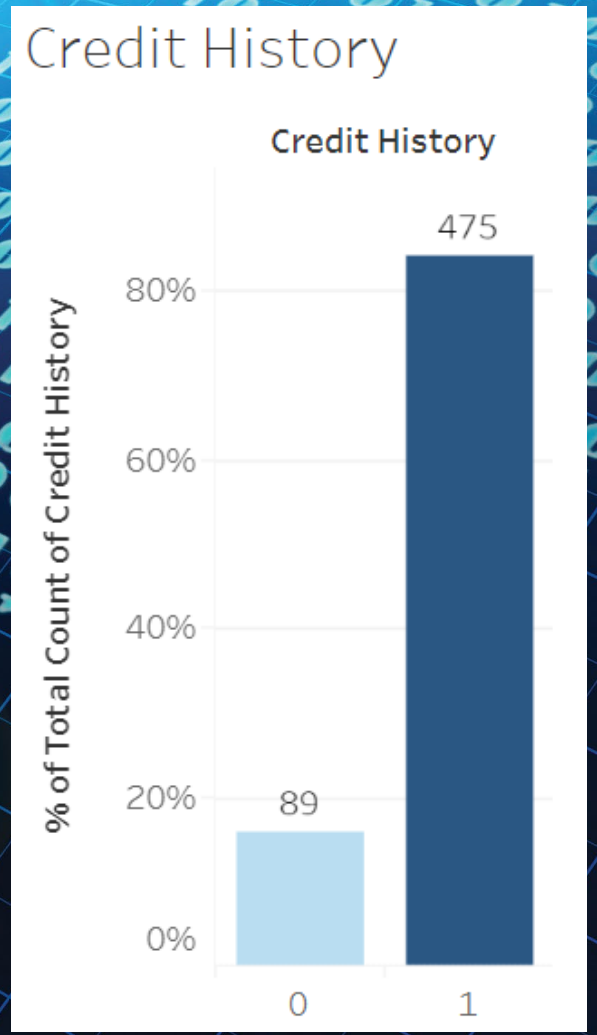


Nominal Variables(Categorical)



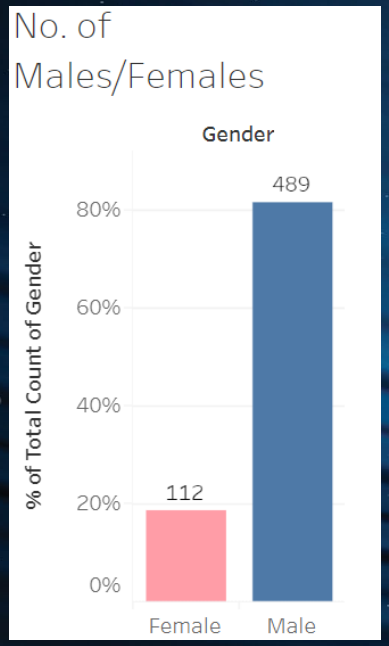


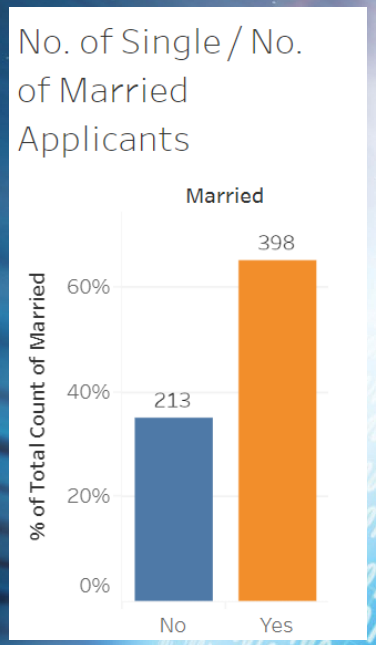


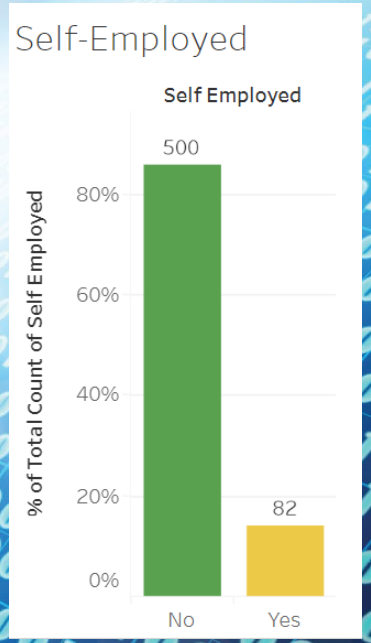


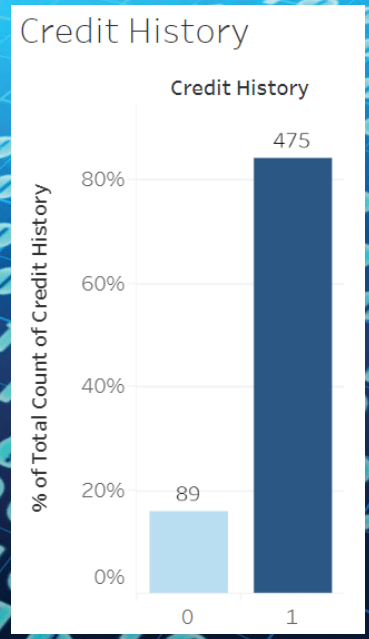
Univariate Analysis

Nominal Variables(Categorical)





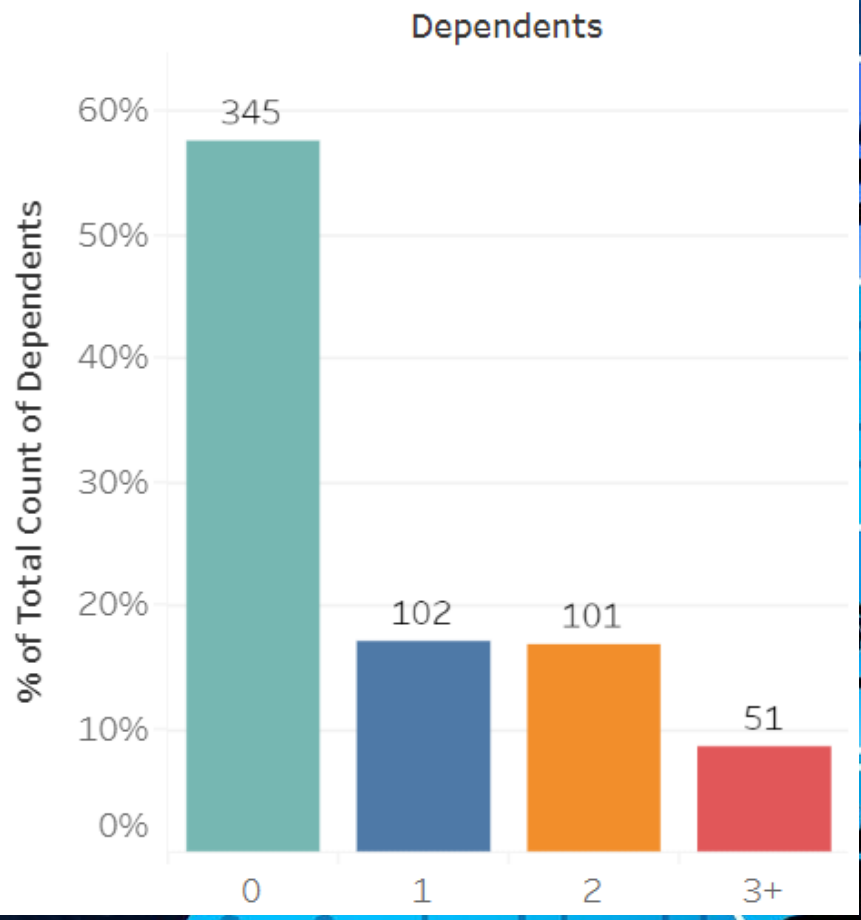




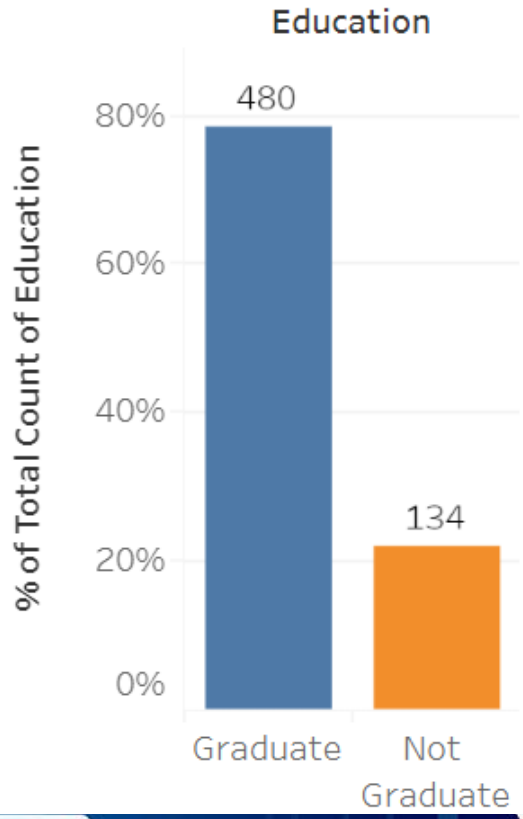
Summary of Nominal Variables

- **81% of applicants in the dataset are male**
- **65% of the applicants are married**
- **Only 15% of the applicants are self-employed**
- **84% of the applicants has repaid their previous debts**

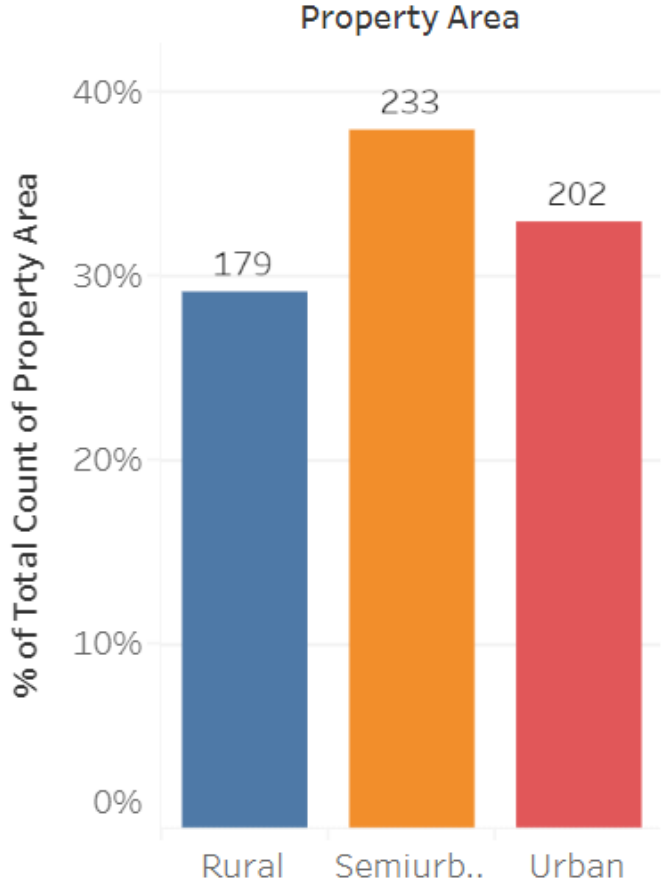
Number of Dependents



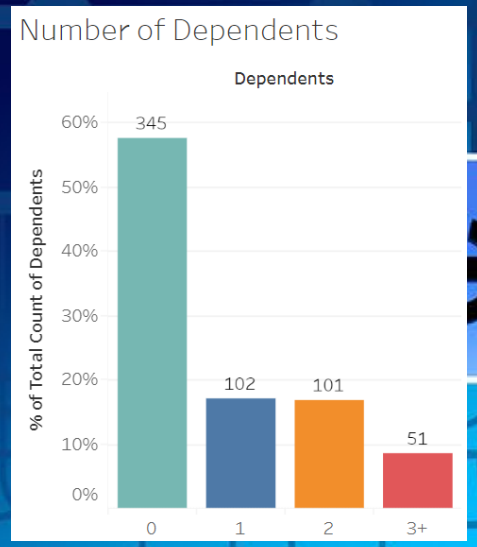
Number of Graduates

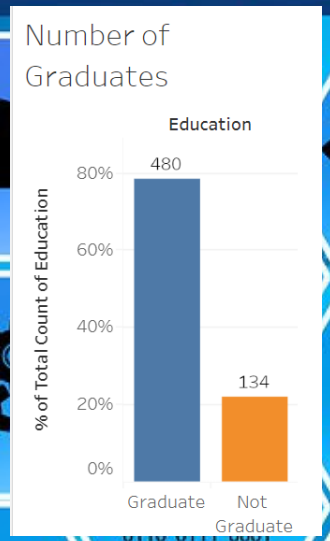


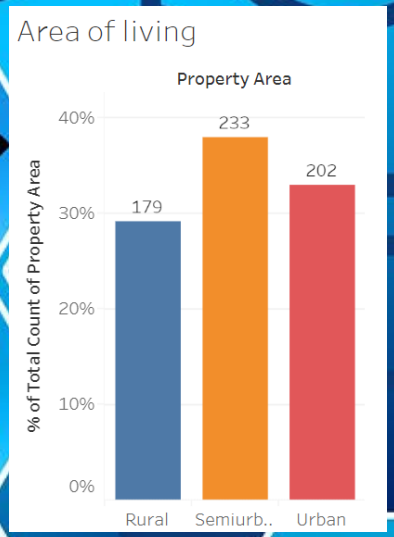
Area of living



Ordinal(Categorical)







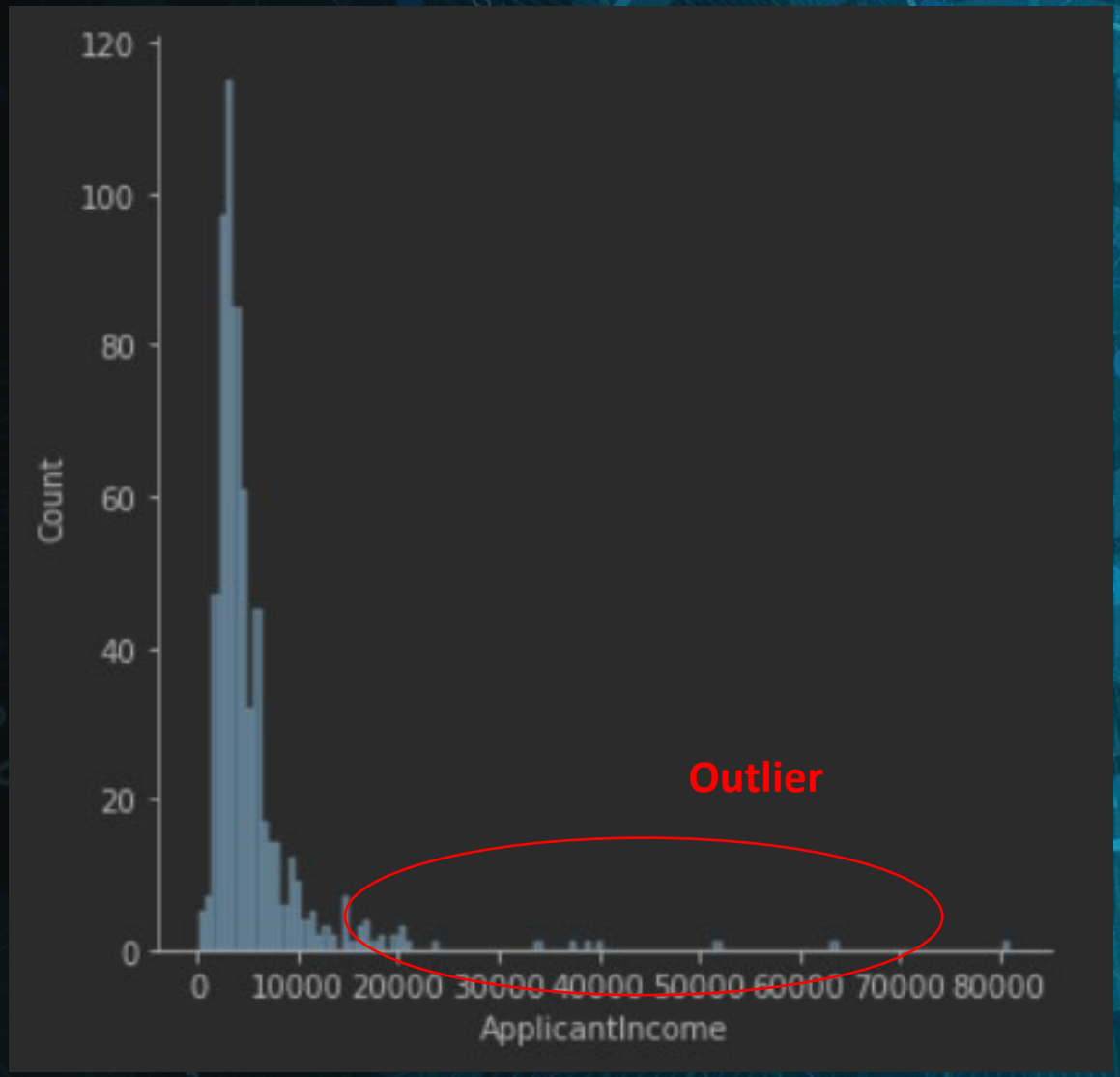
Summary of Ordinal Variables

- **Over 50% of the applicants does not have dependents**
- **Nearly 80% of applicants are graduates**
- **Semiurban area has slightly higher number of applicants**

Ordinal(Categorical)

ApplicantIncome(Numerical)

```
sns.displot(train['ApplicantIncome'])
plt.show()
```



Bulk of data is at the left of the graph and has a long right tail

An example of

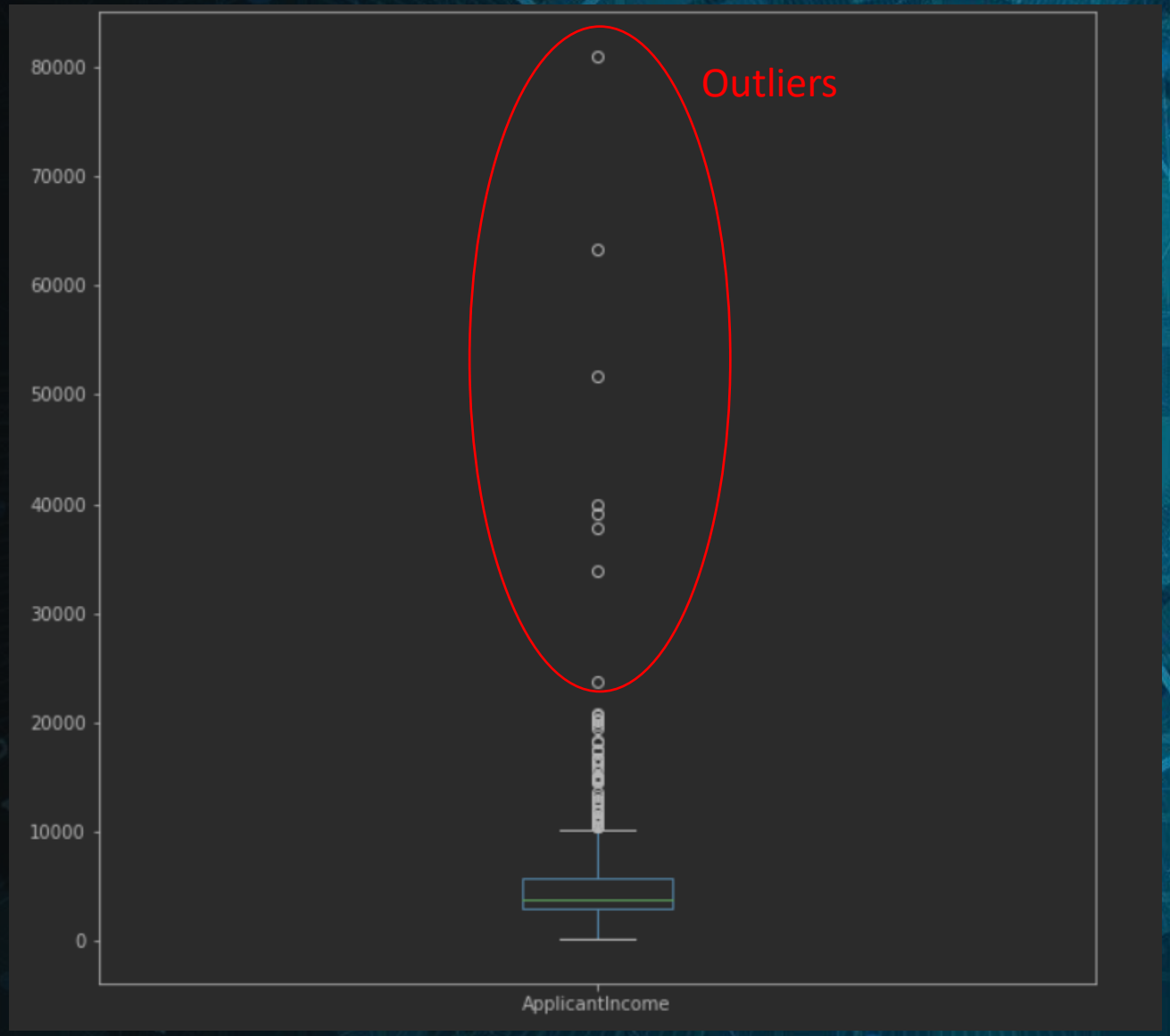
"Right Skewness"

A Density Plot is drawn using Seaborn Library which is good at determining the distribution shape and overall distribution

Univariate Analysis

ApplicantIncome(Numerical)

```
train['ApplicantIncome'].plot.box(figsize=(10,10))
plt.show()
```



