Loan prediction using machine learning

At IC Solutions

Submitted by:

Name	USN	email
Karthik H K	1KI18EC019	karthik.barati.777@gmail.com
Devraj H U	1KI18EC013	Praveendev900@gmail.com
Santosh K K	1KI18EC037	kksanthosh63@gmail.com

Instructor: Abhishek C

Acknowledgement

Being a part of this internship was a really good experience. I would like to thank the coordinators of IC solutions to participate and learn new things in the internship.

I also would like to thank my instructor, Abhishek C, who taught all the concepts very clearly and also the way to use cutting edge machine learning technologies.

Finally, I would like to thank my parents who supported me to get into this internship.

Abstract

Loans are the core business of banks. The main profit comes directly from the loan's interest. So, the loan companies sanction their loans after intense verification of the applicants. But, they still don't have assurance that the applicant is able to repay the loan or not. Also, the verification process and the decision process to sanction a loan to an applicant is very intense and time consuming to say the least.

The objective of this project is to build a loan amount sanction predictor using machine learning which will predict what amount of loan should be given to an applicant, if any. This prediction is based on the data collected about the applicant that is used to make a prediction.

About the company

ICS is a digital service provider that aims to provide software, designing and marketing solutions to individuals and businesses.

At ICS, we believe that service and quality is the key to success. We provide all kinds of technological and designing solutions from Billing Software to Web Designs or any custom demand that you may have. Experience the service like none other!

Some of our services include: Development - We develop responsive, functional and super-fast websites. We keep User Experience in mind while creating websites.
A website should load quickly and should be accessible even on a small view-port and slow internet connection.

Mobile Application - We offer a wide range of professional android, iOS & Hybrid app development services for our global clients, from a start up to a large enterprise. Design - We offer professional Graphic design, Brochure design & Logo design. We are experts in crafting visual content to convey the right message to the customers.
Consultancy - We are here to provide you with expert advice on your design and development requirement.
Videos - We create a polished professional video that impresses your audience

rediction.

Index

Introduction

This is an internship that is related to machine learning and artificial intelligence. The topics that are covered in the internship are pandas, matplotlib, seaborn etc. for data preprocessing and scikit learn for the machine learning algorithms.

We learnt how to clean the data and preprocess the data using pandas and to visualize the data using matplotlib and seaborn to understand the patterns and anomalies in the data. We used these visualizations and findings to decide on the task to be done. After that, the machine learning models are trained and tested from the processed data that we prepared. Then the models will be evaluated on various parameters and the best model will be selected and deployed.

Problem statement and objective

Loans are the core business of banks. The main profit comes directly from the loan's interest. So, the loan companies sanction their loans after intense verification of the applicants. But, they still don't have assurance that the applicant is able to repay the loan or not. Also, the verification process and the decision process to sanction a loan to an applicant is very intense and time consuming to say the least.

The objective of this project is to build a loan amount sanction predictor using machine learning which will predict what amount of loan should be given to an applicant, if any. This prediction is based on the data collected about the applicant that is used to make a prediction.

Requirement specifications

The requirements needed to do this project are given below.

Hardware requirements:

- Intel core i5 CPU.
- Nvidia Geforce GTX 1650 GPU.
- 8GB RAM.
- 1TB SSD.

Software requirements:

- Python 3.9.0.
- Visual Studio Code.
- Jupyter notebook.
- Pandas
- Seaborn.
- Numpy.
- Matplotlib.
- Scikit learn.

Exploratory data analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns in the data, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

This is a very important step in our model building because we can identify which data is the most influential on the output data. We also can determine unnecessary data, we can clean the data in later parts of the process.

To do the EDA part, we are using **Pandas** library and to visualize the data, we are using **Matplotlib** and **Seaborn** libraries.

Here, we start by importing all the necessary libraries required to do the EDA. We then import our dataset and store it in a variable.

```
[12]
# importing all the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
Python
```

```
[13]
# imporing the dataset
raw_data = pd.read_csv('rawData.csv')
Python
```

```
[14]
raw_data
Python
```

If we call the variable like shown above, we get the following output.