Confirm the presence of a lot of outliers with extreme values

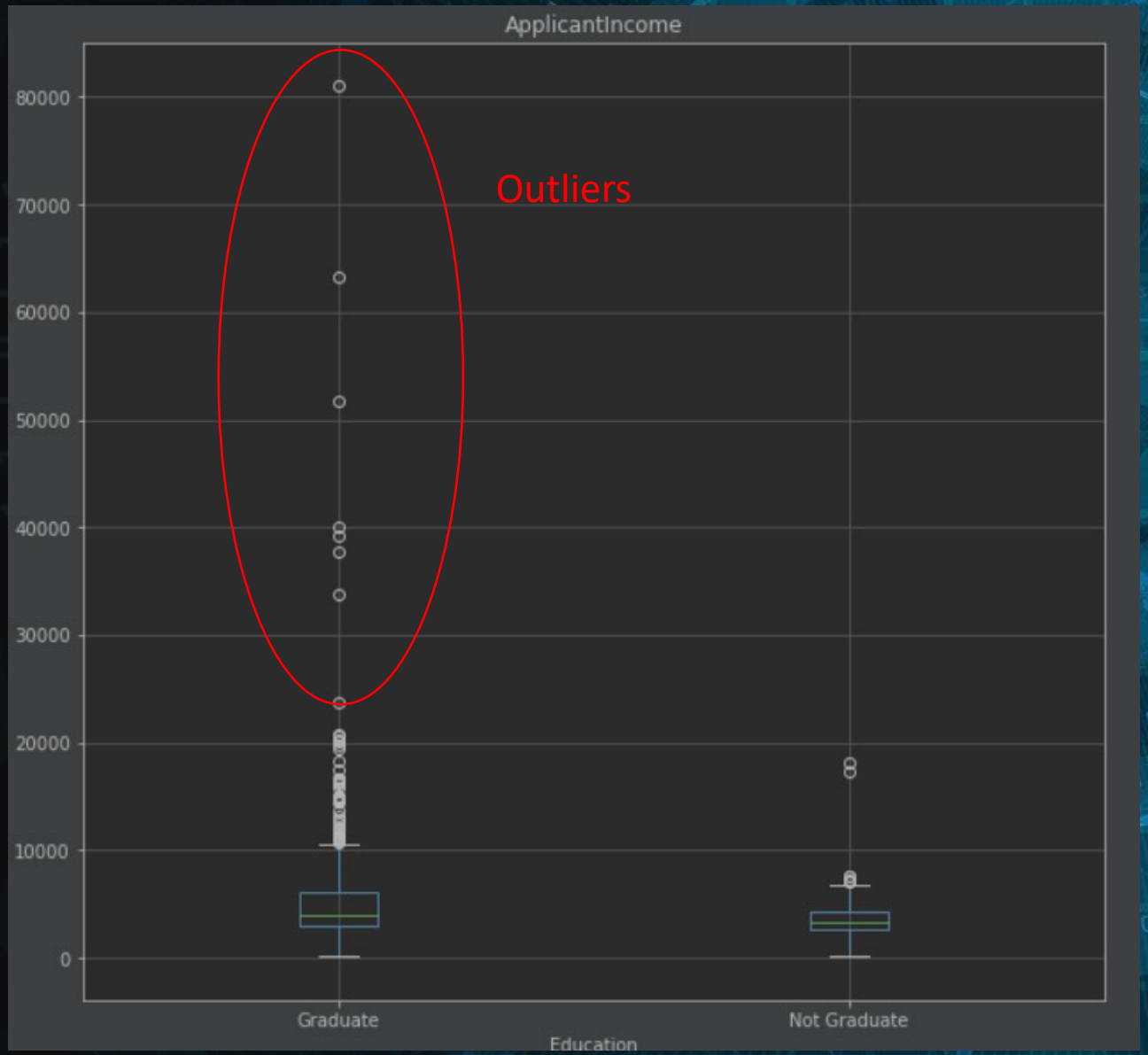
Drawn with box plot with Matplotlib which is useful for quickly summarizing results and identify outliers

Assumption Drawn

This can be attributed by the income disparity in the society

ApplicantIncome(Numerical)

```
train.boxplot(column='ApplicantIncome', by='Education', figsize=(10,10))
plt.suptitle("")
```

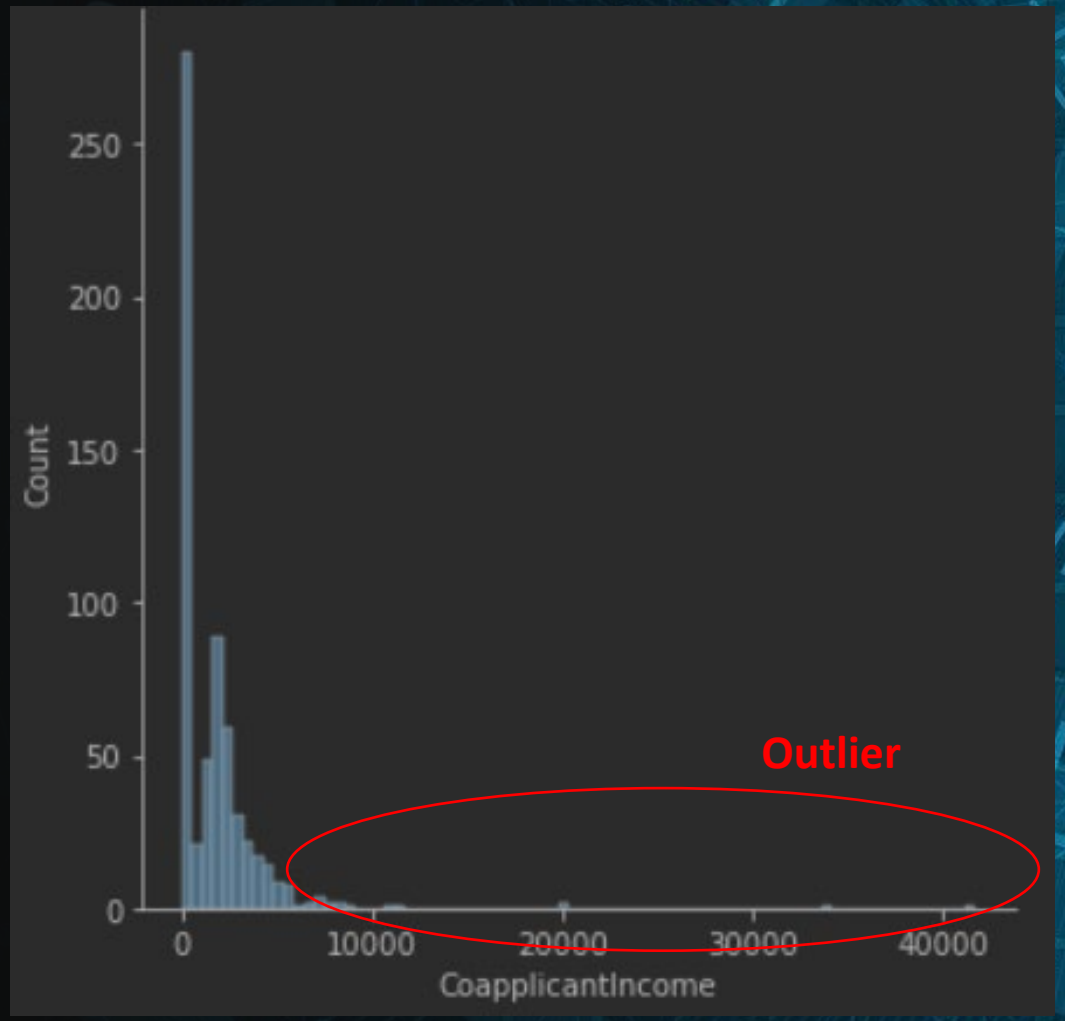


Splitting up the ApplicantIncome according to education level

We can see that there are a higher number of graduates with very high incomes and thus we can identify them as outlier

CoapplicantIncome(Numerical)

```
sns.displot(train['CoapplicantIncome'])
plt.show()
```



Majority of the co-applicants' income range from 0 to 5000

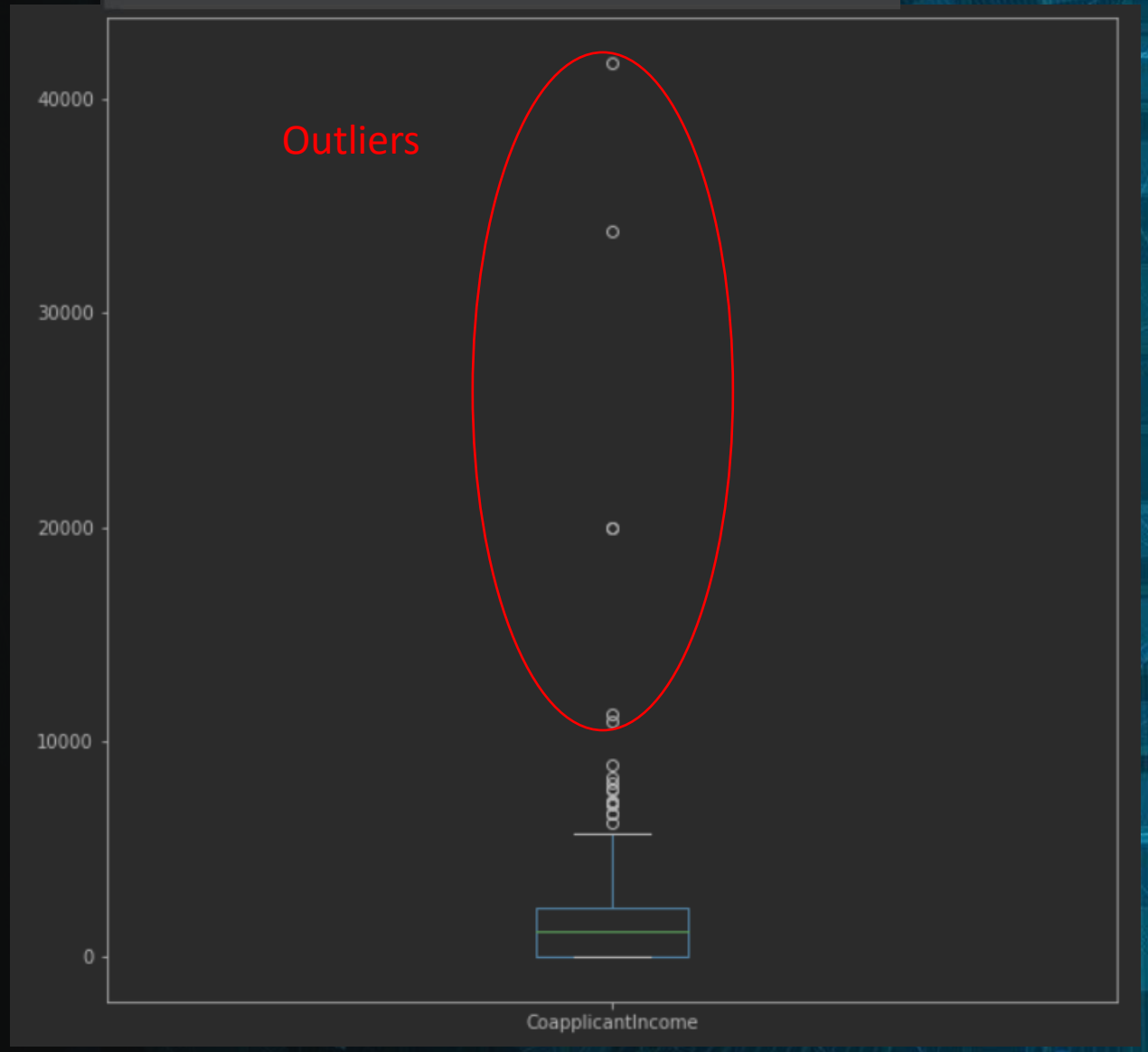
A similar uneven distribution as the applicantIncome (Right Skewness)

A **Density Plot** is drawn using **Seaborn Library** which is good at determining the distribution shape and overall distribution

Univariate Analysis

CoapplicantIncome(Numerical)

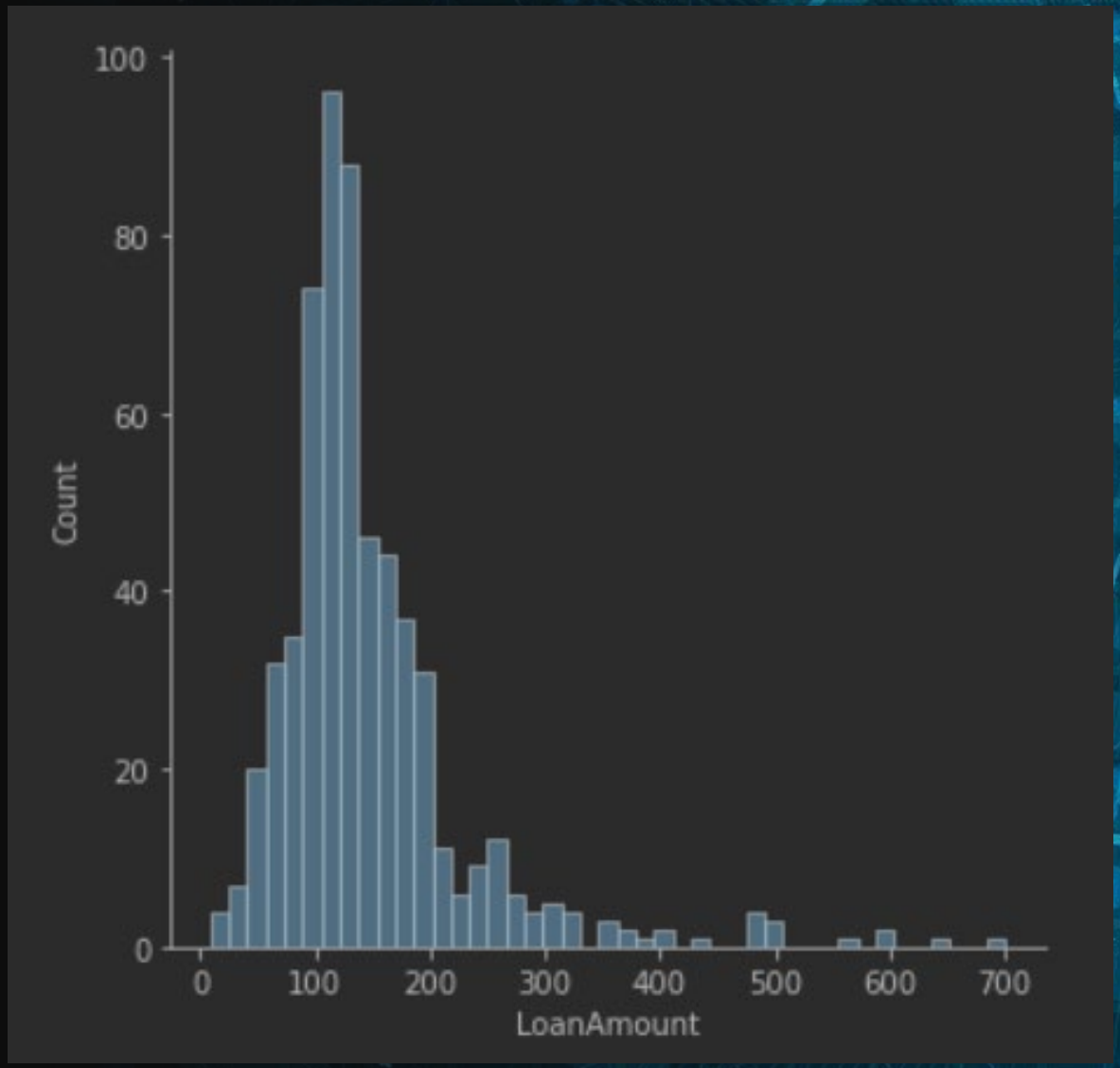
```
train['CoapplicantIncome'].plot.box(figsize=(10,10))
plt.show()
```



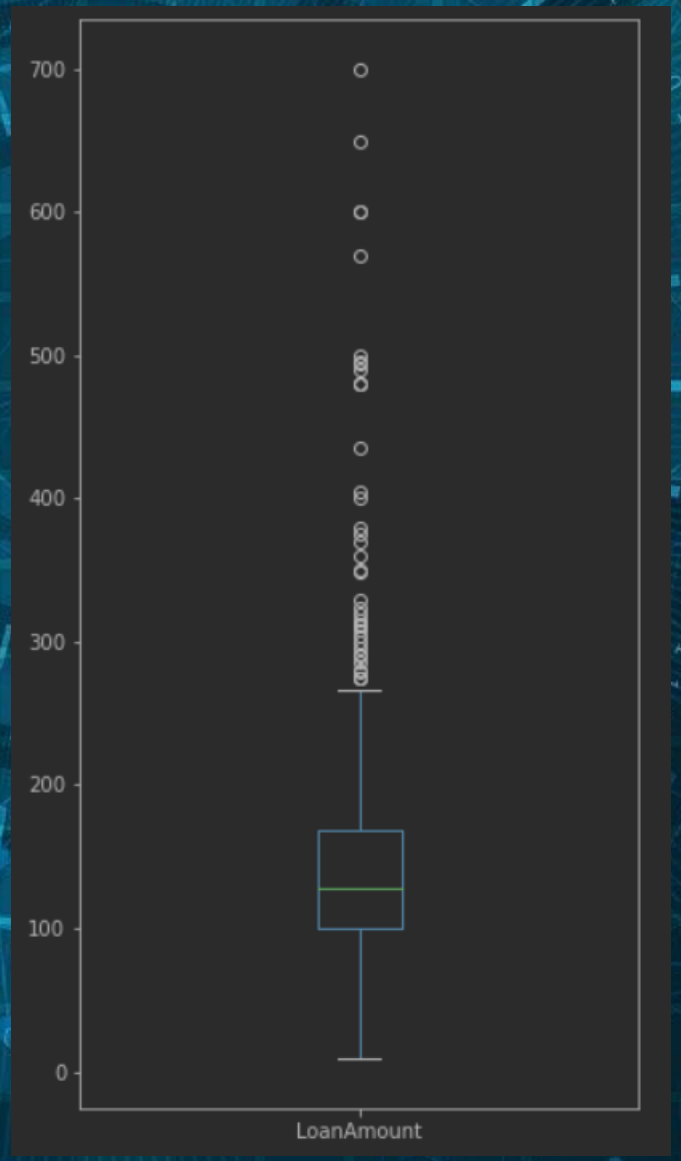
There is still presence of outliers, but not as many as ApplicantIncome

LoanAmount(Numerical)

```
sns.displot(train['LoanAmount'])
plt.show()
```




```
train['LoanAmount'].plot.box(figsize=(5,10))
plt.show()
```



The distribution is fairly normal comparing with the previous two

However, we can still see a lot of outliers

Data Cleansing (Null Value Imputation)

```
train.isnull().sum()
```

	data
Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

Length: 13, dtype: int64

Using the above method, we are able to find out the count of missing data for each variable

Imputation methods

- Replace with Summary Statistic such as mean, median, or mode
- Create a new variable that flags a missing column
- Replace NA with an outlier(Tree-Based models will implicitly understand that these outliers are associated with missing values

Data Cleansing (Null Value Imputation)

```
train.isnull().sum()
```

	data
Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

Length: 13, dtype: int64

First, we group them according to their data type

There are missing values in
Gender
Married
Dependents
Self_Employed
LoanAmount
Loan_Amount_Term
Credit_History

Categorical Variables

Gender
Married
Dependents
Credit_History
Self_Employed

Numerical Variables

Loan Amount
Loan_Amount_Term

Data Cleansing (Null Value Imputation)

	data
Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	0
Property_Area	0
Loan_Status	0

Length: 13, dtype: int64

Categorical Variables

Gender
Married
Dependents
Credit_History
Self_Employed

```
train['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
train['Married'].fillna(train['Married'].mode()[0], inplace=True)
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
train['Credit_History'].fillna(train['Credit_History'].mode()[0], inplace=True)
train.isnull().sum()
```

As Categorical Variables,
-We cannot use mean, and median makes no sense in this situation
-we will use mode (the most frequently appear) to fill in the null

Data Cleansing (Null Value Imputation)

Numerical Variables

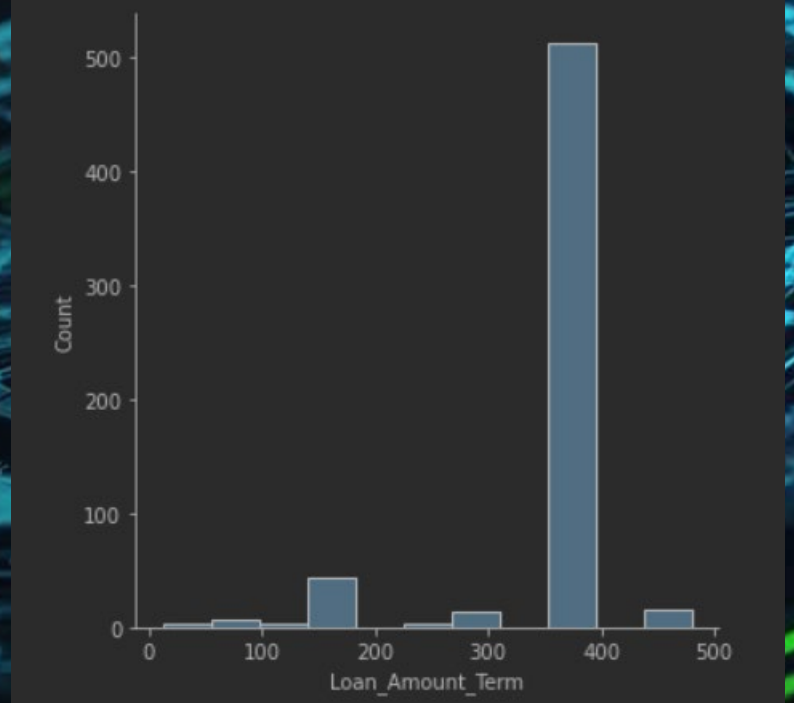
Loan _Amount_Term

```
train['Loan_Amount_Term'].value_counts()
```

	Loan_Amount_Term
360.0	512
180.0	44
480.0	15
300.0	13
84.0	4
240.0	4
120.0	3
36.0	2
60.0	2
12.0	1

```
sns.displot(train.Loan_Amount_Term)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d9599c4100>
```



We cannot use mean as the distribution is too uneven

```
train['Loan_Amount_Term'].mode()[0]
```

```
360.0
```

```
train['Loan_Amount_Term'].median()
```

```
360.0
```

We will use either mode or median to fill in the missing value since both of them returns the same result

Data Cleansing (Null Value Imputation)

Numerical Variables

Loan _Amount

As for Loan Amount, it is a numerical variable, so we will be using mean or median to impute the missing values

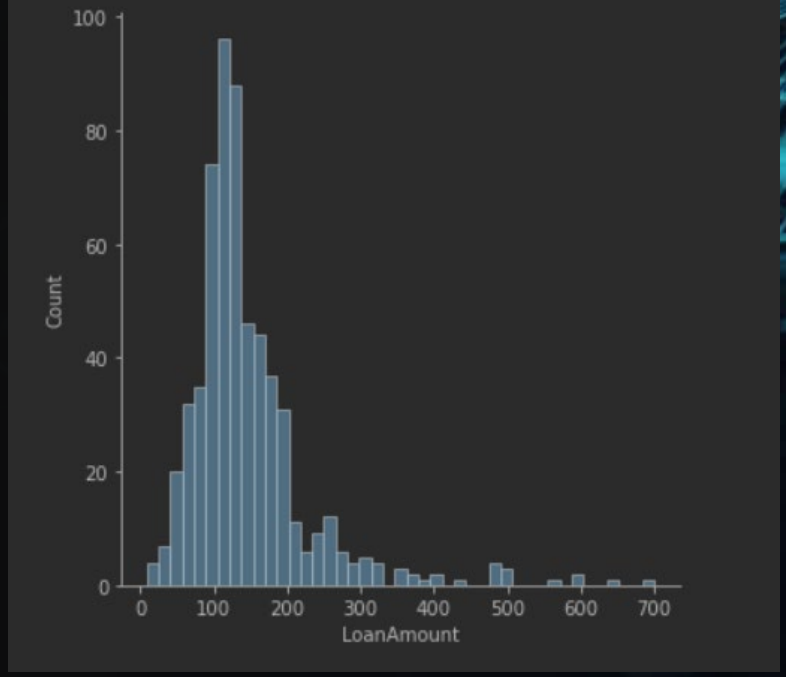
```
train['LoanAmount'].fillna(train['LoanAmount'].median(),inplace=True)
train.isnull().sum()
```

	data
Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0

Length: 13, dtype: int64

```
sns.displot(train.LoanAmount)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d95d4a3df0>
```



However, Loan_Amount has significant number of outliers

Mean approach will be highly affected by the presence of these outliers. So we will be using median

Now we confirm there is no null values in train data. We will do the same operation to test dataset

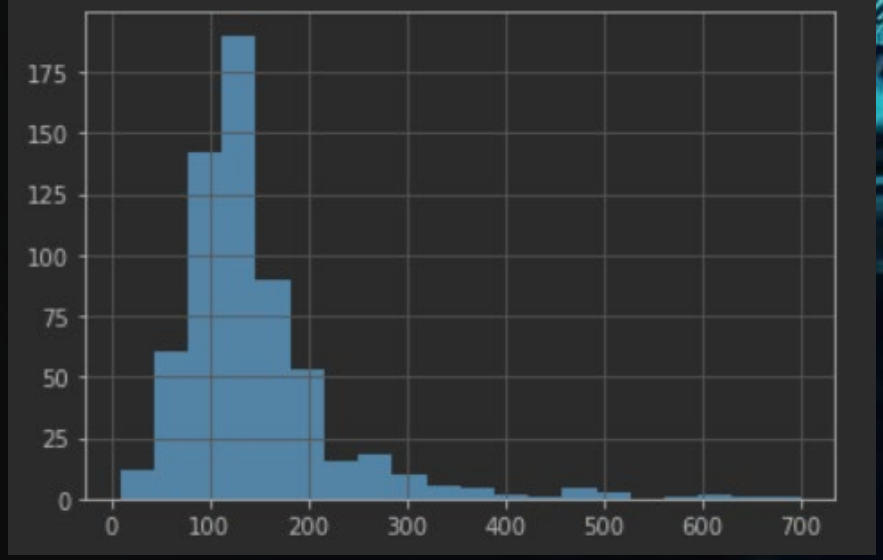
Data Cleansing (Outlier Treatment)

Numerical Variables

Loan _Amount

```
train['LoanAmount'].hist(bins=20)
```

```
<AxesSubplot:>
```



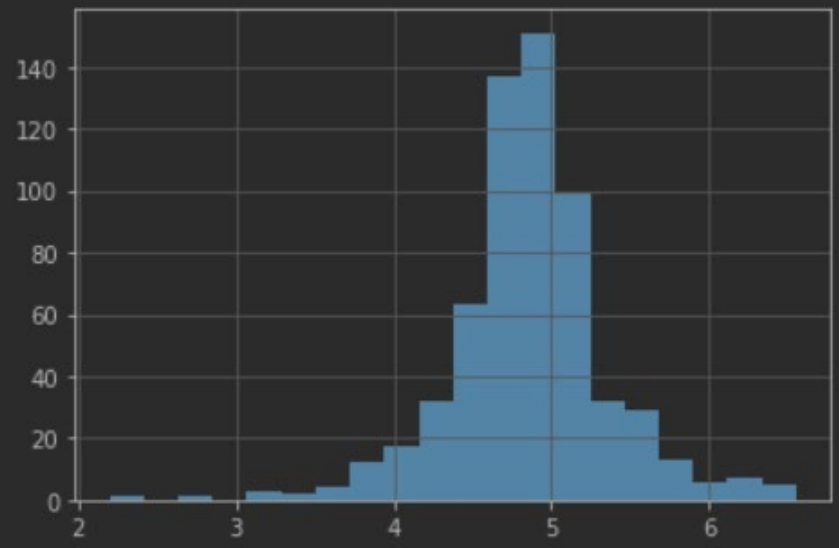
Characteristics of Right Skewness

-Bulk of the data is at left

-Right tail is longer

```
train['LoanAmount_log']=np.log(train['LoanAmount'])
train['LoanAmount_log'].hist(bins=20)
```

```
<AxesSubplot:>
```



Remove skewness by log transformation

Log transformation does not affect smaller values but reduces larger values.

Data Cleansing (Removing Unique Column)

Removing Loan_ID as it is unique and has no contribution to the prediction model

```
train['Loan_ID'].describe()
```

	Loan_ID
count	614
unique	614
top	LP002478
freq	1

Length: 4, dtype: object Open in new tab

```
test['Loan_ID'].describe()
```

	Loan_ID
count	367
unique	367
top	LP002781
freq	1

Length: 4, dtype: object Open in new tab

```
train=train.drop('Loan_ID',axis=1)
test=test.drop('Loan_ID',axis=1)
```

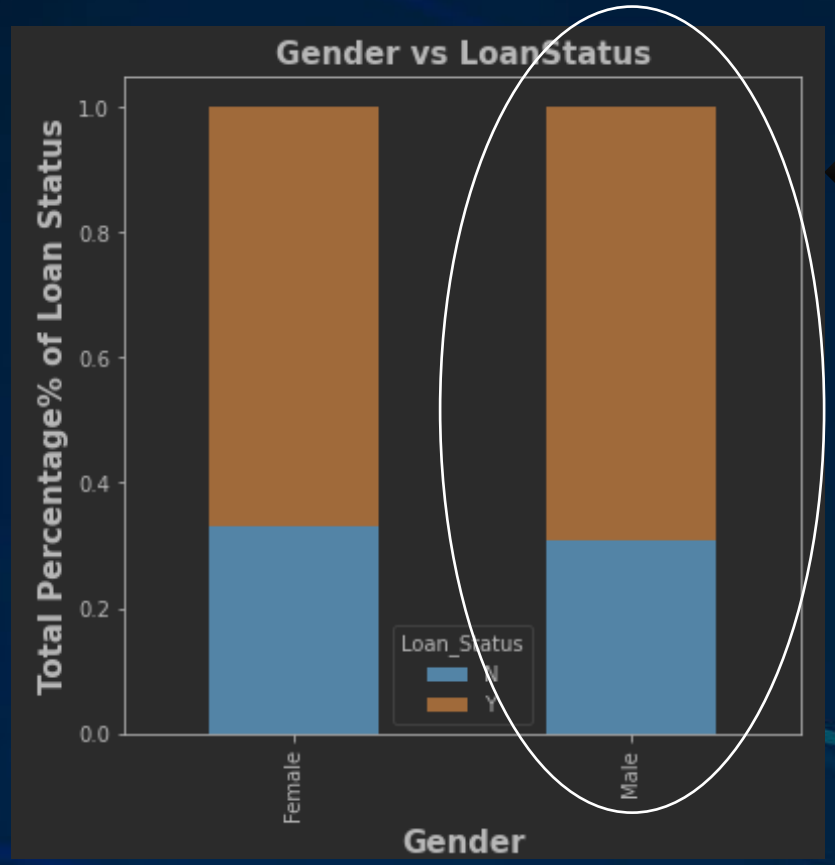
Diagnostic Analytics

Drill Down, Data Discovery, Correlation, Data mining

Hypothesis

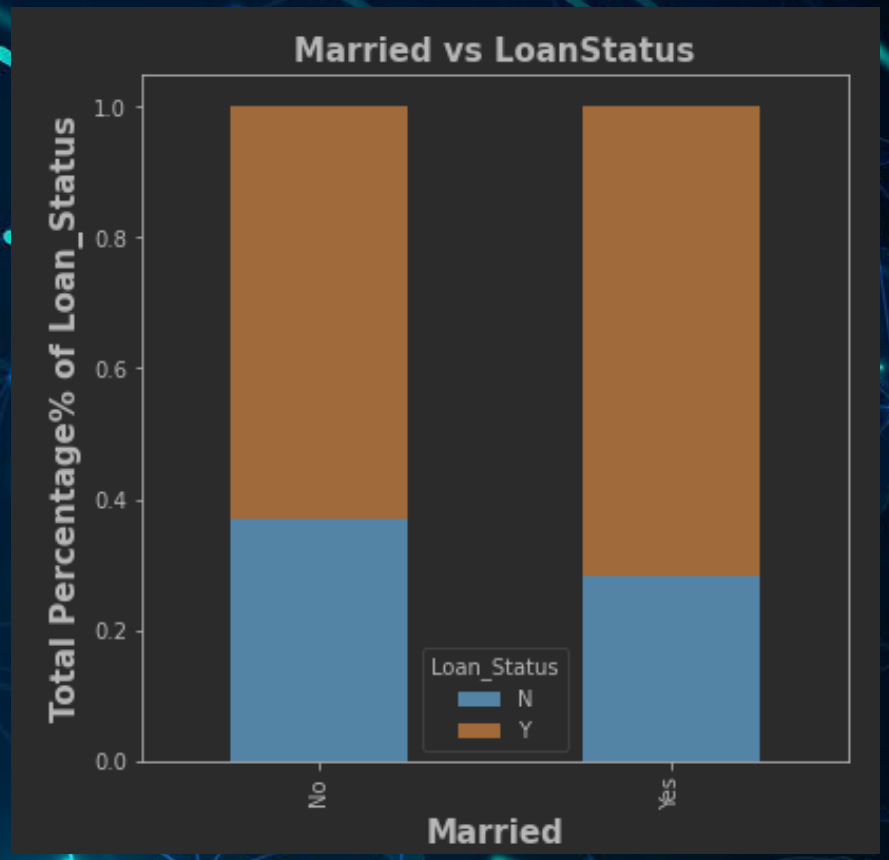
1. Applicants that have repaid their previous debt which shows their credibility is more likely to get their loan approved
2. If the applicant income is low, the chance of getting loan approval is low
3. If the requested loan amount is high, the chances of getting loan approval is low

Bivariate Analysis
(Independent Categorical vs Dependent Variable)



Why use stacked bar chart?

- Data is easy to view when comparing data points
- Interpret data quickly which save time

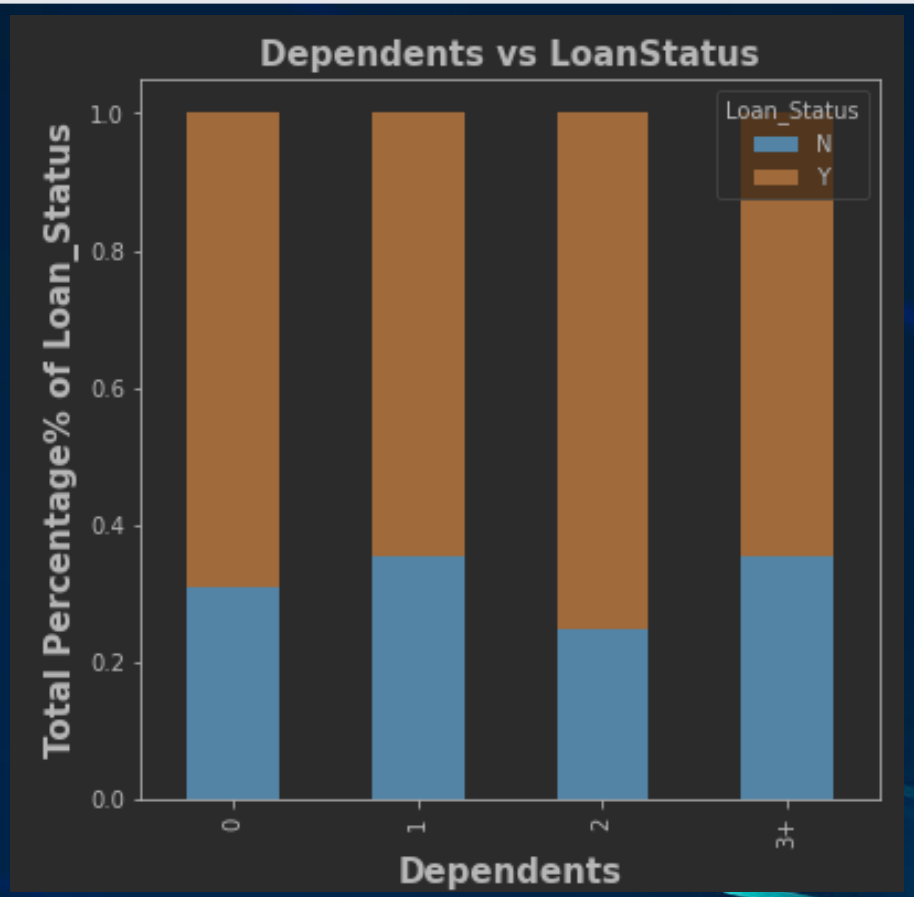


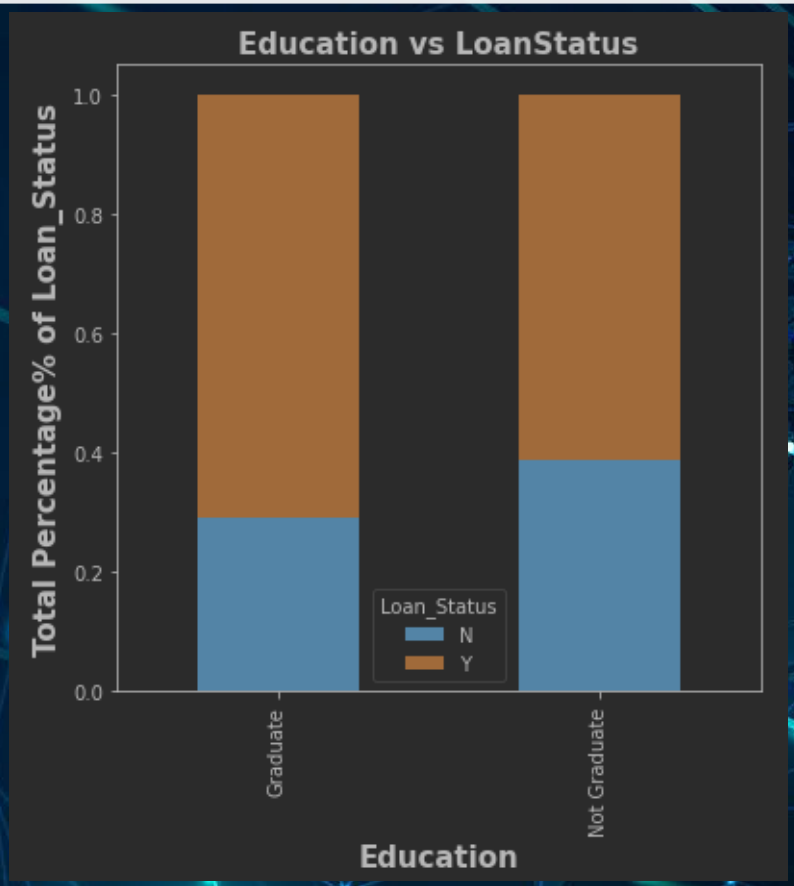
Not a substantial difference between male and female loan approval rates

Married Applicants have a slightly higher chances of loan approval

rawpixel

Bivariate Analysis
(Independent Categorical vs Dependent Variable)



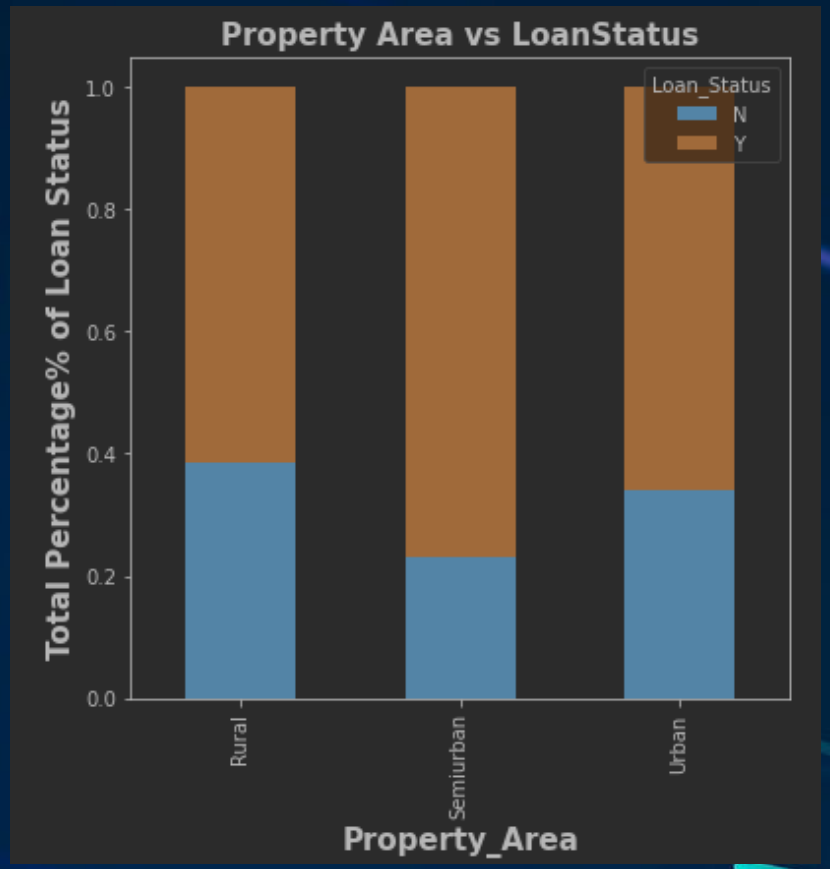


Distribution of applicant of 0 dependent is almost similar with applicant with 3+ dependent

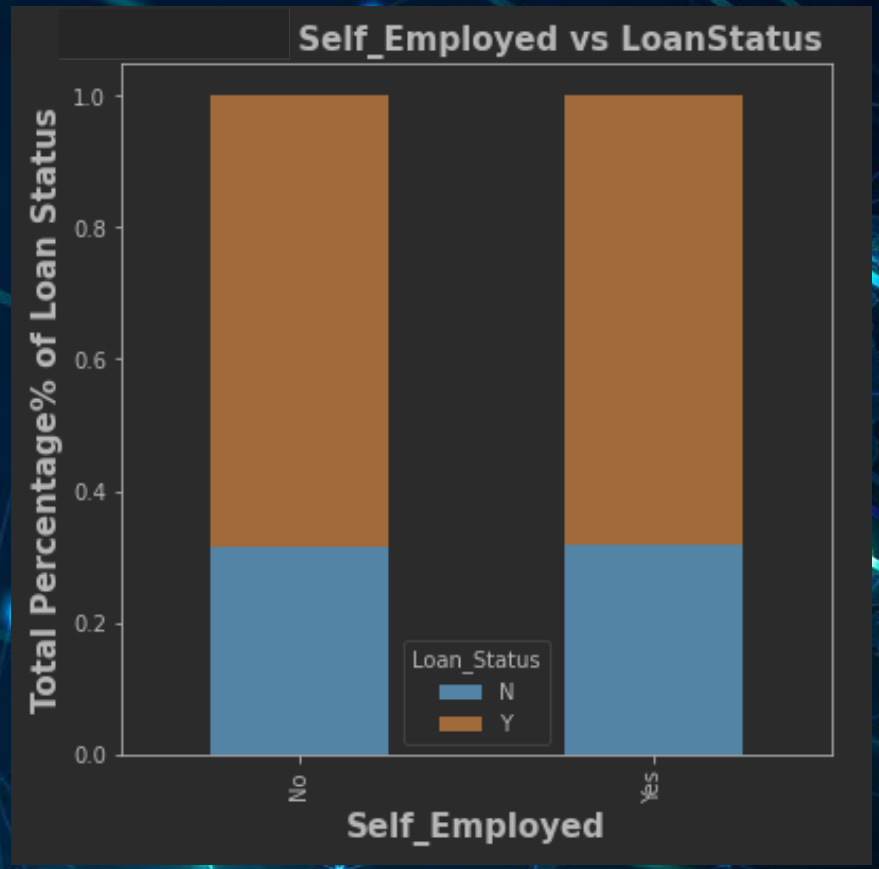
Therefore we can deduce that the number of dependents does not affect the loan status

Graduate have slightly higher chance of loan approval comparing with non-graduates

Bivariate Analysis
(Independent Categorical vs Dependent Variable)