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	...	Credit Score	No. of Defaults	Has Active Credit Card	Property ID	Property Age	Property Type	Property Location	Co-Applicant	Property Price	Loan Sanction Amount (USD)
0	C-36995	Frederica Shealy	F	56	1933.05	Low	Working	Sales staff	Semi-Urban	72809.58	...	809.44	0	NaN	746	1933.05	4	Rural	1	119933.46	54607.18
1	C-33999	America Calderone	M	32	4952.91	Low	Working	NaN	Semi-Urban	46837.47	...	780.40	0	Unpossessed	608	4952.91	2	Rural	1	54791.00	37469.98
2	C-3770	Rosetta Verne	F	65	988.19	High	Pensioner	NaN	Semi-Urban	45593.04	...	833.15	0	Unpossessed	546	988.19	2	Urban	0	72440.58	36474.43
3	C-26480	Zoe Chitty	F	65	NaN	High	Pensioner	NaN	Rural	80057.92	...	832.70	1	Unpossessed	890	NaN	2	Semi-Urban	1	121441.51	56040.54
4	C-23459	Afton Venema	F	31	2614.77	Low	Working	High skill tech staff	Semi-Urban	113858.89	...	745.55	1	Active	715	2614.77	4	Semi-Urban	1	208567.91	74008.28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	C-43723	Angelyn Clevenger	M	38	4969.41	Low	Commercial associate	Managers	Urban	76657.90	...	869.61	0	Unpossessed	566	4969.41	4	Urban	1	111096.56	68992.11
29996	C-32511	Silas Slaugh	M	20	1606.88	Low	Working	Laborers	Semi-Urban	66595.14	...	729.41	0	Inactive	175	1606.88	3	Urban	1	73453.94	46616.60
29997	C-5192	Carmelo Lone	F	49	NaN	Low	Working	Sales staff	Urban	81410.08	...	NaN	0	Active	959	NaN	1	Rural	1	102108.02	61057.56
29998	C-12172	Carolann Osby	M	38	2417.71	Low	Working	Security staff	Semi-Urban	142524.10	...	677.27	1	Unpossessed	375	2417.71	4	Urban	1	168194.47	99766.87
29999	C-33003	Bridget Garibaldi	F	63	3068.24	High	Pensioner	NaN	Rural	156290.54	...	815.44	0	Active	344	3068.24	3	Rural	1	194512.60	117217.90

30000 rows × 24 columns

Here, we can see that the dataset consists of 30000 rows and 24 columns.
It also consists of many columns such as Customer ID, Income, Profession etc.
We will analyze this data and perform operations on the data to further explore it and clean it such that it is good enough to be used to train a machine learning model.

```
[31]  # description of the data [numerical columns]
      data.describe()
                                                                    Python
```

	Age	Income (USD)	Loan Amount Request (USD)	Current Loan Expenses (USD)	Dependents	Credit Score	No. of Defaults	Property ID	Property Age	Property Type	Co-Applicant	Property Price	Loan Sanction Amount (USD)
count	30000.000000	2.542400e+04	30000.000000	29828.000000	27507.000000	28297.000000	30000.000000	30000.000000	2.515000e+04	30000.000000	30000.000000	3.000000e+04	29660.000000
mean	40.092300	2.630574e+03	88826.333855	400.936876	2.253027	739.885381	0.193933	501.934700	2.631119e+03	2.460067	-4.743867	1.317597e+05	47649.342208
std	16.045129	1.126272e+04	59536.949605	242.545375	0.951162	72.163846	0.395384	288.158086	1.132268e+04	1.118562	74.614593	9.354955e+04	48221.146686
min	18.000000	3.777000e+02	6048.240000	-999.000000	1.000000	580.000000	0.000000	1.000000	3.777000e+02	1.000000	-999.000000	-9.990000e+02	-999.000000
25%	25.000000	1.650457e+03	41177.755000	247.667500	2.000000	681.880000	0.000000	251.000000	1.650450e+03	1.000000	1.000000	6.057216e+04	0.000000
50%	40.000000	2.222435e+03	75128.075000	375.205000	2.000000	739.820000	0.000000	504.000000	2.223250e+03	2.000000	1.000000	1.099936e+05	35209.395000
75%	55.000000	3.090593e+03	119964.605000	521.292500	3.000000	799.120000	0.000000	751.000000	3.091408e+03	3.000000	1.000000	1.788807e+05	74261.250000
max	65.000000	1.777460e+06	621497.820000	3840.880000	14.000000	896.260000	1.000000	999.000000	1.777460e+06	4.000000	1.000000	1.077967e+06	481907.320000

Description of the data shows us the description of the numerical columns of the dataset. We can see the count, mean, 25%, max, etc.