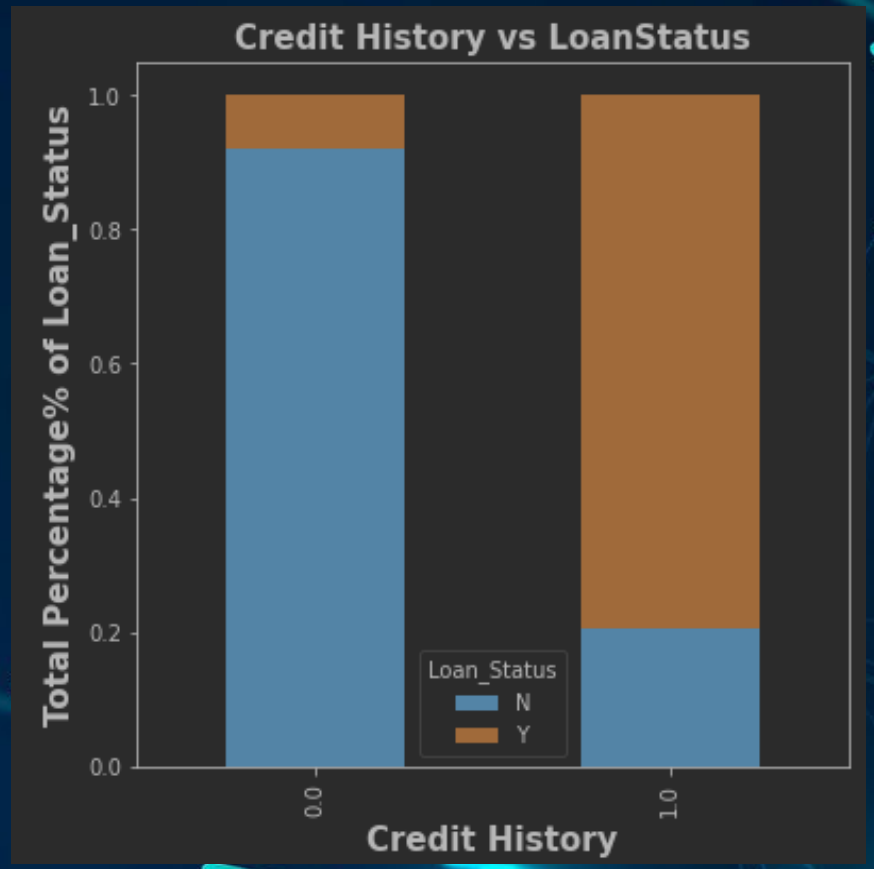
The proportion of loans approved for Semi-Urban area is higher than those living in Rural or Urban Area



There is nothing significant we can infer from the self-employed values

rawpixel

Bivariate Analysis
(Independent Categorical vs Dependent Variable)



Applicants with credit History of 1 is more likely to get their loan approved

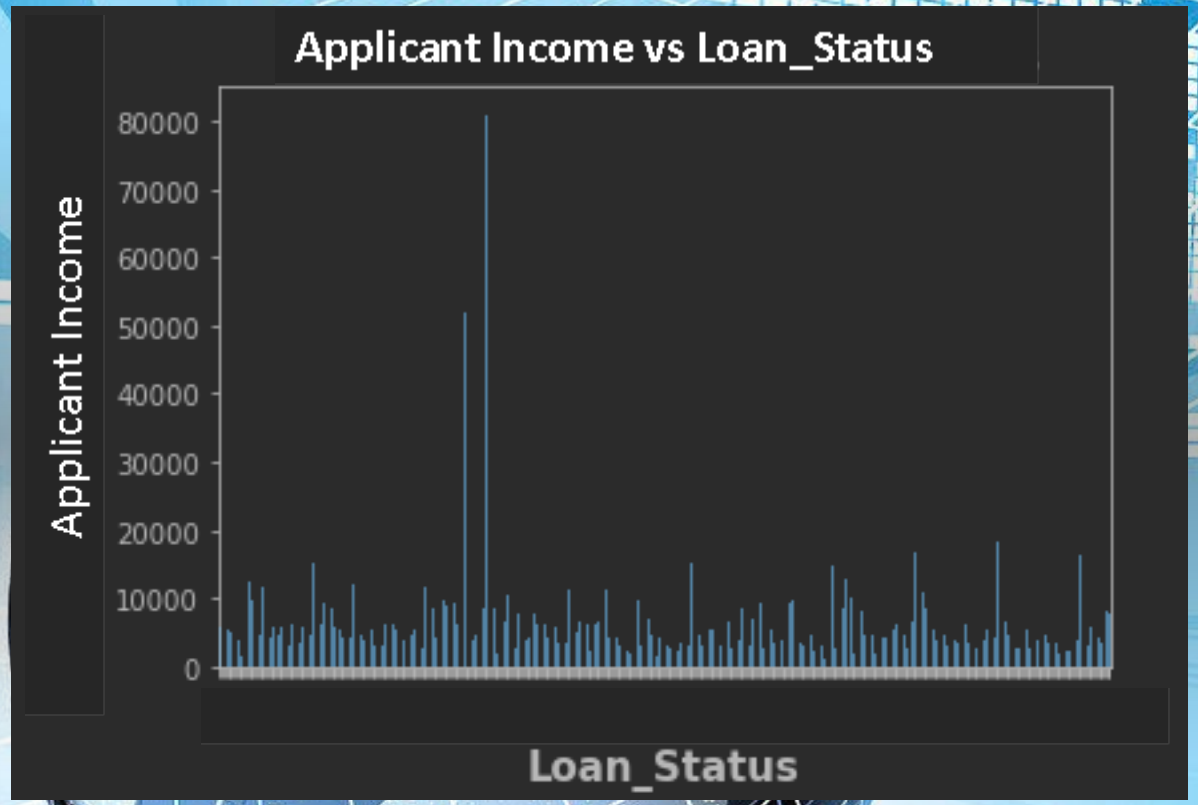
Hypothesis 1:

Applicants that have repaid their previous debt which shows their credibility is more likely to get their loan approved

rawpixel

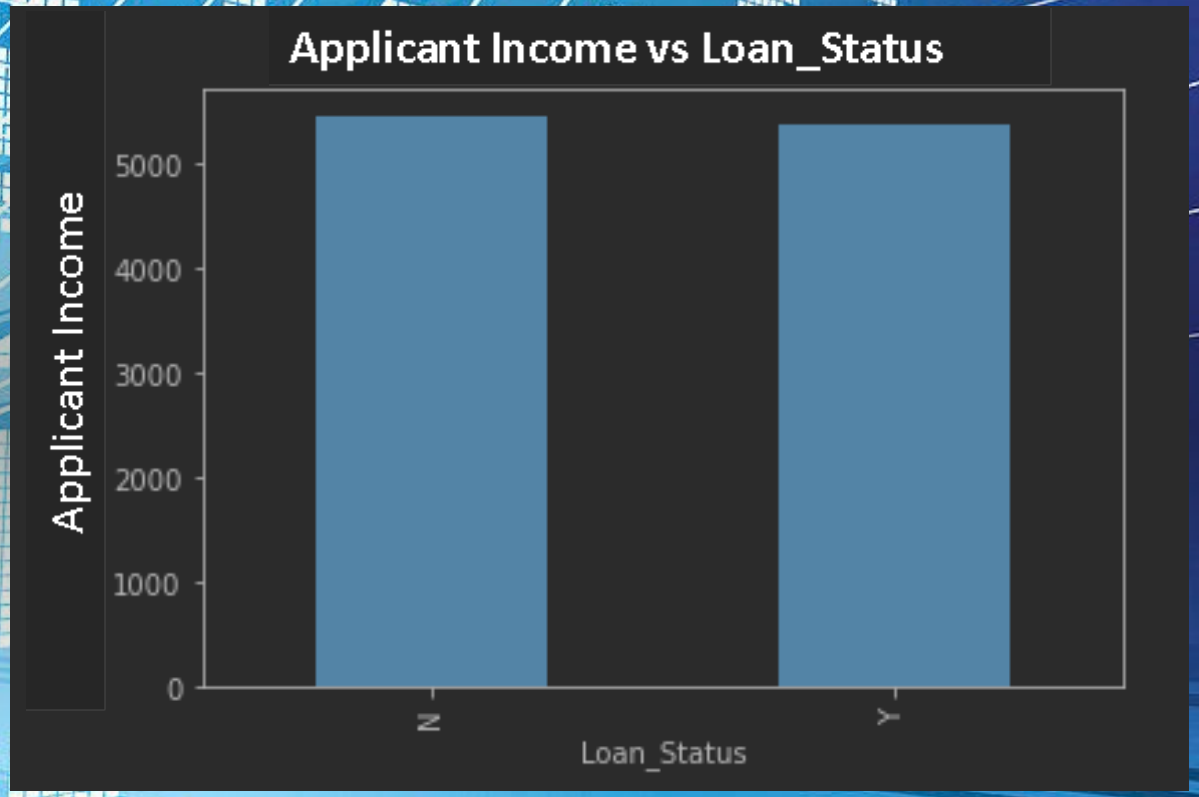
Numerical Independent Variable vs Target Variable (ApplicationIncome vs LoanStatus)

```
train.groupby('Loan_Status')['ApplicantIncome'].plot.bar()
plt.title('ApplicationIncome vs LoanStatus', fontweight='bold', size=15)
plt.ylabel('ApplicationIncome', fontweight='bold', size=15)
plt.xlabel('Loan_Status', fontweight='bold', size=15)
plt.show()
```



Problem 1:
When we do a bar plot of Applicant Income vs Loan_Status with a groupby method, we cannot infer anything from it at all

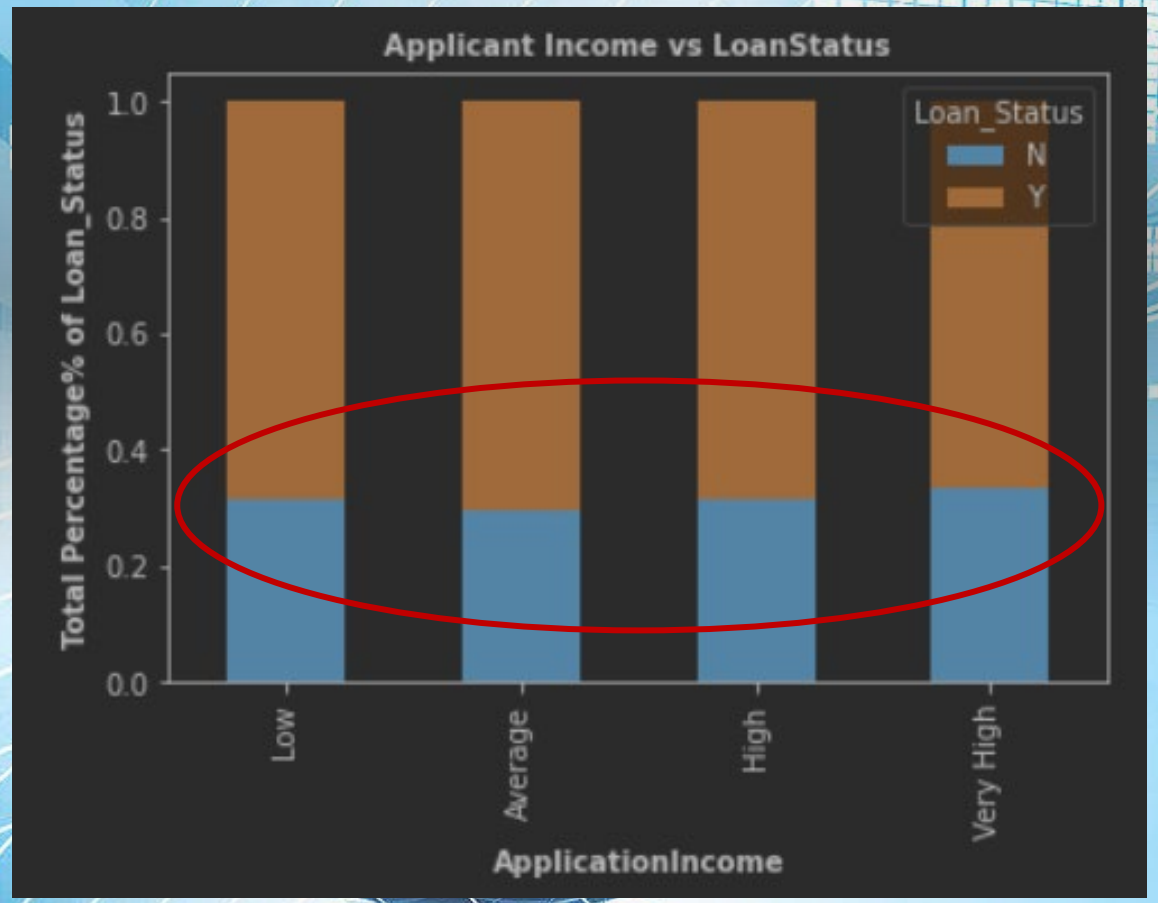
```
train.groupby('Loan_Status')['ApplicantIncome'].mean().plot.bar()
plt.title('ApplicationIncome vs LoanStatus', fontweight='bold', size=15)
plt.ylabel('ApplicationIncome', fontweight='bold', size=15)
plt.xlabel('Loan_Status', fontweight='bold', size=15)
plt.show()
```



Problem 2:
We tried to average out the Applicant Income, but we still don't see any changes in the mean income

Numerical Independent Variable vs Target Variable (Applicant Income vs Loan_Status)

```
bins=[0,2500,4000,6000,81000]
group=['Low','Average','High','Very High']
train['Income_bin']=pd.cut(train['ApplicantIncome'], bins, labels=group)
Income_bin=pd.crosstab(train['Income_bin'],train['Loan_Status'])
Income_bin.div(Income_bin.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True)
plt.title('Applicant Income vs LoanStatus', fontweight='bold', size=10)
plt.ylabel('Total Percentage% of Loan_Status', fontweight='bold', size=10)
plt.xlabel('ApplicationIncome', fontweight='bold', size=10)
plt.show()
```




```
train.describe()
```

	ApplicantIncome
count	614.000000
mean	5403.459283
std	6109.041673
min	150.000000
25%	2877.500000
50%	3812.500000
75%	5795.000000
max	81000.000000

Solution:
We make bins for the Applicant Income based on the percentiles

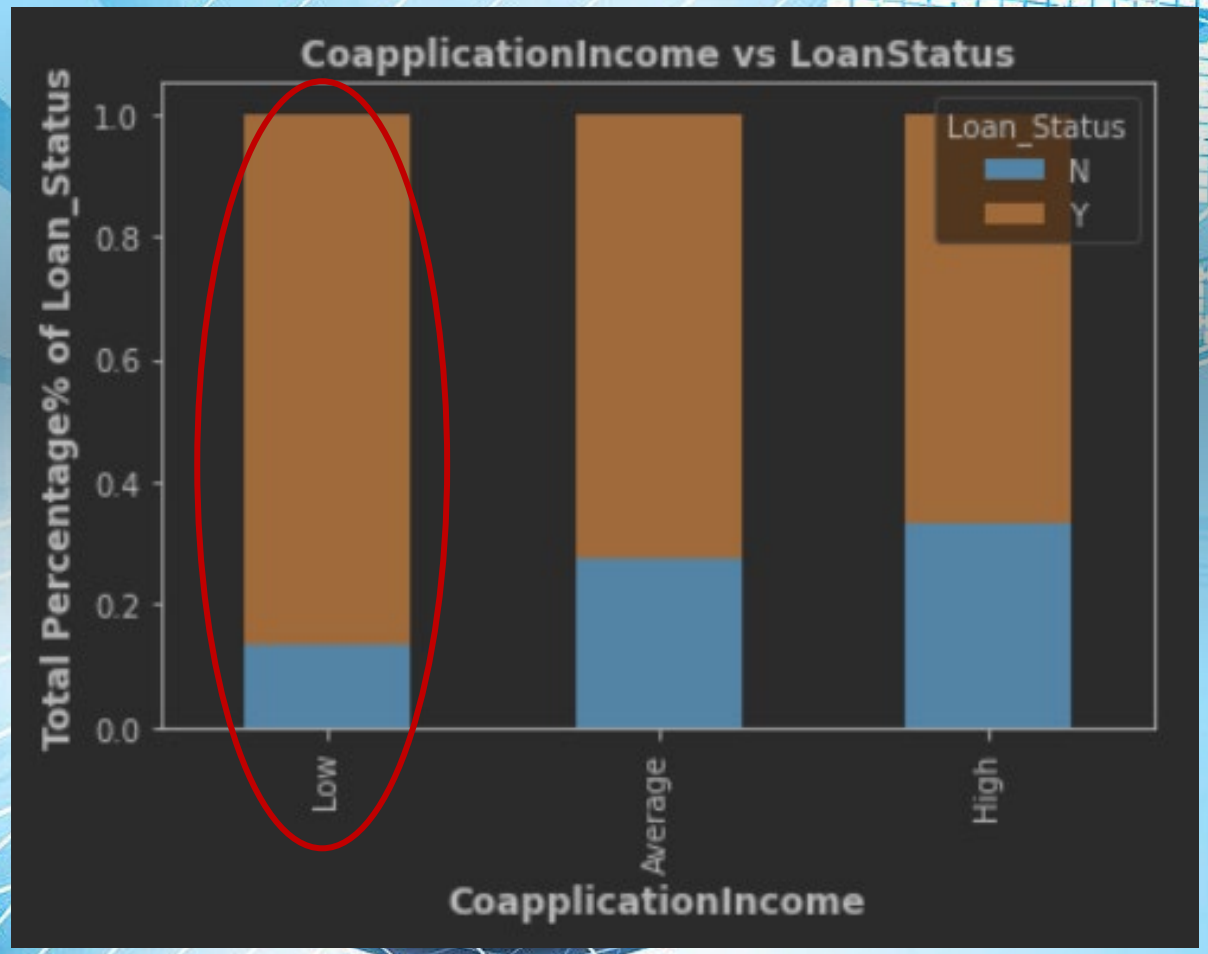
However, we see that Applicant Income does not affect the chances of loan approval which contradicts our hypothesis 2 .

Hypothesis 2:

If the applicant income is low, the chances of getting loan approval is low

Numerical Independent Variable vs Target Variable (Co applicant Income vs Loan_Status)

```
bins=[0,1000,3000,42000]
group=['Low','Average','High']
train['Coapplicant_Income_bin']=pd.cut(train['CoapplicantIncome'],bins,labels=group)
Coapplicant_Income_bin=pd.crosstab(train['Coapplicant_Income_bin'],train['Loan_Status'])
Coapplicant_Income_bin.div(Coapplicant_Income_bin.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True)
plt.title('CoapplicationIncome vs LoanStatus', fontweight='bold', size=12)
plt.ylabel('Total Percentage% of Loan_Status', fontweight='bold', size=12)
plt.xlabel('CoapplicationIncome', fontweight='bold', size=12)
plt.show()
```




```
train['CoapplicantIncome'].describe()
```

	CoapplicantIncome
count	614.000000
mean	1621.245798
std	2926.248369
min	0.000000
25%	0.000000
50%	1188.500000
75%	2297.250000
max	41667.000000

Length: 8, dtype: float64 Open in new tab

Using the same method, we make bins and group the Coapplicant Income to 'Low', 'Average', 'High'

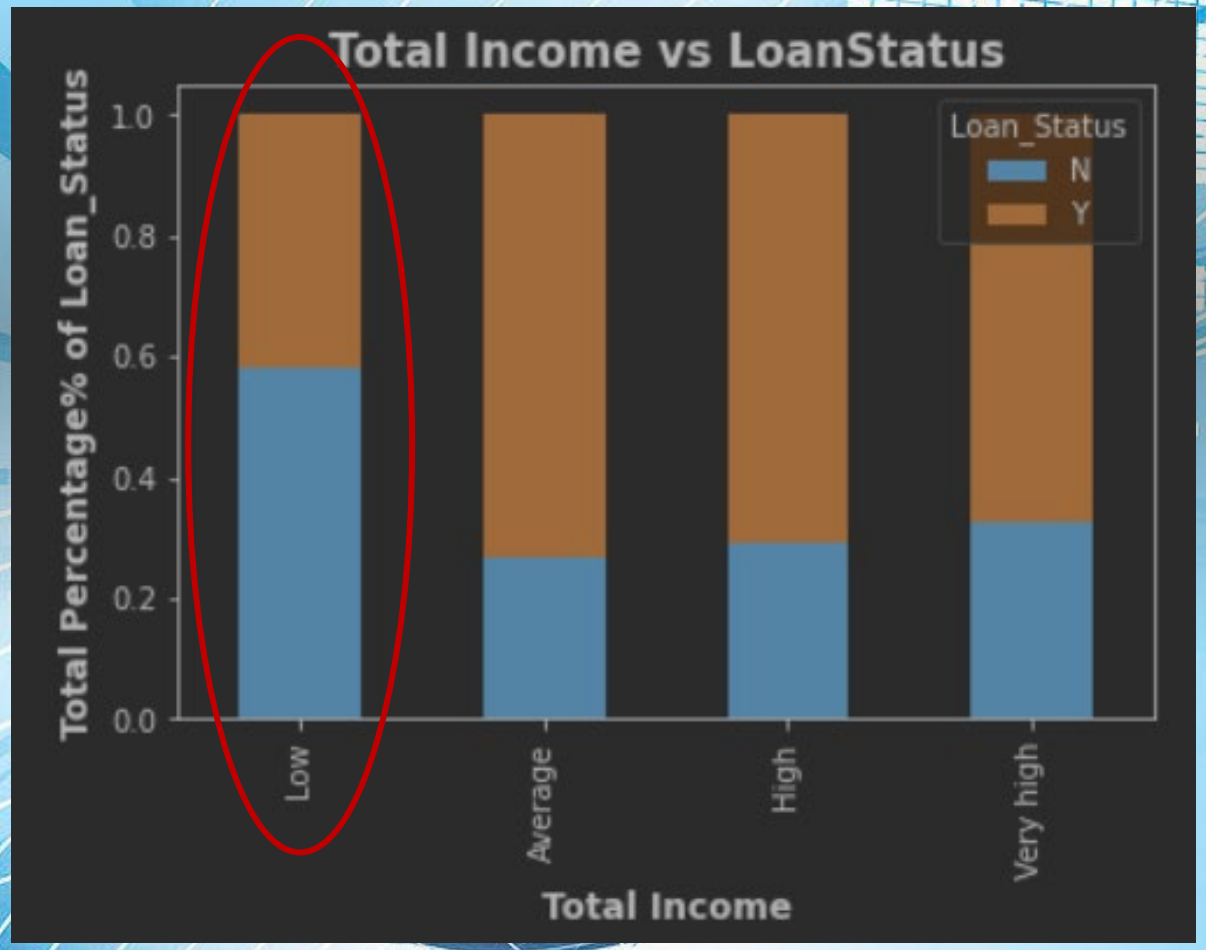
Looking at the chart, it can be inferred as if the Co applicant's income is low, the chance of loan approval is high which is wrong.

The possible reason behind this is that some of applicants do not have co applicant hence the value in the CoapplicantIncome is display as '0' value. Therefore, the Loan_Status is not directly dependent on Coapplicant Income.