From the above result, we can conclude the following:

- 4 columns have a negative min value, which have to be cleaned.
- 25% of the rows in the Loan Sanction Amount (USD) column have a value of 0, which needs to be properly handled.
- Over 75% of the applicants have an age of about 55 years, 50% of them are about 40 years, and about 25% of them are about 25 years of age.
  The applicants have dependents ranging from about 1 to 14 people.

```
[15]  raw_data.info()                                                    Python
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Customer ID                   30000 non-null  object
 1   Name                          30000 non-null  object
 2   Gender                        29947 non-null  object
 3   Age                           30000 non-null  int64
 4   Income (USD)                  25424 non-null  float64
 5   Income Stability              28317 non-null  object
 6   Profession                    30000 non-null  object
 7   Type of Employment            22730 non-null  object
 8   Location                      30000 non-null  object
 9   Loan Amount Request (USD)     30000 non-null  float64
 10  Current Loan Expenses (USD)   29828 non-null  float64
 11  Expense Type 1                30000 non-null  object
 12  Expense Type 2                30000 non-null  object
 13  Dependents                    27507 non-null  float64
 14  Credit Score                  28297 non-null  float64
 15  No. of Defaults               30000 non-null  int64
 16  Has Active Credit Card        28434 non-null  object
 17  Property ID                   30000 non-null  int64
 18  Property Age                  25150 non-null  float64
 19  Property Type                 30000 non-null  int64
 20  Property Location             29644 non-null  object
 21  Co-Applicant                  30000 non-null  int64
 22  Property Price                30000 non-null  float64
 23  Loan Sanction Amount (USD)    29660 non-null  float64
dtypes: float64(8), int64(5), object(11)
memory usage: 5.5+ MB
```

If we call data.info(), we get the following result showing all the columns in our dataset, the number of non-null values, and the dtype of the columns.

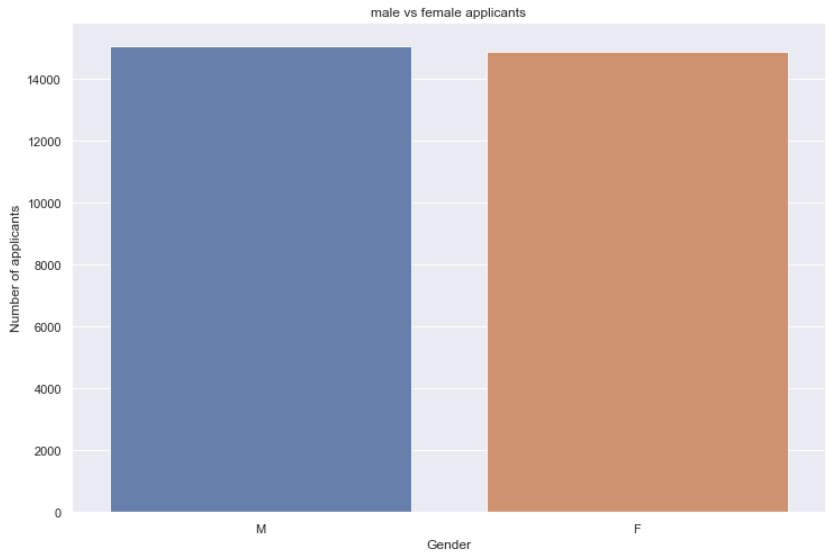
We also get a count of each dtype present in the dataset.

Male vs female applicants

```
gender = raw_data.Gender.value_counts()

plt.figure(figsize=(12,8))
sns.set(style="darkgrid")
sns.barplot(x=gender.index, y=gender.values, alpha=0.9)
plt.title('male vs female applicants')
plt.ylabel('Number of applicants', fontsize=12)
plt.xlabel('Gender', fontsize=12)
plt.show()
```

[16] Python



Plotting a barplot of male vs female applicants shows that male applicants constitute of just above 50% and female applicants constitute just below 50% of all the applicants.

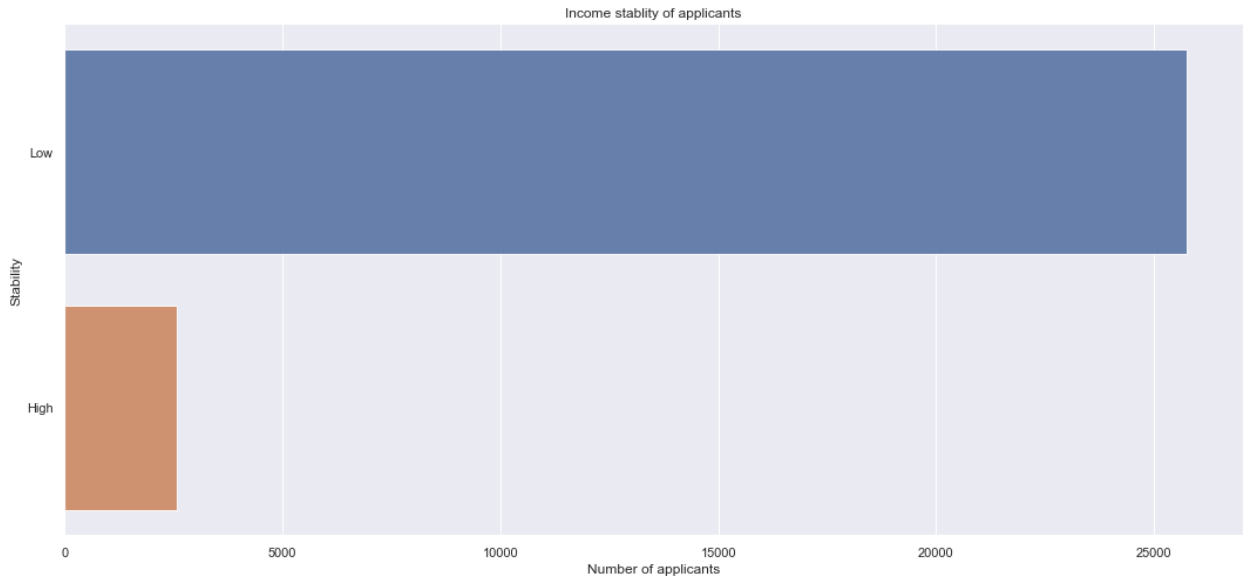
Income Stability

```
inc_stab = raw_data['Income Stability'].value_counts()

plt.figure(figsize=(18, 8))

sns.set(style='darkgrid')
sns.barplot(y=inc_stab.index, x=inc_stab.values, alpha=0.9)
plt.title('Income stablity of applicants')
plt.ylabel('Stability', fontsize=12)
plt.xlabel('Number of applicants', fontsize=12)
plt.show()
```

[17] Python



Plotting a barplot of the income stability shows that the majority of the applicants have a low income stability whereas only a handful of them have a high income stability.

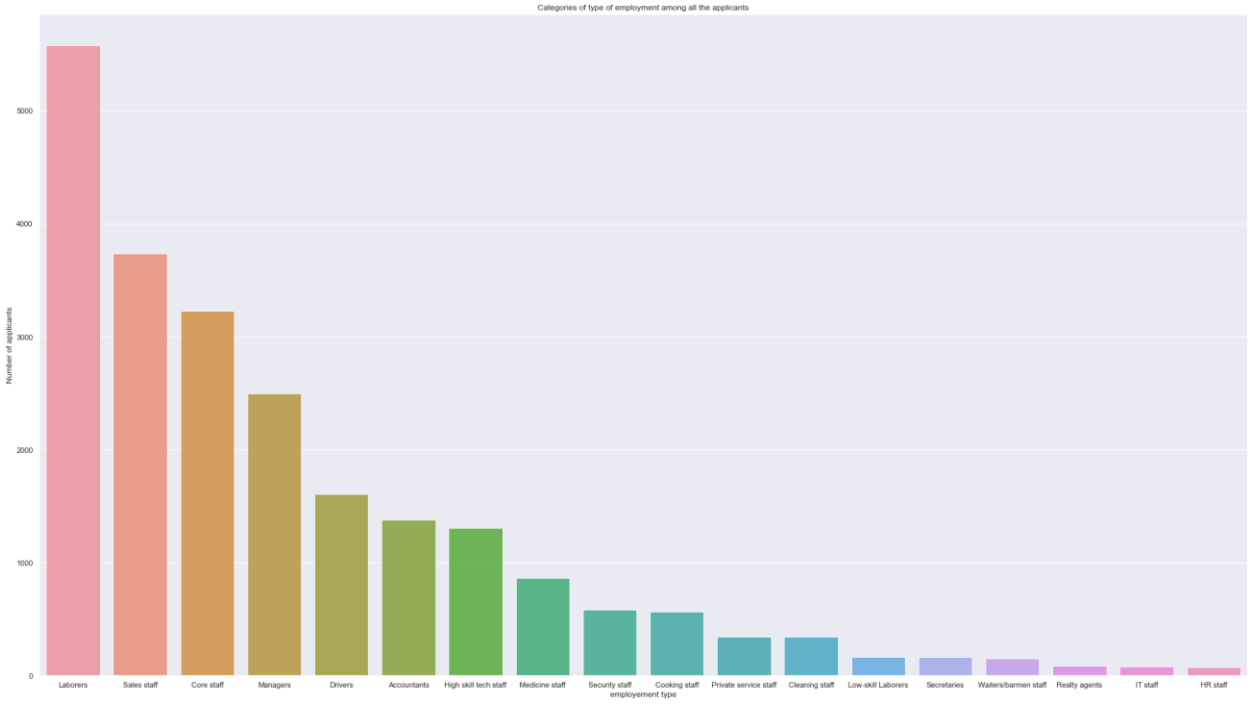
Categories of type of employment of the applicants

```
emp_type = raw_data['Type of Employment'].value_counts()

plt.figure(figsize=(32, 18))

sns.set(style='darkgrid')
sns.barplot(x=emp_type.index, y=emp_type.values, alpha=0.9)
plt.title('Categories of type of employment among all the applicants')
plt.xlabel('employement type', fontsize=12)
plt.ylabel('Number of applicants', fontsize=12)
plt.show()
```

[18] Python



Plotting the graph of the employment type of the applicants, we can conclude that the majority of the applicants are laborers, sales staff, core staff and managers. This explains the low income stability of the applicants.