CoapplicantIncome
2569.0
0.0
1929.0
0.0
0.0
0.0
7750.0

Numerical Independent Variable vs Target Variable (ApplicantIncome + CoapplicationIncome vs LoanStatus)

```
train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']
bins=[0,2500,4000,6000,81000]
group=['Low','Average','High','Very high']
train['Total_Income_bin']=pd.cut(train['Total_Income'],bins,labels=group)
Total_Income_bin=pd.crosstab(train['Total_Income_bin'],train['Loan_Status'])
Total_Income_bin.div(Total_Income_bin.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True)
plt.title('Total Income vs LoanStatus', fontweight='bold', size=15)
plt.ylabel('Total Percentage% of Loan_Status', fontweight='bold', size=12)
plt.xlabel('Total Income', fontweight='bold', size=12)
plt.show()
```



To solve the issue of Applicant Income and Coapplicant Income, we will sum up the two variables into a new column called Total Income.

We will use the Applicant Income's min to max as the categories for our bins

```
train.describe()
```

	ApplicantIncome
count	614.000000
mean	5403.459283
std	6109.041673
min	150.000000
25%	2877.500000
50%	3812.500000
75%	5795.000000
max	81000.000000

From this chart, we can see that applicants with low total income has lesser chance of approval.
So we rephrase our hypothesis 2 to fit the situation.

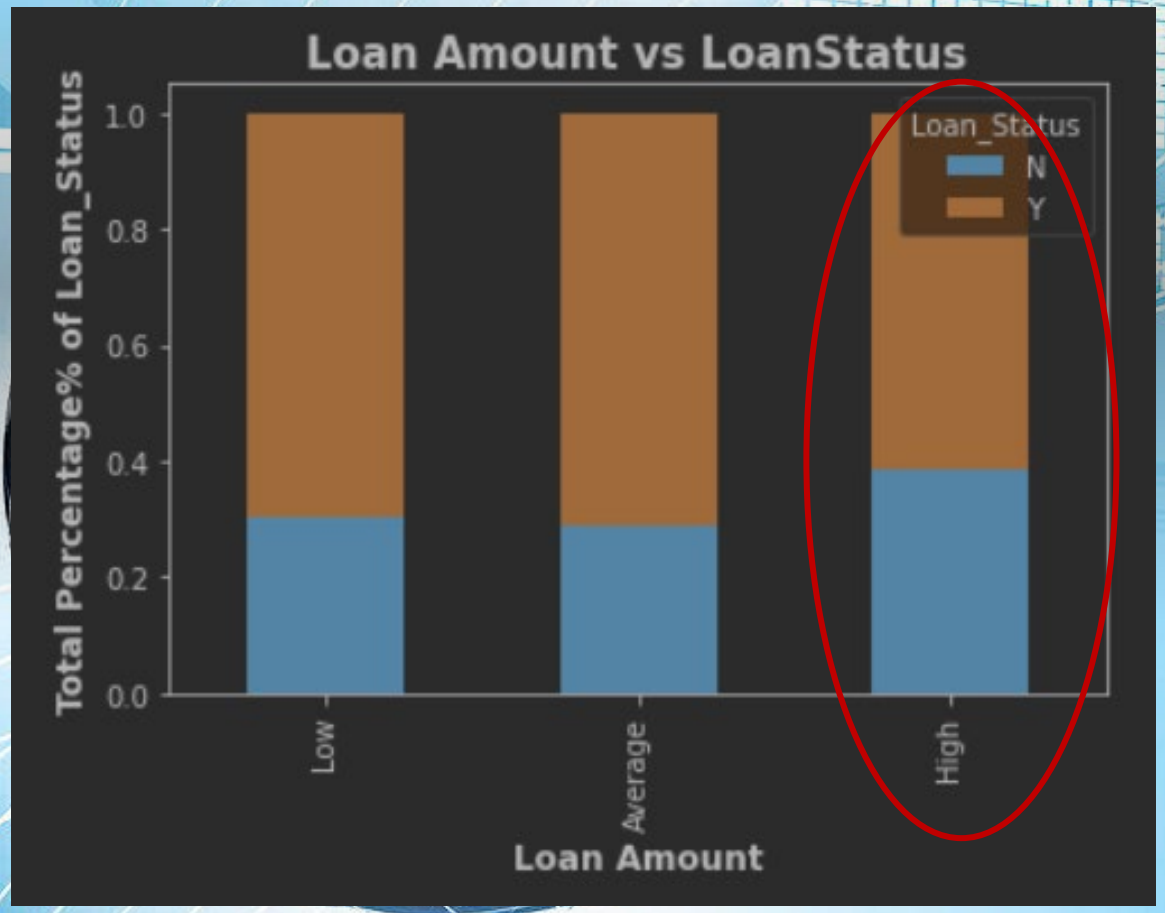
Rephased Hypothesis 2:

If the total income is low, the chances of getting loan approval is low

Numerical Independent Variable vs Target Variable (Loan Amount vs Loan Status)

```
bins=[0,100,200,700]
group=['Low','Average','High']
train['LoanAmount_bin']=pd.cut(train['LoanAmount'],bins,labels=group)
LoanAmount_bin=pd.crosstab(train['LoanAmount_bin'],train['Loan_Status'])
LoanAmount_bin.div(LoanAmount_bin.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True)
plt.title('Loan Amount vs LoanStatus', fontweight='bold', size=15)
plt.ylabel('Total Percentage% of Loan_Status', fontweight='bold', size=12)
plt.xlabel('Loan Amount', fontweight='bold', size=12)
plt.show()
```

	LoanAmount
count	592.000000
mean	146.412162
std	85.587325
min	9.000000
25%	100.000000
50%	128.000000
75%	168.000000
max	700.000000



From this chart, we can see that applicant with ‘High’ loan amount has lesser chance to get loan approved. This supports our hypothesis 3.

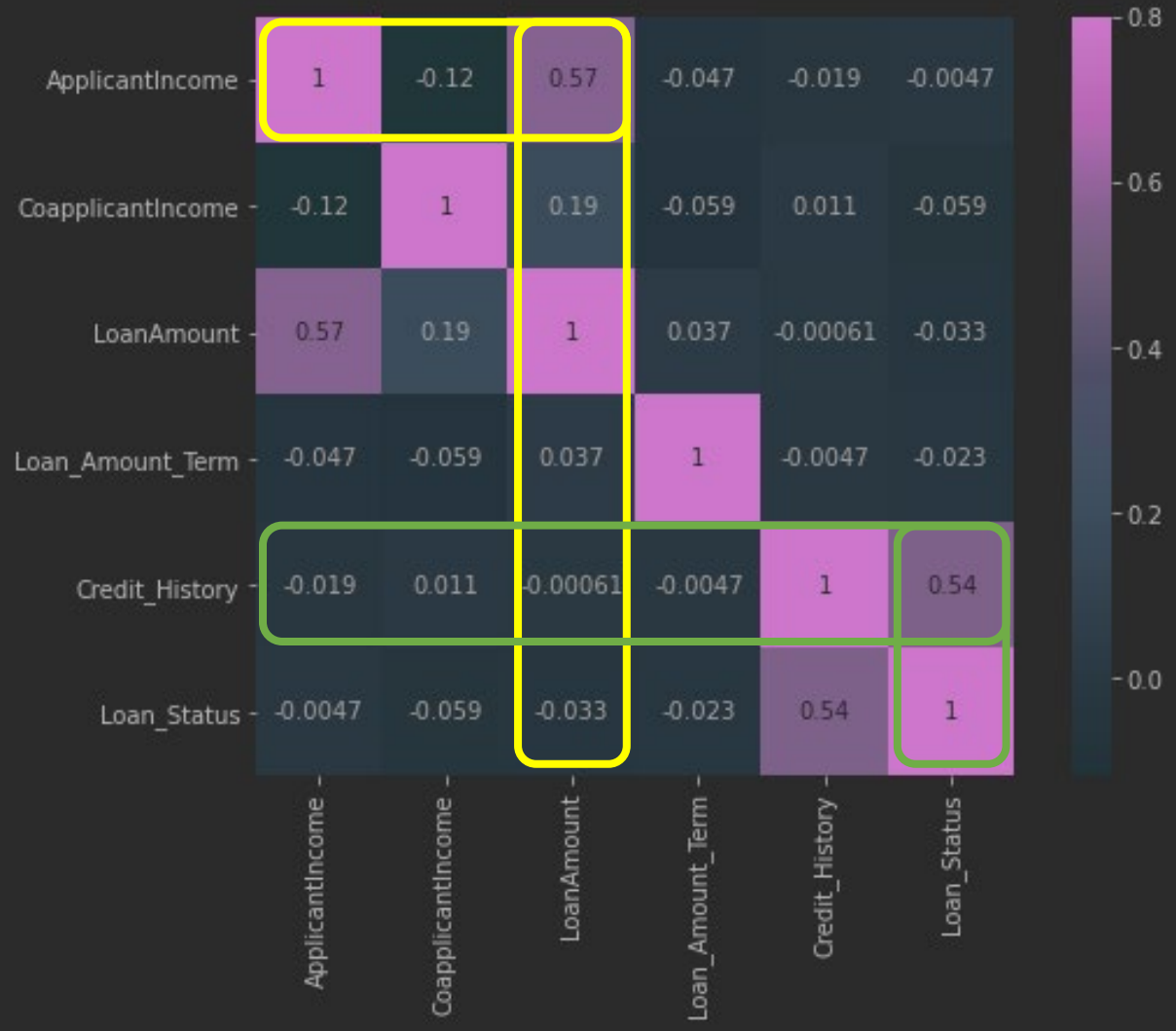
Hypothesis 3:

If the requested loan amount is high, the chances of getting loan approved is low

Correlations

```
matrix = checkcorr.corr()
f, ax = plt.subplots(figsize=(9,6))
sns.heatmap(matrix,vmax=.8,square=True,cmap="BuPu", annot = True)
```

```
<AxesSubplot:>
```



Using Heatmap from Seaborn Library, we can see the graphical representation of data in which values are represented as colours and also the measure of dependence between two variables

The most correlated variables are

-57% (ApplicantIncome — LoanAmount)

-54% (Credit_History — Loan_Status)

Revised Hypothesis(New Features)

Total Income

- As seen in bivariate analysis, we will combine the Applicant Income and Co applicant Income.
- If total income is high, chances of loan approval is high

Proven

Equated Monthly Instalment

- EMI is the monthly amount to be paid by the applicant to repay the loan.
- If Applicant has high EMI, chances of loan approval is low due to higher chance of default

Balanced Income

- Income left after paying EMI.
- If Balanced Income is high, chances of loan approval is high due to ability to pay back with ease

Based on the domain knowledge, we can come up new features that might affect the target variable.

Revised Hypothesis(New Features)

Total Income

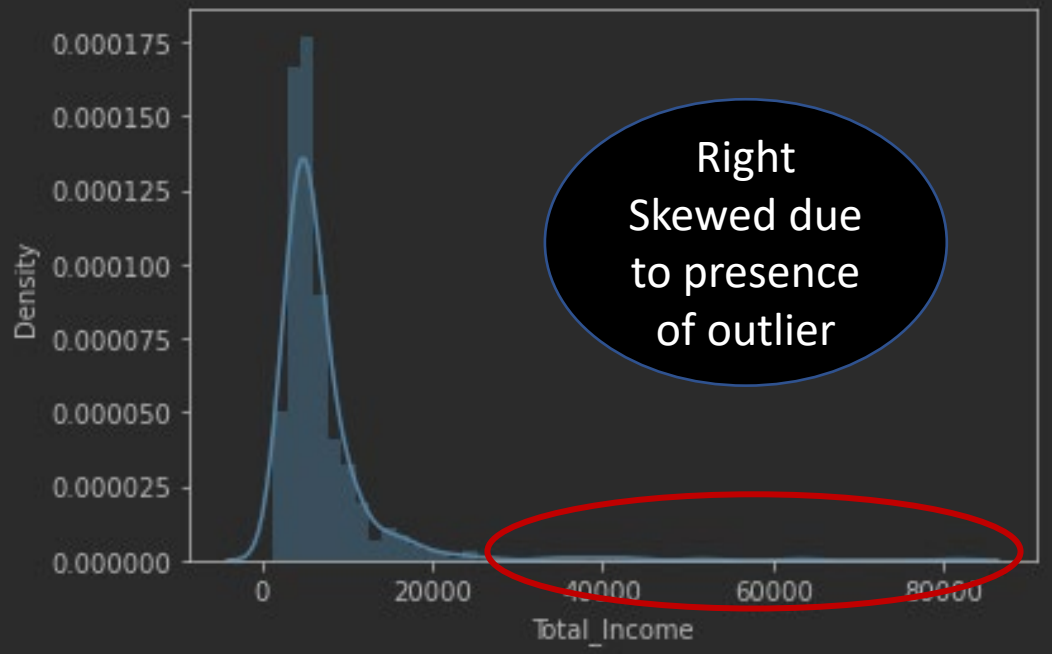
- As seen in bivariate analysis, we will combine the Applicant Income and Co-applicant Income.
- If total income is high, chances of loan approval is high

Proven

Proven from the initial Hypothesis

```
sns.distplot(train_model['Total_Income'])
```

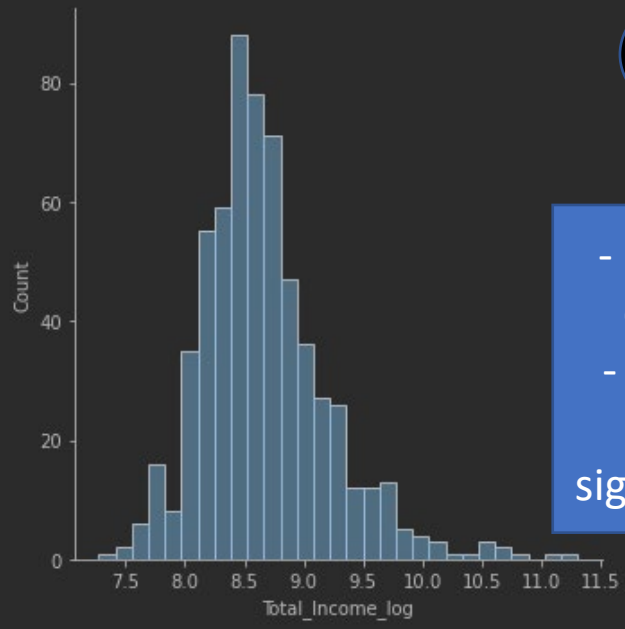
```
<AxesSubplot:xlabel='Total_Income', ylabel='Density'>
```



Right Skewed due to presence of outlier

```
train_model['Total_Income_log'] = np.log(train_model['Total_Income'])
test_model['Total_Income_log'] = np.log(train_model['Total_Income'])
sns.displot(train_model['Total_Income_log'])
```

```
<seaborn.axisgrid.FacetGrid at 0x1a417de8400>
```



Applied Log Function

- Distribution looks closer to normal
- Effect of extreme values are significantly subsided

Revised Hypothesis(New Features)

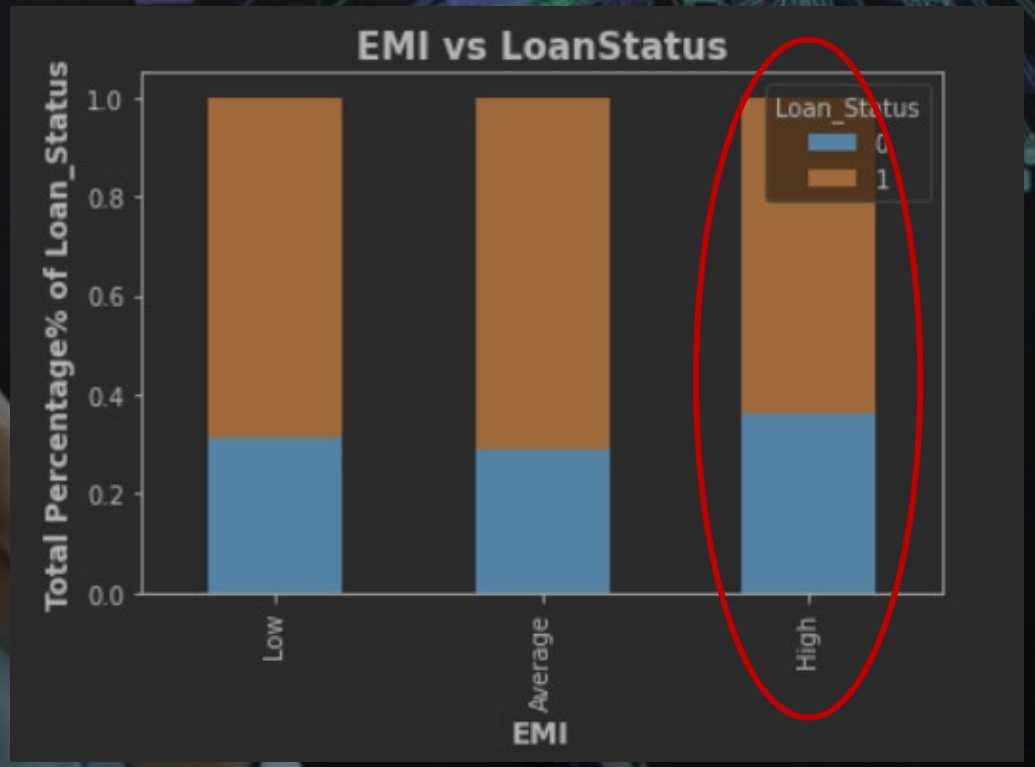
Equated Monthly Instalment

- EMI is the monthly amount to be paid by the applicant to repay the loan.
- If Applicant has high EMI, chances of loan approval is low due to higher chance of default

Formula for EMI

(EMI = Loan Amount/Loan Amount Term)

```
train_model['EMI']=train_model['LoanAmount']/train_model['Loan_Amount_Term']
```



Assumption

Interest rate is constant

Judging from the plot graph, we can see that if the EMI is high, the chances of not getting loan approved is higher

Revised Hypothesis(New Features)

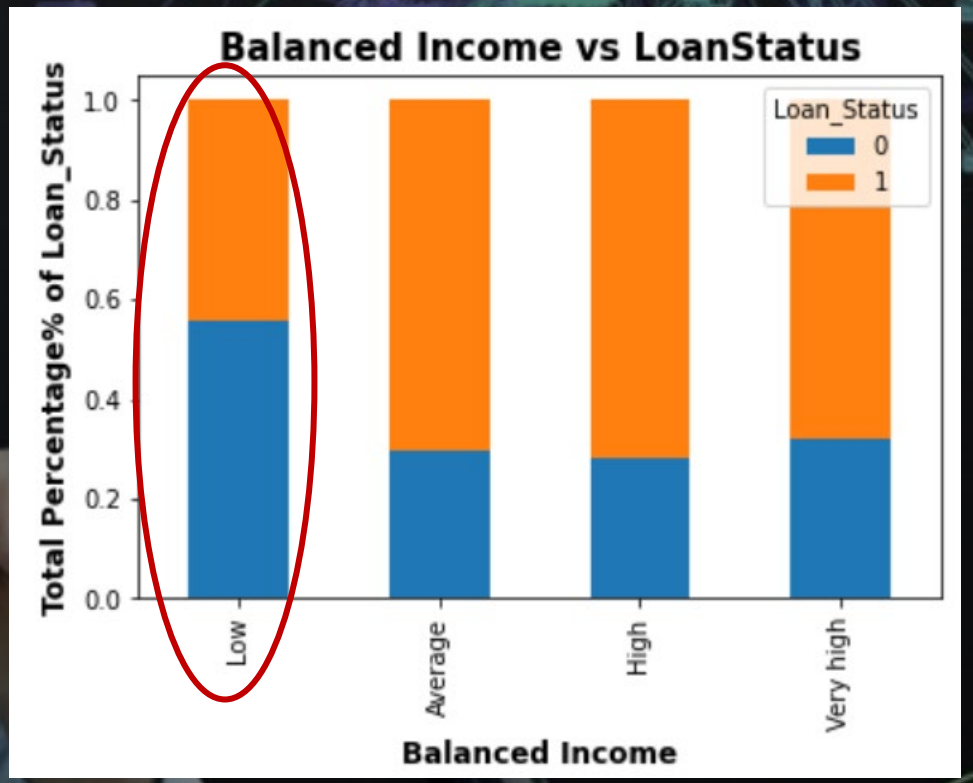
Balanced Income

- Total Income left after paying the EMI
- If Balanced Income is high, chances of loan approval is high due to ability to pay back with ease

Formula for Balanced Income

(Balanced Income = Total Income – (EMI*1000))

```
train_model['Balanced Income'] = train_model['Total_Income']-(train_model['EMI']*1000)
```



Judging from the plot graph, we can see that if the Balanced Income is low, the percentage of non-approved loans is higher

Feature Engineering
(Data Transform on Dependents and Loan Status variables into numerical format)

```
train['Dependents'].value_counts()
```

	Dependents
0	345
1	102
2	101
3+	51

```
train['Dependents'].replace('3+', 3, inplace=True)
test['Dependents'].replace('3+', 3, inplace=True)
```

```
train['Dependents'].value_counts()
```

	Dependents
0	360
1	102
2	101
3	51

Length: 4, dtype: int64 Open in new tab

```
train['Loan_Status'].value_counts()
```

	Loan_Status
Y	422
N	192

Length: 2, dtype: int64 Open in new tab

```
train['Loan_Status'].replace('Y', 1, inplace=True)
train['Loan_Status'].replace('N', 0, inplace=True)
```

```
train['Loan_Status'].value_counts()
```

	Loan_Status
1	422
0	192

Length: 2, dtype: int64 Open in new tab

Feature Engineering

What is One Hot Encoding?