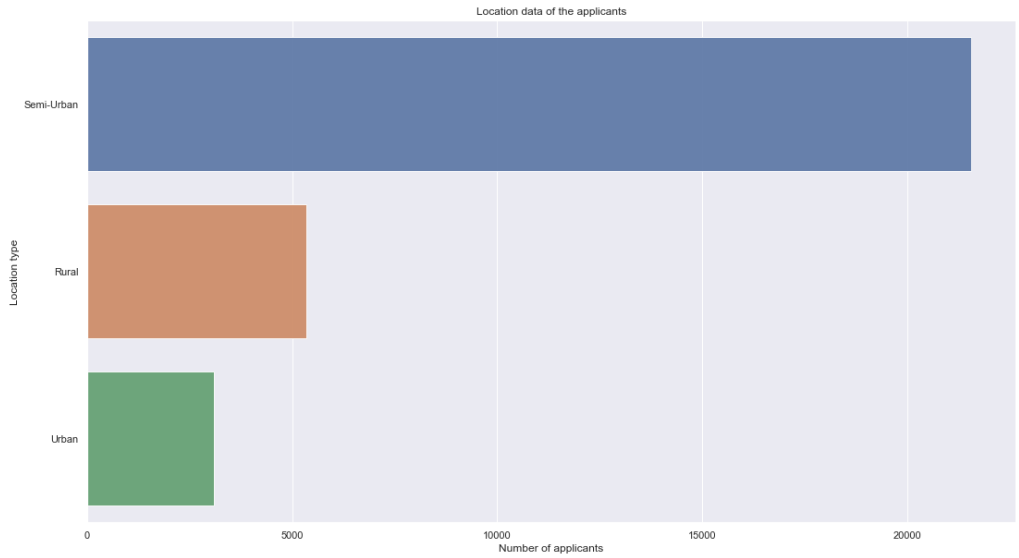
Location data of the applicants

```
loc_data = raw_data['Location'].value_counts()

plt.figure(figsize=(18, 10))

sns.set(style='darkgrid')
sns.barplot(y=loc_data.index, x=loc_data.values, alpha=0.9)
plt.title('Location data of the applicants')
plt.ylabel('Location type', fontsize=12)
plt.xlabel('Number of applicants', fontsize=12)
plt.show()
```

[19] Python



Plotting a graph of the location data given, we can see that people from a semi urban area applied the most, while people from urban areas applied the least for a loan.

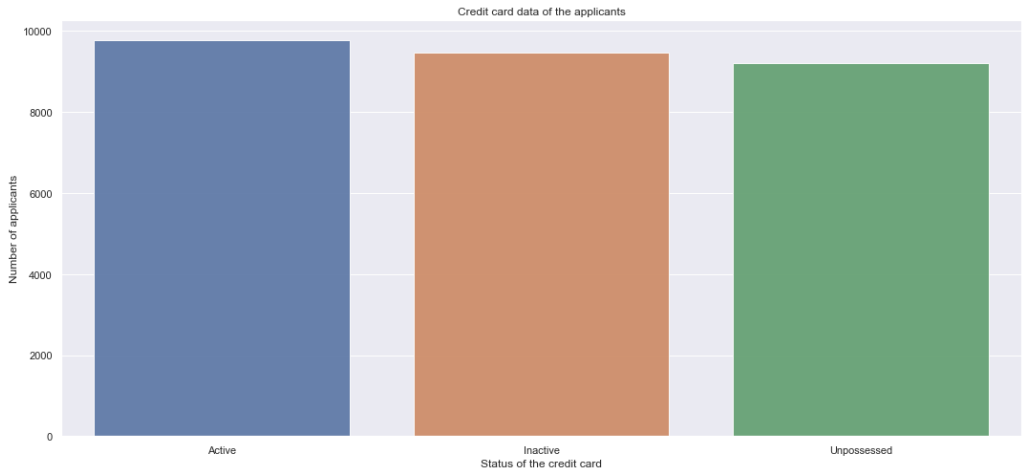
Credit card data

```
cre_card = raw_data['Has Active Credit Card'].value_counts()

plt.figure(figsize=(18, 8))

sns.set(style='darkgrid')
sns.barplot(x=cre_card.index, y=cre_card.values, alpha=0.9)
plt.title('Credit card data of the applicants')
plt.xlabel('Status of the credit card', fontsize=12)
plt.ylabel('Number of applicants', fontsize=12)
plt.show()
```

[20] Python



From the graph above, we can see that the number of applicants having an active credit card, the ones who have an inactive card and the ones who don't have a credit card are almost equal, but the number of credit card holding applicants is slightly higher than the other two categories.