Human-Readable | Machine-Readable

Pet	Cat	Dog	Turtle	Fish
Cat	1	0	0	0
Dog	0	1	0	0
Turtle	0	0	1	0
Fish	0	0	0	1
Cat	1	0	0	0

A method of converting data to prepare it for an algorithm and get a better prediction

A requirement for Python machine learning model as all data needs to be in numerical format

With One Hot Encoding, we convert each categorical value into new columns and assign a binary value of 1 or 0 to those columns

Feature Engineering (One Hot Encoding)

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	LoanAmount_log	Total_Income	Total_Income_log	EMI	Balanced Income
3	Male	Yes	0	Not Graduate	No	1.0	Urban	4.787492	4941.0	8.505323	0.333333	4607.666667
4	Male	No	0	Graduate	No	1.0	Urban	4.948760	6000.0	8.699515	0.391667	5608.333333
...	...	...	...	...	...	...	...	...	...	...	...	...
609	Female	No	0	Graduate	No	1.0	Rural	4.262680	2900.0	7.972466	0.197222	2702.777778
610	Male	Yes	3	Graduate	No	1.0	Rural	3.688879	4106.0	8.320205	0.222222	3883.777778
611	Male	Yes	1	Graduate	No	1.0	Urban	5.533389	8312.0	9.025456	0.702778	7609.222222
612	Male	Yes	2	Graduate	No	1.0	Urban	5.231109	7583.0	8.933664	0.519444	7063.555556
613	Female	No	0	Graduate	Yes	0.0	Semiurban	4.890349	4583.0	8.430109	0.369444	4213.555556

614 rows × 12 columns Open in new tab

Before One Hot Encoding :
12 columns

Dummy Variables Conversion(in Python)

```
train_model=pd.get_dummies(train_model)
```

Gender_Female	Gender_Male	Married_No	Married_Yes	Education_Graduate	Education_Not Graduate	Self_Employed_No	Self_Employed_Yes	Property_Area_Rural	Property_Area_Semiurban
0	1	1	0	1	0	1	0	0	0
0	1	0	1	1	0	1	0	1	0
0	1	0	1	1	0	0	1	0	0
0	1	0	1	0	1	1	0	0	0
0	1	1	0	1	0	1	0	0	0

5 rows × 18 columns Open in new tab

After One Hot Encoding :
18 columns

Feature Engineering (Removing Columns)

Removing columns such as
- ApplicantIncome
- CoapplicantIncome
as they were replaced by new variable Total_Income

- LoanAmount
- Loan_Amount_Term
as they were replaced by new variable EMI

-Any other data that was used purely for visualization purpose

```
train_model=train_model.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term','
```

Finalised dataset

	Dependents	Credit_History	LoanAmount_log	Total_Income	Total_Income_log	EMI	Balanced Income	Gender_Female	Gender_Male	Married_No
0	0	1.0	4.852030	5849.0	8.674026	0.355556	5493.444444	0	1	1
1	1	1.0	4.852030	6091.0	8.714568	0.355556	5735.444444	0	1	0
2	0	1.0	4.189655	3000.0	8.006368	0.183333	2816.666667	0	1	0
3	0	1.0	4.787492	4941.0	8.505323	0.333333	4607.666667	0	1	0
4	0	1.0	4.948760	6000.0	8.699515	0.391667	5608.333333	0	1	1

Married_Yes	Education_Graduate	Education_Not Graduate	Self_Employed_No	Self_Employed_Yes	Property_Area_Rural	Property_Area_Semiurban	Property_Area_Urban
0	1	0	1	0	0	0	1
1	1	0	1	0	1	0	0
1	1	0	0	1	0	0	1
1	0	1	1	0	0	0	1
0	1	0	1	0	0	0	1

Predictive Analytics

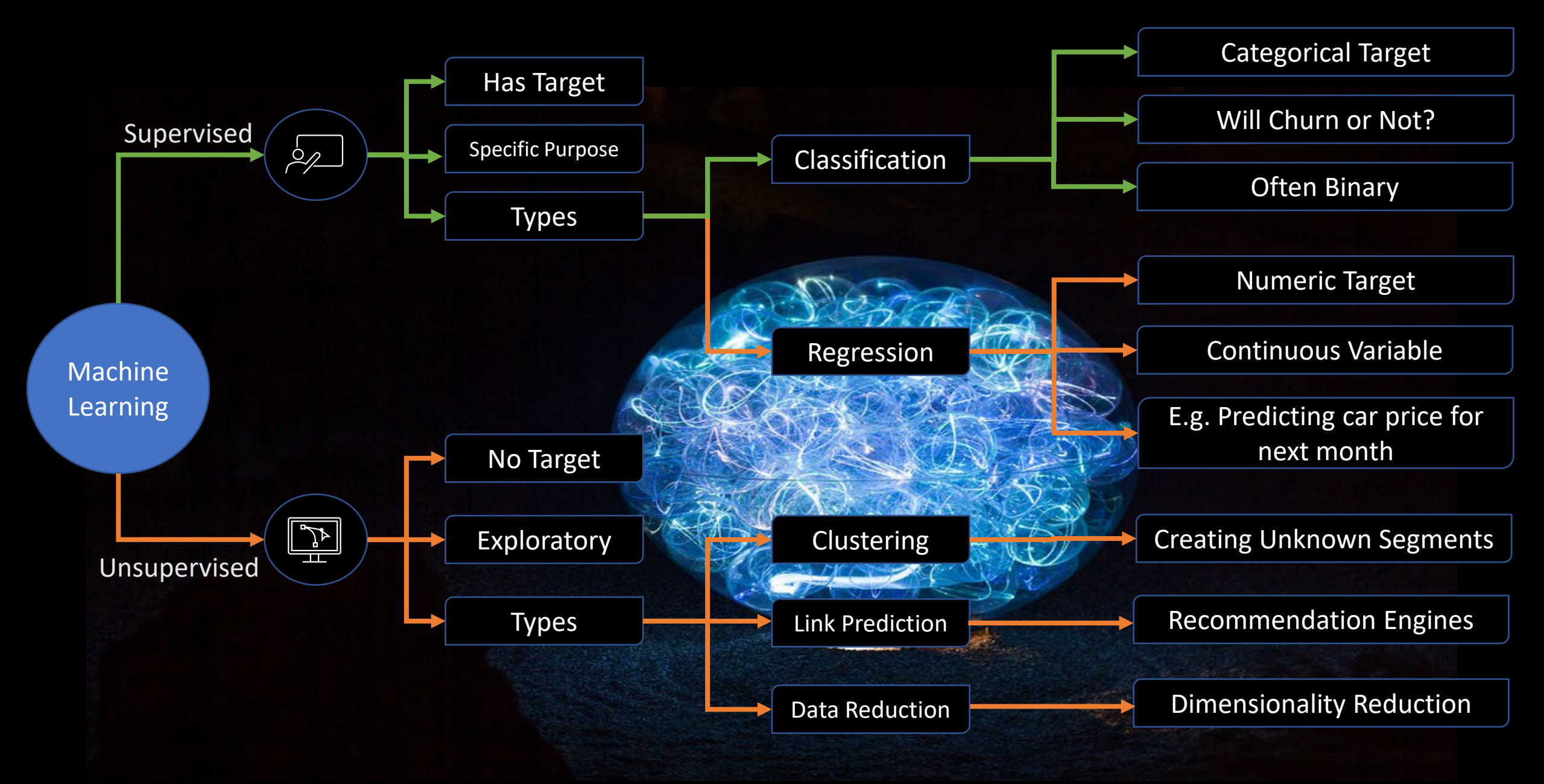
Utilizes models for predicting events

Machine Learning Toolkits

Scikit Learn

AutoGluon

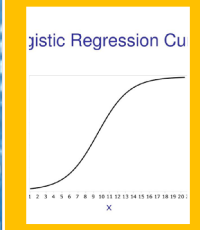
AutoAI



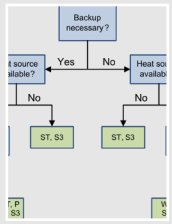
Machine Learning Technique Choice

Available Estimators for Binary Classification

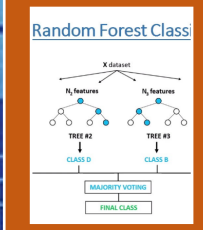
Binary Classification



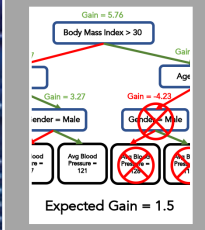
Logistic Regression



Decision Tree

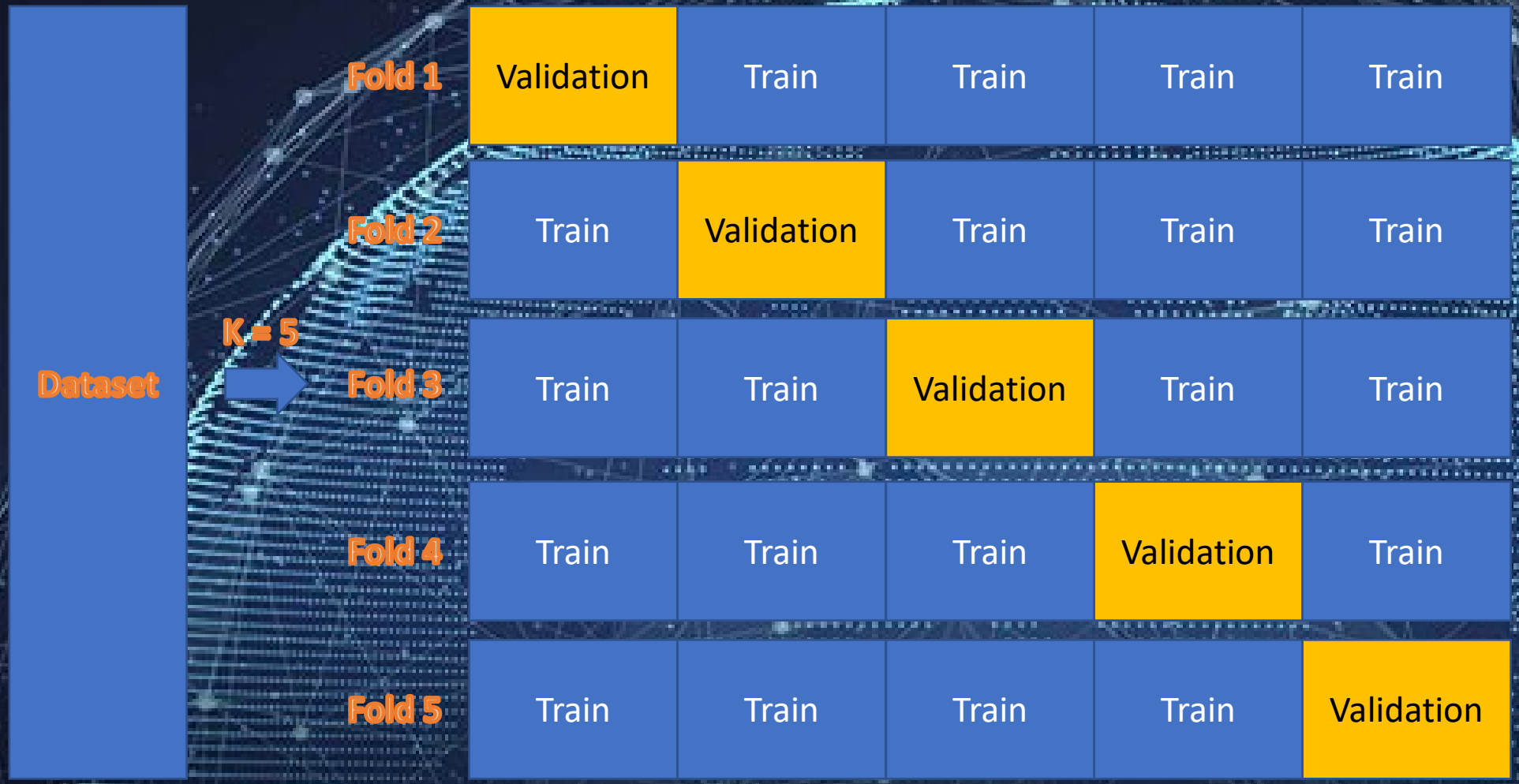


Random Forest



Gradient Boosted Tree (XGBoost)

K-folds cross-validation

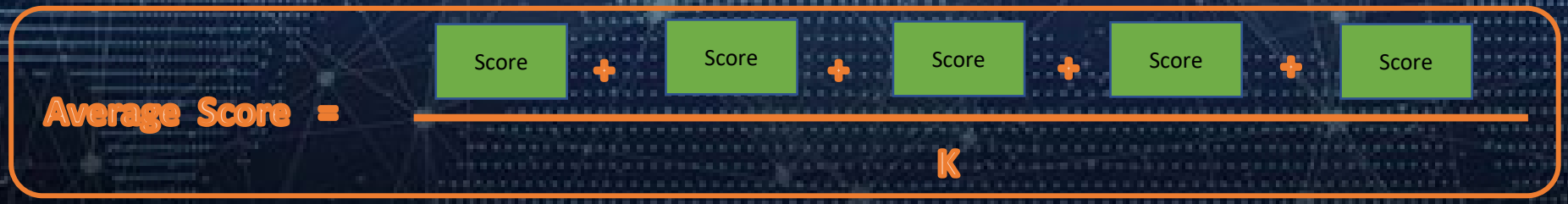


-a technique used to assess a machine learning model and test its performance

-Uses all the data available for training and validation

-Gives more information on algorithm performance

-Crucial where amount of data available is limited

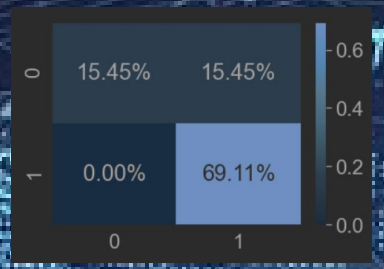


Logistics Regression with k-Fold CV(Sklearn)

Fold 1

1 of kfold 5

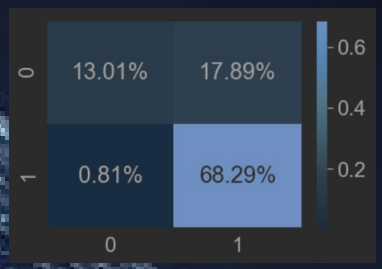
	precision	recall	f1-score	support
0	1.00	0.50	0.67	38
1	0.82	1.00	0.90	85
accuracy			0.85	123
macro avg	0.91	0.75	0.78	123
weighted avg	0.87	0.85	0.83	123



Fold 2

2 of kfold 5

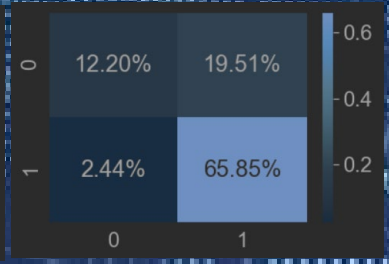
	precision	recall	f1-score	support
0	0.94	0.42	0.58	38
1	0.79	0.99	0.88	85
accuracy			0.81	123
macro avg	0.87	0.70	0.73	123
weighted avg	0.84	0.81	0.79	123



Fold 3

3 of kfold 5

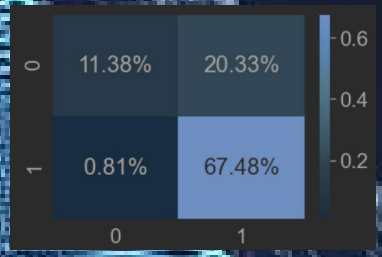
	precision	recall	f1-score	support
0	0.83	0.38	0.53	39
1	0.77	0.96	0.86	84
accuracy			0.78	123
macro avg	0.80	0.67	0.69	123
weighted avg	0.79	0.78	0.75	123



Fold 4

4 of kfold 5

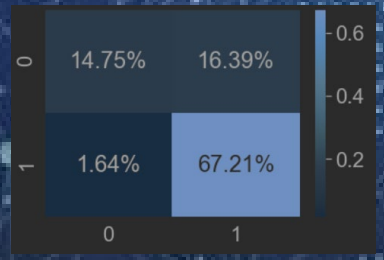
	precision	recall	f1-score	support
0	0.93	0.36	0.52	39
1	0.77	0.99	0.86	84
accuracy			0.79	123
macro avg	0.85	0.67	0.69	123
weighted avg	0.82	0.79	0.75	123



Fold 5

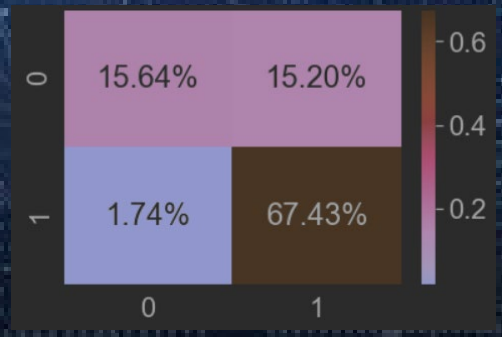
5 of kfold 5

	precision	recall	f1-score	support
0	0.90	0.47	0.62	38
1	0.80	0.98	0.88	84
accuracy			0.82	122
macro avg	0.85	0.72	0.75	122
weighted avg	0.83	0.82	0.80	122



Average Score

Mean Validation Accuracy 0.8094628815140611
Mean Validation Precision 0.7907258362141825
Mean Validation Recall 0.9833613445378152

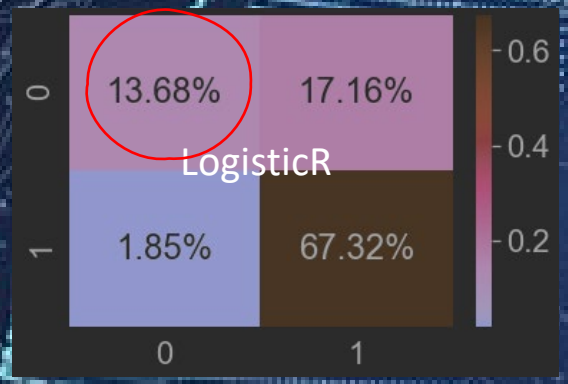


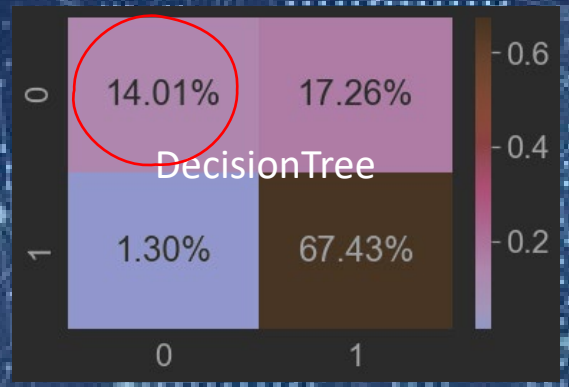
Validate Data Models

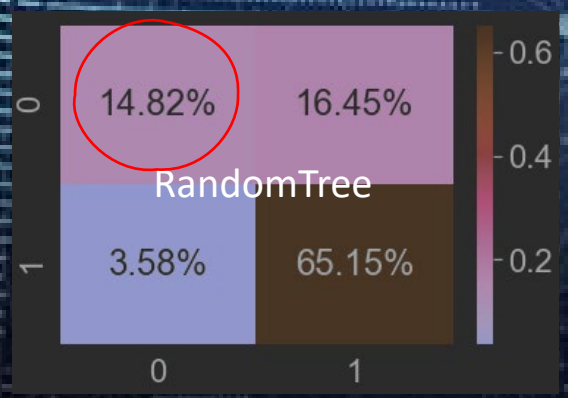
Chosen Base Model for Sklearn

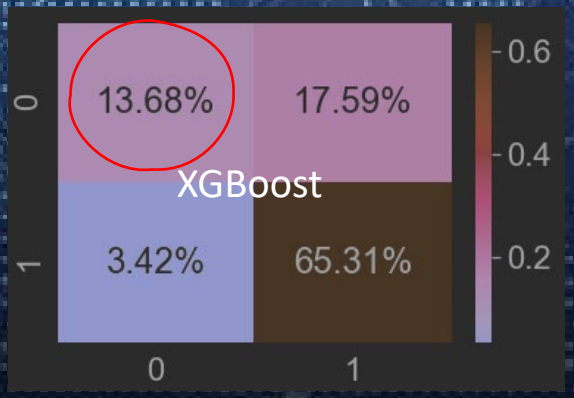
	Accuracy	Precision	Recall	Roc_Auc	F1
LogisticR	0.809463	0.790726	0.983361	0.807331	0.876585
DecisionTree	0.809423	0.790662	0.983417	0.685620	0.876568
RandomTree	0.788245	0.797829	0.926555	0.805451	0.857387
XGBoost	0.762215	0.795797	0.879300	0.805451	0.835467

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive}$$

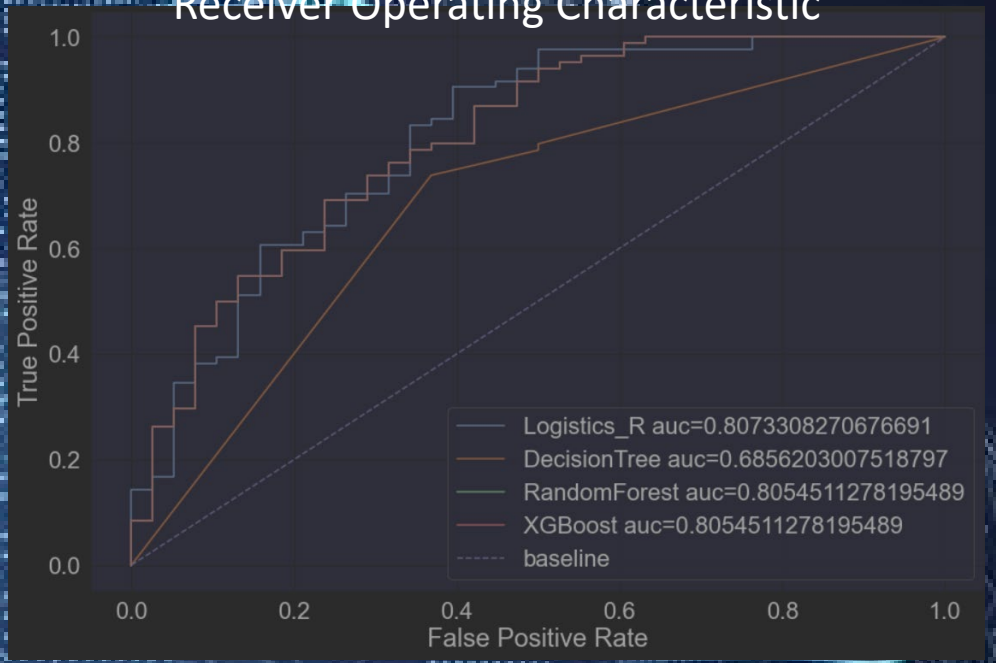








Receiver Operating Characteristic



Recall is more important where Overlooked Cases (False Negatives) are more costly than False Alarms (False Positive)

We realised the number of True Negative is too low, which will be explained in later section on why this is occurring