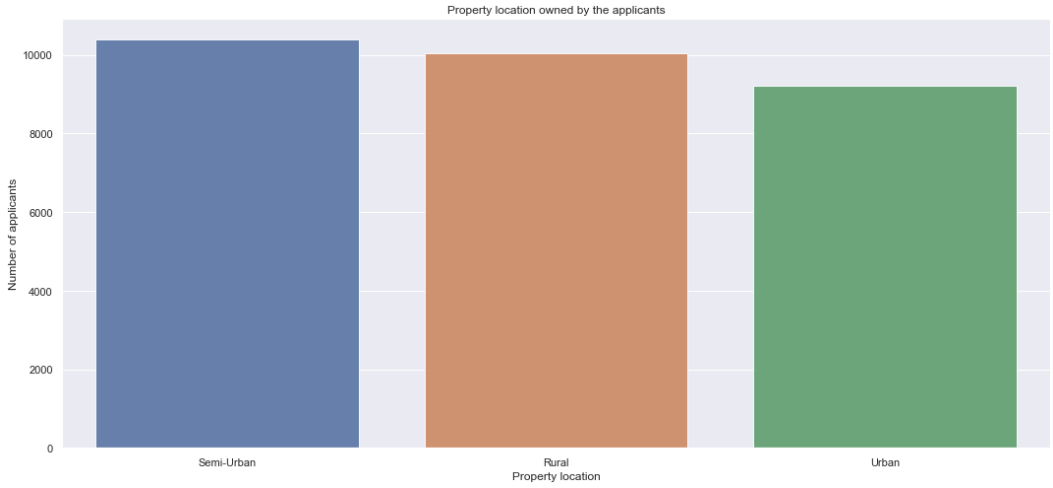
Property location owned by the applicants

```
prop_loc = raw_data['Property Location'].value_counts()

plt.figure(figsize=(18, 8))

sns.set(style='darkgrid')
sns.barplot(x=prop_loc.index, y=prop_loc.values, alpha=0.9)
plt.title('Property location owned by the applicants')
plt.xlabel('Property location', fontsize=12)
plt.ylabel('Number of applicants', fontsize=12)
plt.show()
```

[21] Python



From the above plot, we can conclude that the number of applicants having a property in a semi-urban area is a little higher than the ones having a property in a rural area. The applicants having a property in an urban area are the least among the three.

Credit score vs Loan Amount Request (USD)

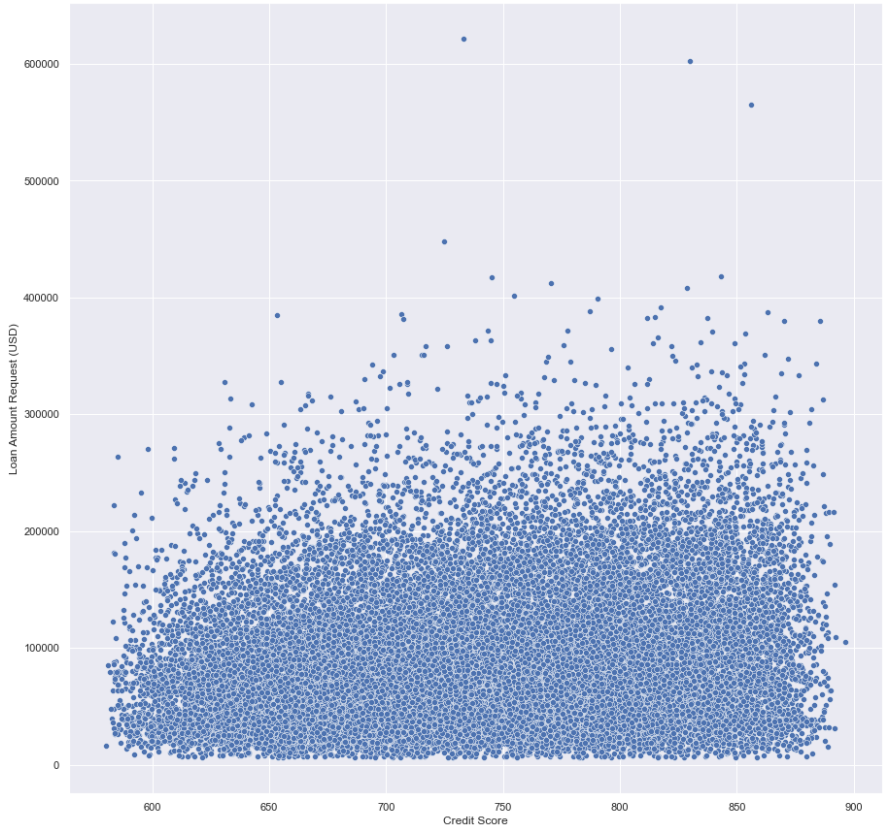
```
plt.figure(figsize=(15, 15))

sns.scatterplot(x='Credit Score', y='Loan Amount Request (USD)', data=raw_data)
```