AutoGluon

Upon using fit() method, Autogluon will analyse that it is a binary problem

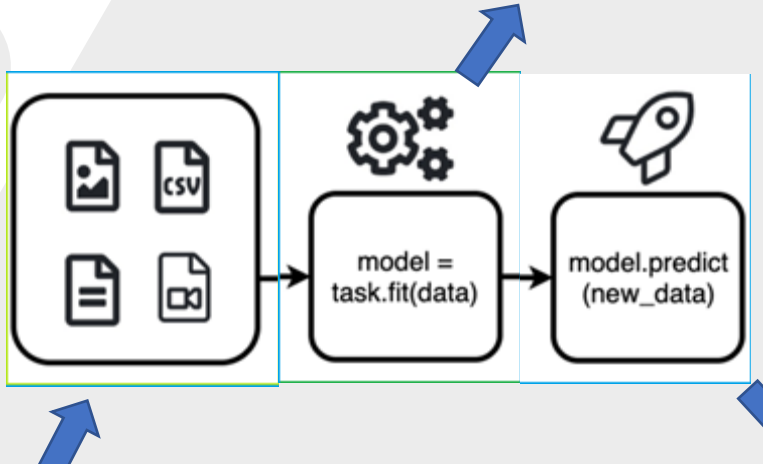
```
metric = 'roc_auc'  # specify your evaluation metric here
predictor = TabularPredictor(label_column, eval_metric=metric).fit(train_data, presets='best_quality')
predictor.leaderboard(test_data, silent=True)
```

```
No path specified. Models will be saved in: "AutogluonModels/ag-20211230_220402/"
Presets specified: ['best_quality']
Beginning AutoGluon training ...
AutoGluon will save models to "AutogluonModels/ag-20211230_220402/"
AutoGluon Version:  0.3.1
Train Data Rows:    491
Train Data Columns: 12
Preprocessing data ...
AutoGluon infers your prediction problem is: 'binary' (because only two unique label-values observed).
        2 unique label values:  [0, 1]
        If 'binary' is not the correct problem_type, please manually specify the problem_type argument
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    11410.74 MB
        Train Data (Original)  Memory Usage: 0.18 MB (0.0% of available memory)
```

We will be using Amazon's AutoGluon (open source ML toolkit) with Google Collaboratory (IDE)

3 Easy steps to Predict Results

1. Feed in data such as image, text, video. In our case csv file
2. Train multiple models with .fit() method on the data
3. Use the trained models to make predictions



LoanAmount_log	Total_Income	Total_Income_log	EMI	Balanced Income
4.852030	5849.0	8.674026	0.355556	5493.444444
4.852030	6091.0	8.714568	0.355556	5735.444444
4.189655	3000.0	8.006368	0.183333	2816.666667
4.787492	4941.0	8.505323	0.333333	4607.666667
4.948760	6000.0	8.699515	0.391667	5608.333333

Pre-cleansed csv file with added new Revised features but no one-hot encoding

```
testing_data = TabularDataset("https://github.com/Lig
predictor.predict(testing_data, model='RandomForestGi
```

```
Loaded data from: https://github.com/Lightgamers/Loan
0      1
1      1
2      1
3      1
4      1
      ..
362    1
363    1
364    1
365    1
366    1
Name: Loan_Status, Length: 367, dtype: int64
```

the collaboratory

(IDE)

AutoGluon

```
metric = 'roc_auc'  # specify your evaluation metric here
predictor = TabularPredictor(label_column, eval_metric=metric).fit(train_data, presets='best_quality')
predictor.leaderboard(test_data, silent=True)
```

	model	score_test	score_val	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order
0	RandomForestGini_BAG_L1	0.792735	0.745291	0.224070	0.148235	0.985045	0.224070	0.148235	0.985045	1	True	5
1	CatBoost_BAG_L1	0.790598	0.766272	0.011178	0.020931	2.848943	0.011178	0.020931	2.848943	1	True	7
2	XGBoost_BAG_L1	0.781746	0.739558	0.167597	0.040845	1.202494	0.167597	0.040845	1.202494	1	True	11
3	RandomForestEntr_BAG_L1	0.781593	0.741453	0.123988	0.149804	1.086812	0.123988	0.149804	1.086812	1	True	6
4	LightGBM_BAG_L1	0.780372	0.741695	0.038489	0.044147	1.315540	0.038489	0.044147	1.315540	1	True	4
5	LightGBMLarge_BAG_L1	0.772283	0.726167	0.051426	0.042465	2.126164	0.051426	0.042465	2.126164	1	True	12

Based on what we learned from our base models in Sklearn, we know our recall is almost close to 100%. Therefore, we choose RoC_AuC as our optimized metric

RoC is a probability curve that represents the degree or measure of separability.

It tells how much the model is capable of distinguishing between classes.

Higher the AUC, the better the model at predicting 0 classes as 0, 1 classes as 1 ,which fulfil what we need for loan_status

RandomForestGini Model come out top with the highest score which we will pick for the model for AutoGluon

```
#y_pred = predictor.predict(test_data_nolab )
print("Predictions:  \n", y_pred)
perf = predictor.evaluate_predictions(y_true=y_test, y_pred=pred_probs, auxiliary_metrics=True )
```

```
Evaluation: roc_auc on test data: 0.76007326007326
Evaluations on test data:
{
    "roc_auc": 0.76007326007326,
    "accuracy": 0.8048780487804879,
    "balanced_accuracy": 0.6991758241758241,
    "mcc": 0.5371074443955691,
    "f1": 0.8736842105263158,
    "precision": 0.7830188679245284,
    "recall": 0.9880952380952381
}
Predictions:
```

the collaboratory (IDE)

And by running evaluate_predictions() method, we can see the evaluation scores that we can use for comparison later

AutoGluon

finding out the metadata

```
print("AutoGluon infers problem type is: ", predictor.problem_type)
print("AutoGluon identified the following types of features:")
print(predictor.feature_metadata)
```

```
AutoGluon infers problem type is:  binary
AutoGluon identified the following types of features:
('category', [])  : 1 | ['Property_Area']
('float', [])     : 5 | ['LoanAmount_log', 'Total_Income', 'Total_Income_log', 'EMI', 'Balanced Income']
('int', [])       : 1 | ['Dependents']
('int', ['bool']) : 5 | ['Gender', 'Married', 'Education', 'Self_Employed', 'Credit_History']
```

Additional information such as

Probability score of the records

```
pred_probs = predictor.predict_proba(test_data, model='RandomForestGini_BAG_L1')
pred_probs.head(5)
```

	0	1
533	0.203333	0.796667
544	0.023333	0.976667
41	0.243333	0.756667
148	0.420000	0.580000
111	0.020000	0.980000

Feature importance of variables

```
predictor.feature_importance(test_data, model='RandomForestGini_BAG_L1')
```

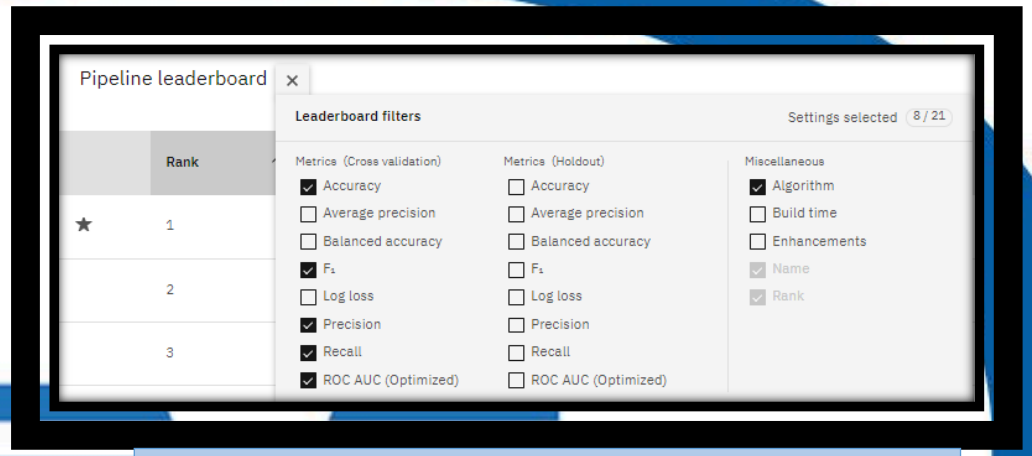
```
Computing feature importance via permutation shuffling for 12 features using 12
        5.11s   = Expected runtime (1.7s per shuffle set)
        0.91s   = Actual runtime (Completed 3 of 3 shuffle sets)
```

	importance	stddev	p_value	n	p99_high	p99_low
Credit_History	0.283171	0.014282	0.000423	3	0.365008	0.201333
LoanAmount_log	0.028897	0.016113	0.044942	3	0.121226	-0.063431
Balanced Income	0.019689	0.014163	0.068865	3	0.100844	-0.061466
Total_Income	0.013075	0.016793	0.154947	3	0.109302	-0.083152
Married	0.010328	0.009279	0.096851	3	0.063499	-0.042843
Property_Area	0.008445	0.011050	0.158306	3	0.071761	-0.054870
Self_Employed	0.007580	0.002291	0.014563	3	0.020708	-0.005548
EMI	0.007326	0.016689	0.263230	3	0.102953	-0.088301
Total_Income_log	0.004375	0.024077	0.391379	3	0.142340	-0.133590
Gender	0.000102	0.008760	0.492888	3	0.050298	-0.050095
Education	-0.001272	0.014058	0.555065	3	0.079284	-0.081828
Dependents	-0.009208	0.005497	0.949448	3	0.022292	-0.040709

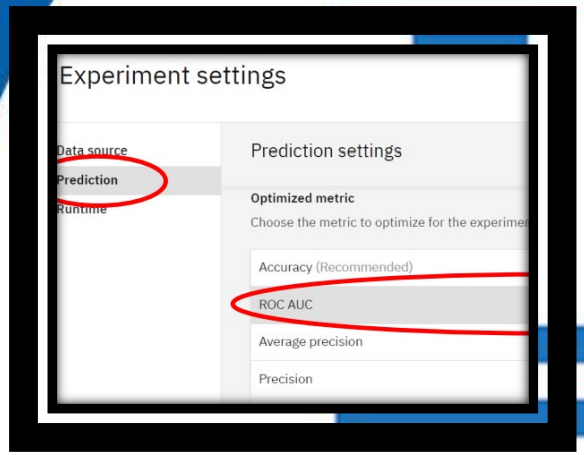
IBM AUTOAI

LoanAmount_log	Total_Income	Total_Income_log	EMI	Balanced Income
4.852030	5849.0	8.674026	0.355556	5493.444444
4.852030	6091.0	8.714568	0.355556	5735.444444
4.189655	3000.0	8.006368	0.183333	2816.666667
4.787492	4941.0	8.505323	0.333333	4607.666667
4.948760	6000.0	8.699515	0.391667	5608.333333

We fed the same dataset to AutoAI for comparison (Pre-cleansed csv file with added new Revised features but no one-hot encoding)

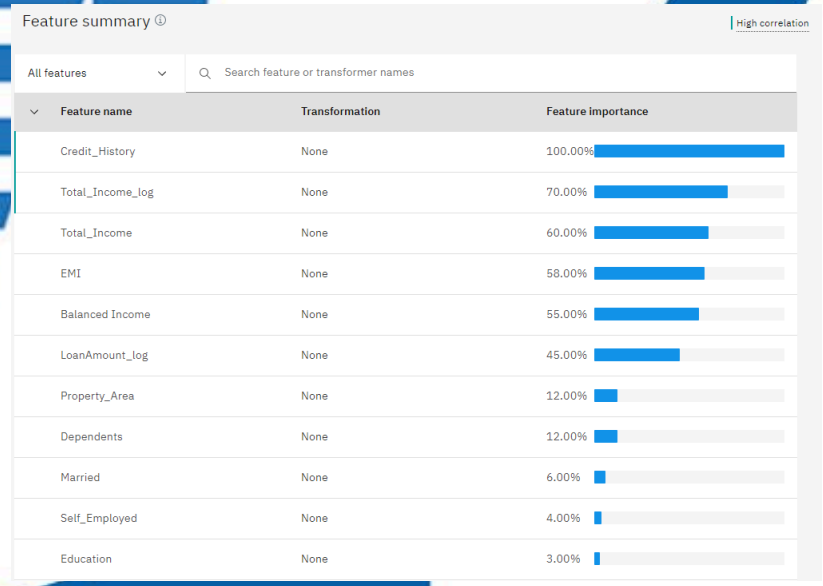


Under Pipeline leaderboard filters,
We select the metrics that we want to see



In Experiment Setting under Prediction, we select ROC AUC option

Inspect every model to check the feature summary that does not include AutoAI self created feature engineering variables

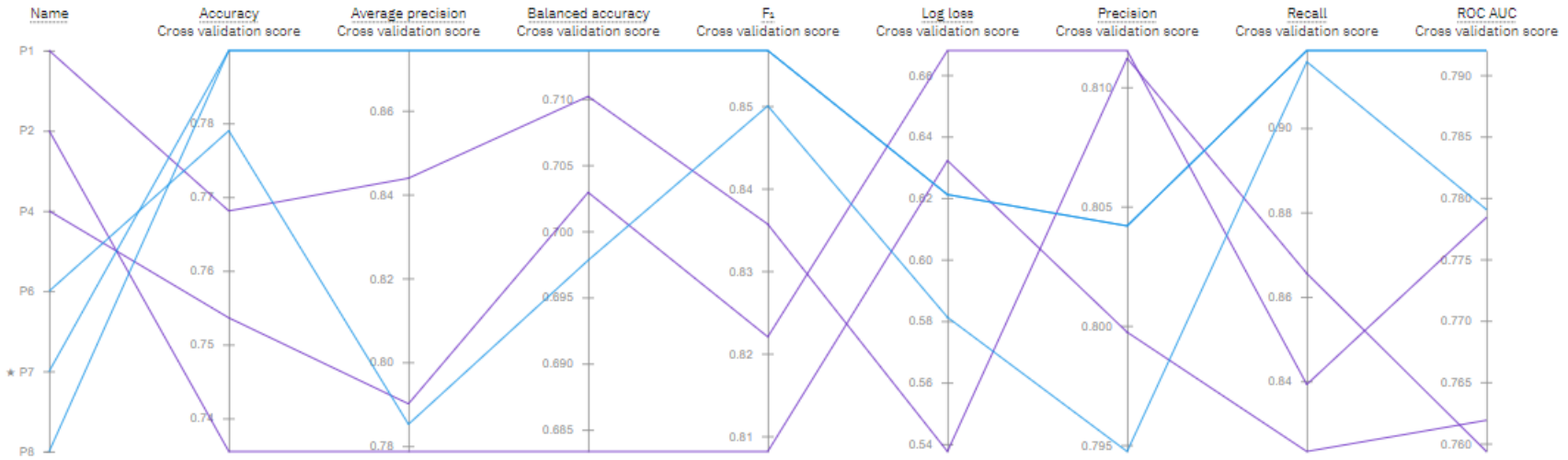


IBM AUTOAI

Experiment summary | Pipeline comparison

Relationship map

Prediction column: Loan_Status



Among all Pipelines, Pipeline 1 and 2 does not contains newly added Feature Engineering

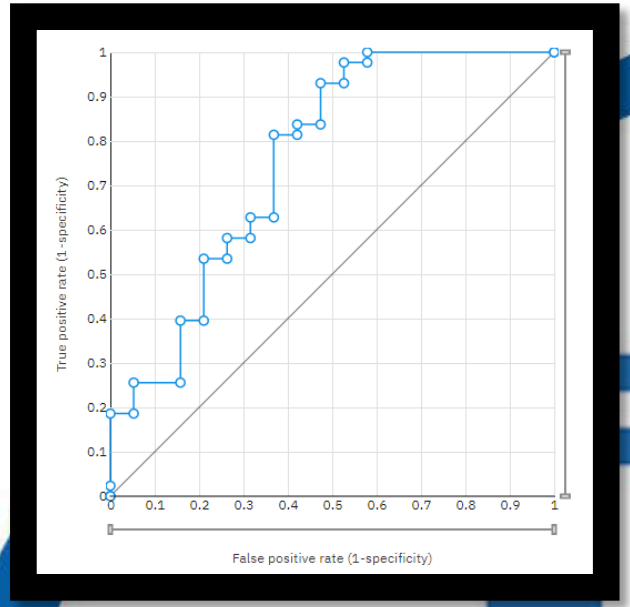
We choose Pipeline 1 because it has the higher overall metrics out of the 2 based on pipeline comparison

Pipeline leaderboard

	Rank	Name	Algorithm	ROC AUC (Optimized) Cross Validation	Accuracy Cross Validation	F1 Cross Validation	Precision Cross Validation	Recall Cross Validation
★	1	Pipeline 7	Extra Trees Classifier	0.792	0.790	0.857	0.804	0.918
	2	Pipeline 8	Extra Trees Classifier	0.792	0.790	0.857	0.804	0.918
	3	Pipeline 6	Extra Trees Classifier	0.779	0.779	0.850	0.795	0.916
	4	Pipeline 4	Snap Logistic Regression	0.778	0.754	0.822	0.812	0.839
	5	Pipeline 2	Snap Logistic Regression	0.762	0.736	0.808	0.800	0.823
	6	Pipeline 1	Snap Logistic Regression	0.759	0.768	0.836	0.811	0.866

IBM AUTOAI

From Pipeline 1, we can see that the model fulfils the expectation and generally having same metric scores as previous toolkits



Model information

Experiment parameters

Prediction column	Loan_Status
Number of features	12
Number of evaluation instances	62
Created on	30/12/2021, 01:22:00

We noticed that True Negative is low as well in the confusion matrix too, this will be explained in later section

Accuracy	0.806
Area under ROC	0.758
Precision	0.804
Recall	0.953
F1	0.872
Average precision	0.914
Log loss	0.487

Observed	Predicted		
	1	0	Percent correct
1	41	2	95.3%
0	10	9	47.4%
Percent correct	80.4%	81.8%	80.6%

Revised Hypothesis solidify with Feature Importance

Total Income

Equated Monthly Instalment

Balanced Income

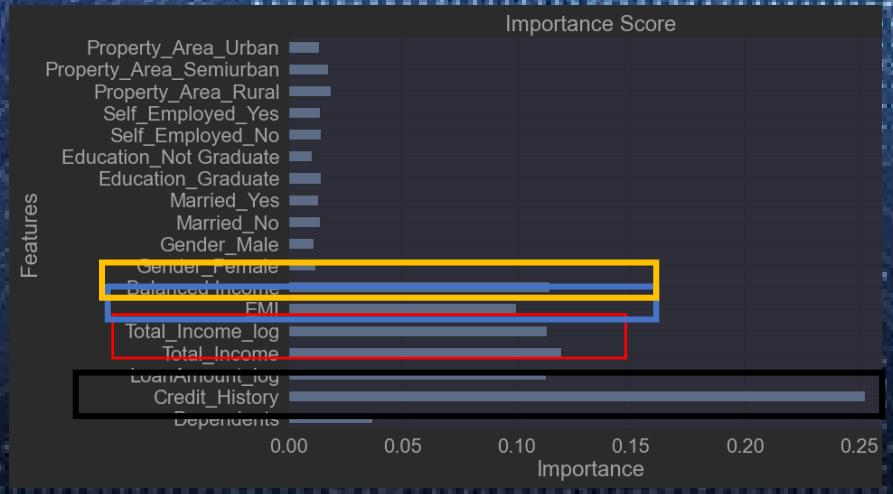
Feature name	Transformation	Feature importance
Credit_History	None	100.00%
Total_Income_log	None	70.00%
Total_Income	None	60.00%
EMI	None	58.00%
Balanced Income	None	55.00%
LoanAmount_log	None	45.00%
Property_Area	None	12.00%
Dependents	None	12.00%
Married	None	6.00%
Self_Employed	None	4.00%
Education	None	3.00%
Gender	None	0.00%

i) Feature Importances from AutoAI

Observation

1) Credit History is main determining feature that has the greatest influence over dependent variable (Loan_Status)

2) The revised added hypothesis of Total Income, Equated Monthly Instalment and Balanced Income shows the importance and the correlations comparing to the rest of the features. It proves the correctness of our hypothesis.



ii) Feature Importances from Sklearn

```
predictor.feature_importance(test_data, model='RandomForestGini_BAG_L1')

Computing feature importance via permutation shuffling for 12 features using 1
        5.11s   = Expected runtime (1.7s per shuffle set)
        0.91s   = Actual runtime (Completed 3 of 3 shuffle sets)
```

	importance	stddev	p_value	n	p99_high	p99_low
Credit_History	0.283171	0.014282	0.000423	3	0.365008	0.201333
LoanAmount_log	0.028897	0.016113	0.044942	3	0.121226	-0.063431
Balanced Income	0.019689	0.014163	0.068865	3	0.100844	-0.061466
Total_Income	0.013075	0.016793	0.154947	3	0.109302	-0.083152
Married	0.010328	0.009279	0.096851	3	0.063499	-0.042843
Property_Area	0.008445	0.011050	0.158306	3	0.071761	-0.054870
Self_Employed	0.007580	0.002291	0.014563	3	0.020708	0.005548
EMI	0.007326	0.016689	0.263230	3	0.102953	-0.088301
Total_Income_log	0.004375	0.024077	0.391379	3	0.142340	-0.133590
Gender	0.000102	0.008760	0.492888	3	0.050298	-0.050095
Education	-0.001272	0.014058	0.555065	3	0.079284	-0.081828
Dependents	-0.009208	0.005497	0.949448	3	0.022292	-0.040709

iii) Feature Importances from Autogluon

Comparison on different ML tools

Advantages

scikit-learn

- Offer most useful and robust open-source library
- Provides a selection of efficient tools for machine learning and statistical modelling

IBM Watson

- Automates everything from data preparation, model development, feature engineering, and hyperparameter optimization
- Generates model pipelines that are customized for the predictive modeling problem, encourage user for further experimentation

AutoGluon

- Simple and fast optimization or hyperparameter tuning for all of the models
- Provides prototype solutions in a few lines of codes with no prior machine learning knowledge

Disadvantages

- Required to know Python language prior
- Needs to possess strong knowledge on data structure and arrays: ie needs to know how to work with Data Frame from Pandas

- Does not support Integrated Visual Recognition anymore
- Limitation of Data usage per free account

- New toolkit was launched only recently and currently lacking of community involvement unlike tools like scikit-learn that has been around for years
- Several useful tools are still work in progress ie. Confusion Matrix