[22] Python

Plotting a scatterplot of the credit score vs loan amount request in USD shows us the following:

- Over 95% of all the applicants requested a loan amount less than 200000USD.



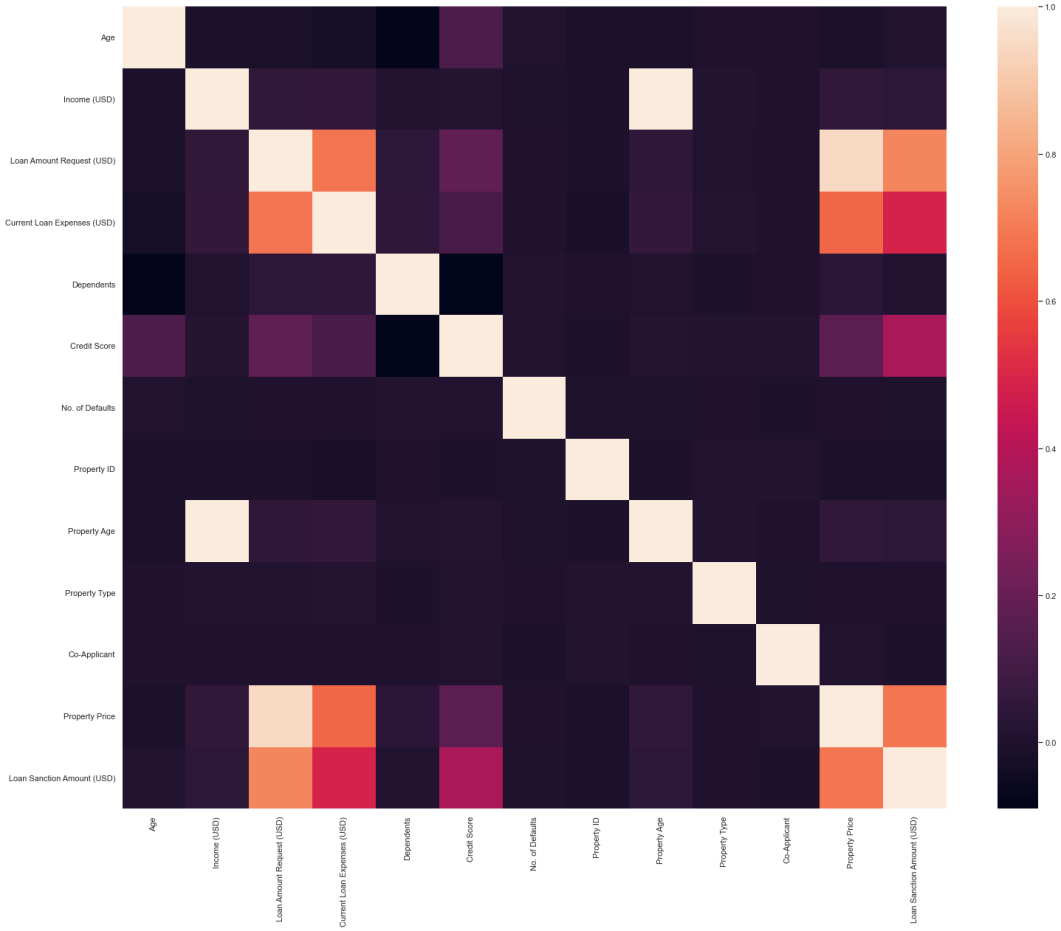
- The remaining applicants requested for more than 200000USD.
- There are a few anomalies such as one person requesting a loan of over 600000USD.

Heatmap of all the data

```
corr = raw_data.corr()


plt.figure(figsize=(25, 20))
sns.heatmap(corr)
```

[23] Python



From the above heatmap, we can observe that, the factors such as Loan Amount Request, Current Loan Expenses and Property price matter the most and also Credit score has an impact on the Loan Sanction Amount.