Testing Result on different ML tools

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	LoanAmount_log	Total_Income	Total_Income_log	EMI	Balanced Income	Actual Loan_Status
1	Male	No	0	Graduate	No	1.0	Urban	4.85	5849	8.67	0.356	5493.44	1
2	Male	Yes	1	Graduate	No	1.0	Rural	4.85	6091	8.71	0.356	5735.44	0
3	Male	Yes	0	Graduate	Yes	1.0	Urban	4.19	3000	8.01	0.183	2816.67	1
4	Male	Yes	3	Graduate	No	0.0	Semiurban	5.06	5540	8.62	0.439	5101.11	0
5	Nil	Nil	0	Nil	Nil	1.0	Nil	Nil	Nil	Nil	Nil	Nil	Nil
6	Nil	Nil	0	Nil	Nil	0.0	Nil	Nil	Nil	Nil	Nil	Nil	Nil

	AutoAI Probability on Loan_Status(%)		Resulted Loan_Status
	0	1	
1	50.04	49.96	0
2	15	85	1
3	39	61	1
4	89	11	0
5	19	81	1
6	94	6	0

	AutoGluon Probability on Loan_Status(%)		Resulted Loan_Status
	0	1	
1	9	91	1
2	74	26	0
3	12	88	1
4	86	14	0
5	74	26	0
6	83	17	0

1) From Row 1 to 4, we will use records from the training dataset that has the actual Loan_Status result
2) From Row 5 to 6, we take the credit history of 1 and 0 to see the impact since credit history has the highest importance

We observed that AutoAI has different probability and incorrect prediction than AutoGluon on several results , this will be explained why in later section

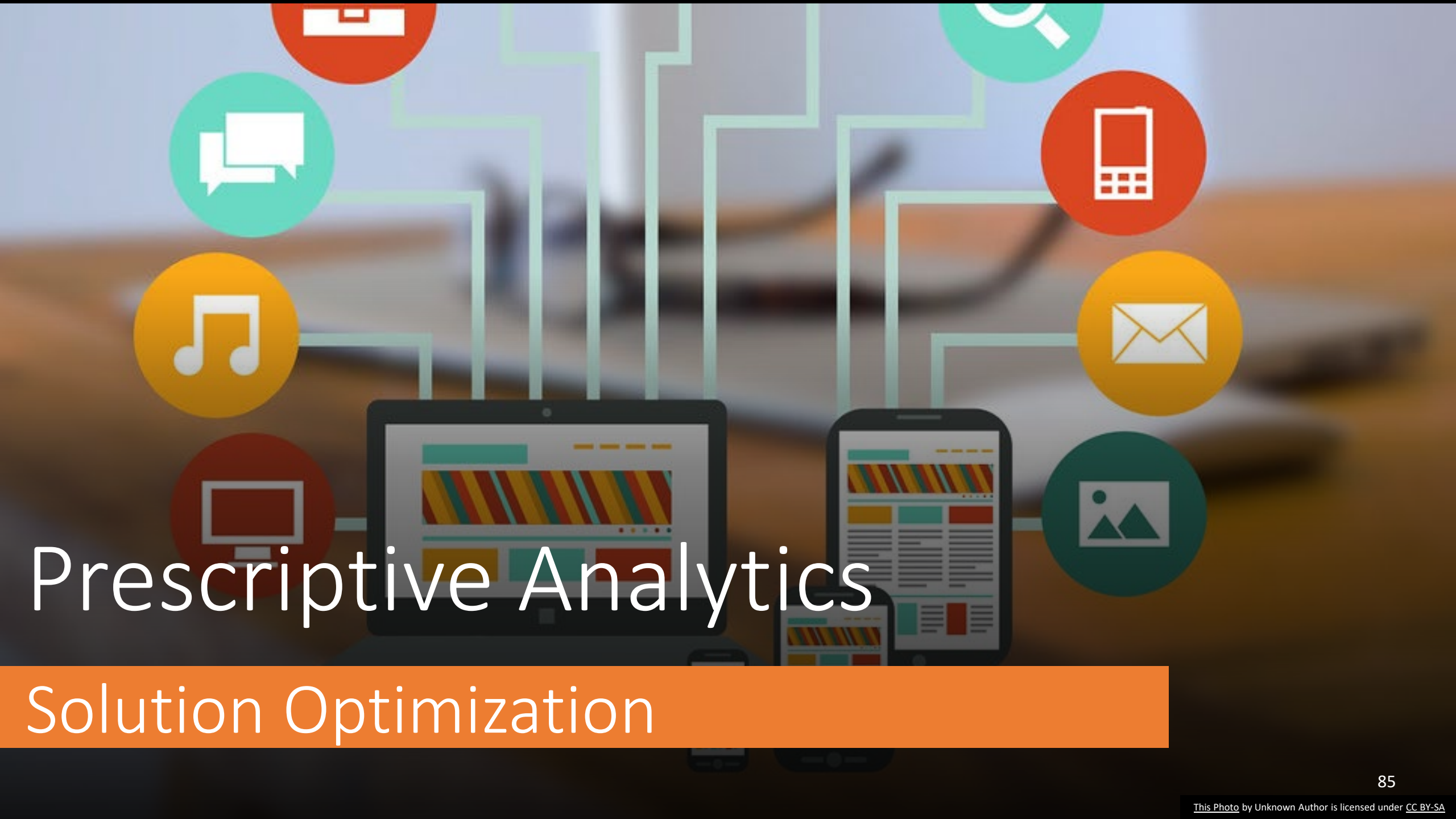
Choosing the final model

	Accuracy(%)	Precision(%)	Recall(%)	F1(%)	Roc_AuC(%)
Logistic Regression (SciKit-Learn)	80.94	79.07	98.33	87.65	80.73
Random ForestGini (AutoGluon)	78.05	78.79	92.86	85.25	79.27
Snap Logistic Regression (AutoAI)	80.6	80.4	95.3	87.2	75.9

After much consideration, we decided to choose Random ForestGini from AutoGluon

Reason

1. All the metrics scores are not too different much from one another. The highest difference is only 5.47 points in Recall percentage.
2. We picked the 2nd option because of the advantages that Autogluon can benefit us (in our standpoint) a startup company with limited resources and capital
3. Autogluon distinct advantages such as
   - Open-source toolkit that does not need pay a single dime
   - A lot less complex than the other 2 options
   - Helps to save time which can be done in a few lines of codes

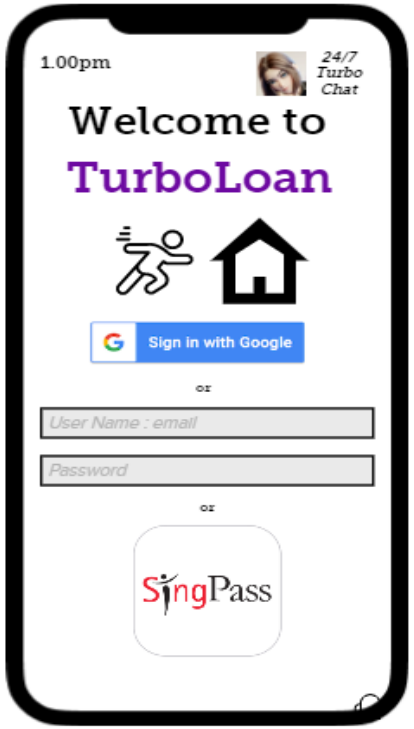


Prescriptive Analytics

Solution Optimization

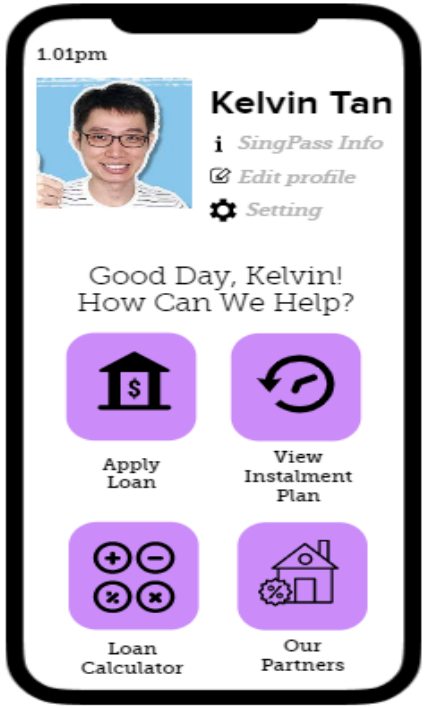
Model Deployment (Master Prototype – Iteration 2)

1
Sign-in

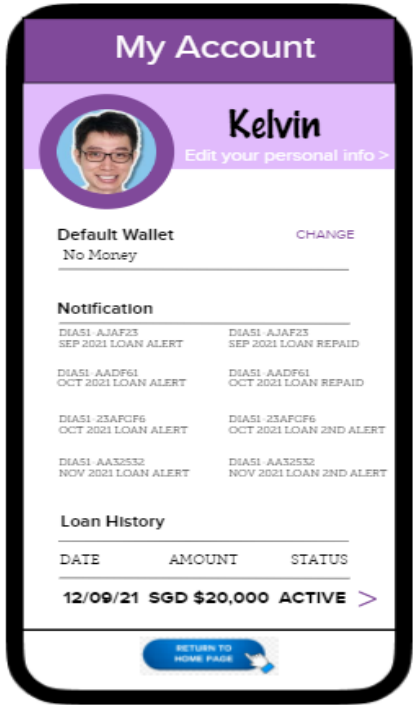


2
Singpass Sign-in

3
Main Menu



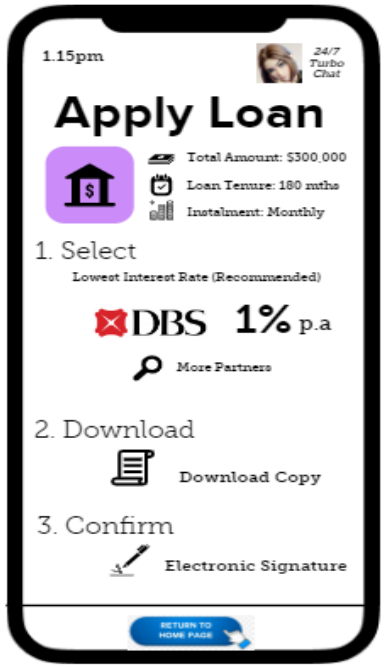
4
Profile



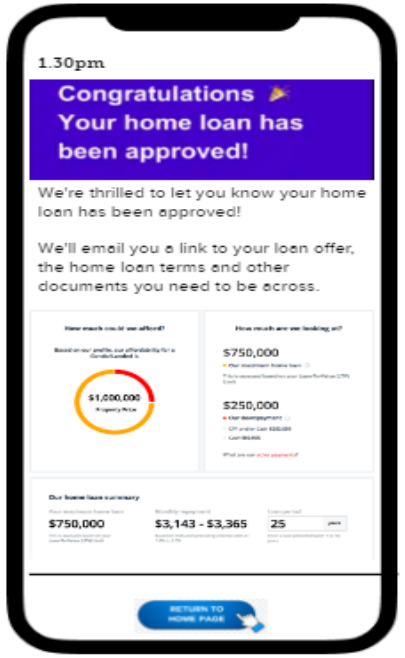
Model Deployment (Master Prototype – Iteration 2)

5
Loan application

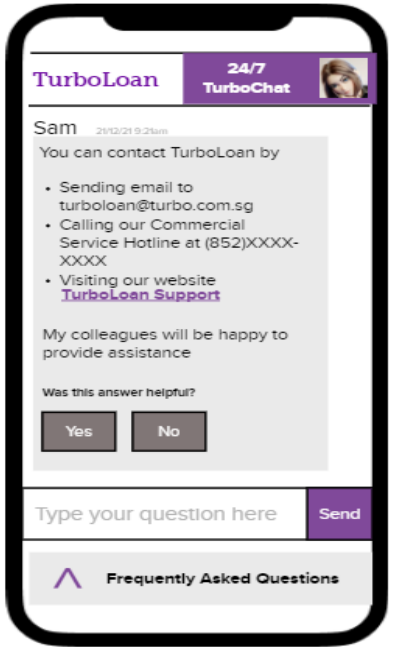
6
Loan application



7
Loan Approval



8
AI assistance



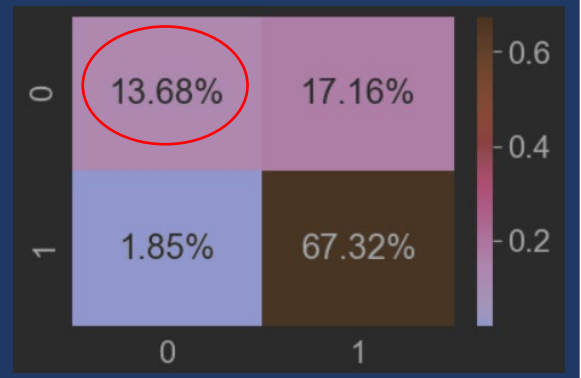
Reality Check 1

Environment Feedback(Reality Check)

Observed	Predicted		
	1	0	Percent correct
1	41	2	95.3%
0	10	9	47.4%
Percent correct	80.4%	81.8%	80.6%

Problem

our True Negatives in every Confusion matrix are too low that affects our prediction on the number of unapproved loans

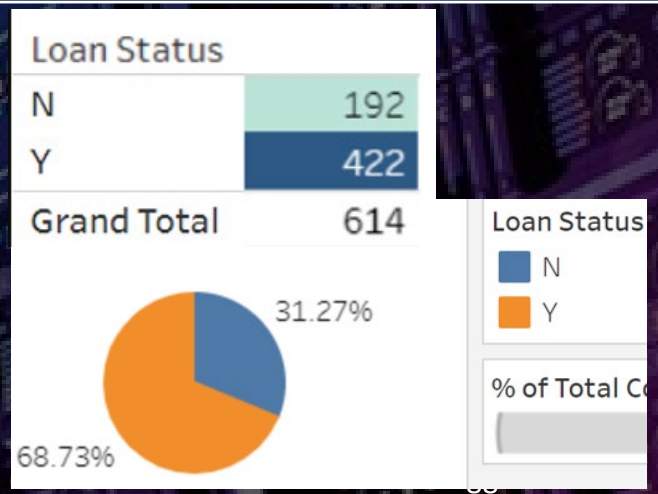


Solution

Need to have a balanced training dataset such that the dependent variable is equally half of each True and False Boolean of the Binary Classification

Why?

We have skewed training dataset that the number of approved loan is roughly 70% comparing to unapproved loan which is only 30% of the whole dataset.



Reality Check 2

Problem 1

Results are shown vastly different in terms of probability and results on different model

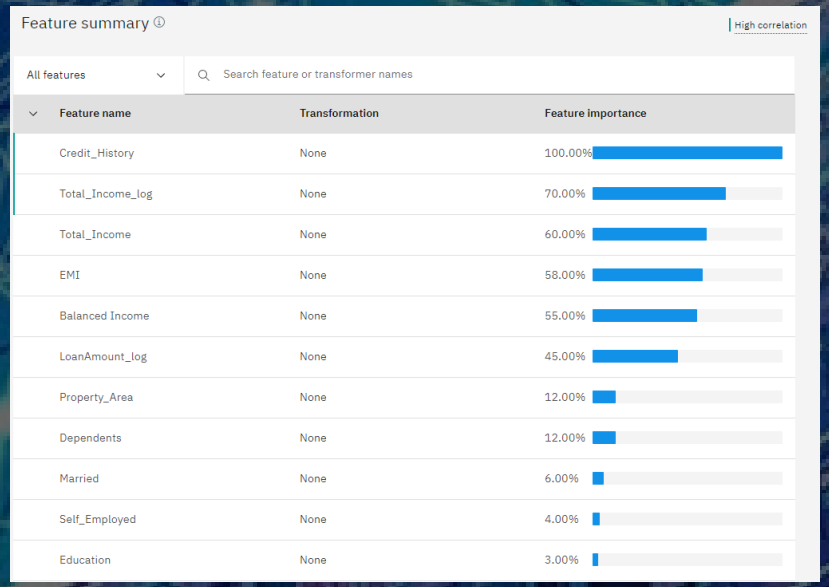
	AutoAI Probability on Loan_Status(%)	
	0	1
1	50.04	49.96
2	15	85
3	39	61
4	89	11
5	19	81
6	94	6

	AutoGluon Probability on Loan_Status(%)	
	0	1
1	9	91
2	74	26
3	12	88
4	86	14
5	74	26
6	83	17

Explanation

Different models has different underlying algorithms and weightage in the Feature Importance, hence the different results

Snap Logistic Regression



Random ForestGini

```
predictor.feature_importance(test_data, model='RandomForestGini_BAG_L1')
Computing feature importance via permutation shuffling for 12 features using 12
        5.11s   = Expected runtime (1.7s per shuffle set)
        0.91s   = Actual runtime (Completed 3 of 3 shuffle sets)
```

	importance	stddev	p_value	n	p99_high	p99_low
Credit_History	0.283171	0.014282	0.000423	3	0.365008	0.201333
LoanAmount_log	0.028897	0.016113	0.044942	3	0.121226	-0.063431
Balanced Income	0.019689	0.014163	0.068865	3	0.100844	-0.061466
Total_Income	0.013075	0.016793	0.154947	3	0.109302	-0.083152
Married	0.010328	0.009279	0.096851	3	0.063499	-0.042843
Property_Area	0.008445	0.011050	0.158306	3	0.071761	-0.054870
Self_Employed	0.007580	0.002291	0.014563	3	0.020708	-0.005548
EMI	0.007326	0.016689	0.263230	3	0.102953	-0.088301
Total_Income_log	0.004375	0.024077	0.391379	3	0.142340	-0.133590
Gender	0.000102	0.008760	0.492888	3	0.050298	-0.050095
Education	-0.001272	0.014058	0.555065	3	0.079284	-0.081828
Dependents	-0.009208	0.005497	0.949448	3	0.022292	-0.040709

Resulted Loan_Status
0
1
1
0
1
0

Problem 2

AutoAI is showing inaccurate results when comparing to the true results of Training dataset

Explanation

1) The imbalanced number of Loan_Status that was mentioned just now contributed to the inaccuracy of the dataset
2) Small records size of training dataset (614 records) contributed as well

Solution

Getting more data helps to train the model to be better in prediction

Reality Check 3

Variable	Description	Data Type
Loan_ID	Unique Loan ID identifier	String
Gender	Male/Female	String
Married	Applicant married (Y/N)	String
Dependents	Number of dependents	String
Education	Applicant Education (Graduate/Under Graduate)	String
Self_Employed	Self employed (Y/N)	String
ApplicantIncome	Applicant's Income	Integer
CoapplicantIncome	Co-Applicant's Income	Float
LoanAmount	Loan amount in thousands	Float
Loan_Amount_Term	Term of loan in months	Float
Credit_History	credit history meets guidelines	Float
Property_Area	Urban/Semi Urban/Rural	String
Loan_Status	Loan approved (Y/N)	String

Problem

The dataset does not define from our source (Github) does not define what type of loan this is.

Solution

Based on the given dataset, we assumed that it is a Housing loan based on

1) Loan_Amount and Loan Amount Term

Lowest Loan = 9k with 1 year repayment

Highest Loan = 700k with 40 years repayment

2) Property_Area

They included property area in the dataset

```
train.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

8 rows × 5 columns Open in new tab

Reality Check 4

Environment Feedback(Reality Check)

	data
Loan_ID	object
Gender	object
Married	object
Dependents	object
Education	object
Self_Employed	object
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	object
Loan_Status	object

Length: 13, dtype: object

Problem

In every bank loan, there should be an interest rate which shows the profit earned by the bank.
However, this is not given in our dataset

Solution

We just have to assume there is a constant interest rate that we can take into consideration

Summary and Reflections

CONSTRAINTS

- Some libraries do not work on IDE software.
*IDE is software that consolidates the basic tools needed for testing and writing

- Limited Machine Learning usage allowance while using IBM Watson such that we need to create multiple accounts for experimenting purpose

CHALLENGES

- Efforts and hours spent when trying to debug various programming codes

- Having to search through the functions of the libraries and how to apply

- Learning new languages/programs such as Python, Jupyter Notebook, Tableau, AutoGluon etc

- For Sci-kit Learn, it does not have the luxury of automation. We have to learn and split the dataset ourselves, applying the suitable cross validation method, choosing the right model based on results.

Conclusion

- **Planning Analytics** – We have identified the business problem in housing loan application

- **Descriptive Analytics –** Using tools like Autogluon and Python, we did data preparation, data exploration and data cleansing.

- **Diagnostic Analytics –** With Data Visualization, we are able to prove if our hypothesis is correct.

- **Predictive Analytics -** By using different models to analyze our dataset to make prediction for our loan approval

- **Prescriptive Analytics - D**ecision-making and come out with an optimized solution for our loan approval

Future Enhancement

Implementing GridSearchCV technique (Sklearn Library) that helps in optimizing the model (hyperparameter tuning) by determining the optimal values for a given model. The performance of a model in Python significantly depends on the value of the hyperparameters.

Using Python Flask to create our backend API to fit the model into various applications. Using API calls such as “get_prediction” method for communicating through the internet in JSON format.

Implement chatbot in our application improve customers engagement process and operational efficiency.

The Journey Takeaways

- We learn how to use different type of libraries such as Pandas, Matlabplot, Seaborn and Tableau to deal with large volume of data in order to find different types of unseen patterns through data Visualisation and derive meaningful information on our data sets.

- We make use of the 5 data analytics cycles which consists of Planning analytics , Descriptive analytics, Diagnostic analytics , Predictive analytics and Prescriptive analytics to make decision support and seek continuous improvement for our business values

- We learn that data preparation is very important as a stepping stone to ensure accuracy in our data. With careful and clean data preparation, it will lead us to more accurate and meaningful insights and thus better outcomes

- We make use of hypothesis testing to test assumptions and assess the plausibility of a hypothesis by using sample data from different sources. Hence we make use of different types of analysis such as Univariate Analysis and Bivariate Analysis to prove if the hypothesis is true.

- We learn that data visualization curates data into forms like density plot, heat map, histogram and stacked bar graph so that it will be easier for us to identify trends and patterns in our data set . This enables us to prove our hypothesis and aid us in making decisions.

- We learn that data engineers, data analysts and data scientists have different roles, skills-sets, and tools. For instance, data scientists take projects from end to end. They create predictive models and present business insights. Data engineers use computer science to process large datasets, focusing on coding, cleaning up datasets and processing requests from data scientists while data analysts typically help stakeholders in the company understand queries with charts. Together, as a team, they are a valuable asset for business planning and development.


THANK
YOU