Preparing the machine learning model

In this section, we will prepare the machine learning model. The steps required are:

- Data cleaning / Data preprocessing.
- Identifying the type of task and choosing the proper machine model.
- Training the model with the prepared data.
- Testing the trained model.

Data cleaning/Data preprocessing

We start by cleaning the data by the following steps:

- Identifying the data that needs to be dropped.
- Identifying and handling null values.
- Identifying and handling negative values and inconsistency in the data.
- Encoding the categorical data.

This is a very important step in the process because, without cleaning and preparing the data properly, training a machine learning model will result in the model not giving accurate predictions. The model may not be even trained in some instances because the data is of a different data type other than the one that the model accepts.

```
[30]
# importing the dataset
data = pd.read_csv('rawData.csv')
data.head()
```
Python

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	...	Credit Score	No. of Defaults	Has Active Credit Card	Property ID	Property Age	Property Type	Property Location	Co-Applicant	Property Price	Loan Sanction Amount (USD)
0	C-36995	Frederica Shealy	F	56	1933.05	Low	Working	Sales staff	Semi-Urban	72809.58	...	809.44	0	NaN	746	1933.05	4	Rural	1	119933.46	54607.18
1	C-33999	America Calderone	M	32	4952.91	Low	Working	NaN	Semi-Urban	46837.47	...	780.40	0	Unpossessed	608	4952.91	2	Rural	1	54791.00	37469.98
2	C-3770	Rosetta Verne	F	65	988.19	High	Pensioner	NaN	Semi-Urban	45593.04	...	833.15	0	Unpossessed	546	988.19	2	Urban	0	72440.58	36474.43
3	C-26480	Zoe Chitty	F	65	NaN	High	Pensioner	NaN	Rural	80057.92	...	832.70	1	Unpossessed	890	NaN	2	Semi-Urban	1	121441.51	56040.54
4	C-23459	Afton Venema	F	31	2614.77	Low	Working	High skill tech staff	Semi-Urban	113858.89	...	745.55	1	Active	715	2614.77	4	Semi-Urban	1	208567.91	74008.28

5 rows × 24 columns

The above code displays the first 5 rows of the dataset. As we can see, there are 24 columns in the dataset, many of them are not necessary and need to be dropped and many of them are categorical and need to be encoded.

Unique data such as names, IDs etc. are not necessary to train the machine learning model and thus need to be dropped from the dataset. Categorical data such as Income Stability, Type of employment etc. has to be encoded into a numerical data type so that it can be fed into the machine learning model.

```
[31]
# description of the data [numerical columns]
data.describe()
```
Python

	Age	Income (USD)	Loan Amount Request (USD)	Current Loan Expenses (USD)	Dependents	Credit Score	No. of Defaults	Property ID	Property Age	Property Type	Co-Applicant	Property Price	Loan Sanction Amount (USD)
count	30000.000000	2.542400e+04	30000.000000	29828.000000	27507.000000	28297.000000	30000.000000	30000.000000	2.515000e+04	30000.000000	30000.000000	3.000000e+04	29660.000000
mean	40.092300	2.630574e+03	88826.333855	400.936876	2.253027	739.885381	0.193933	501.934700	2.631119e+03	2.460067	-4.743867	1.317597e+05	47649.342208
std	16.045129	1.126272e+04	59536.949605	242.545375	0.951162	72.163846	0.395384	288.158086	1.132268e+04	1.118562	74.614593	9.354955e+04	48221.146686
min	18.000000	3.777000e+02	6048.240000	-999.000000	1.000000	580.000000	0.000000	1.000000	3.777000e+02	1.000000	-999.000000	-9.990000e+02	-999.000000
25%	25.000000	1.650457e+03	41177.755000	247.667500	2.000000	681.880000	0.000000	251.000000	1.650450e+03	1.000000	1.000000	6.057216e+04	0.000000
50%	40.000000	2.222435e+03	75128.075000	375.205000	2.000000	739.820000	0.000000	504.000000	2.223250e+03	2.000000	1.000000	1.099936e+05	35209.395000
75%	55.000000	3.090593e+03	119964.605000	521.292500	3.000000	799.120000	0.000000	751.000000	3.091408e+03	3.000000	1.000000	1.788807e+05	74261.250000
max	65.000000	1.777460e+06	621497.820000	3840.880000	14.000000	896.260000	1.000000	999.000000	1.777460e+06	4.000000	1.000000	1.077967e+06	481907.320000

When we call data.describe(), it gives a description of all the numerical columns of the dataset. From this, we can see that the columns Current loan expenses, Co-applicant, Property price and Loan Sanction Amount have a negative value as minimum. This does not correlate to the real world where a loan sanction amount is a positive value, not a negative one. So, these columns must be handled properly.

The next step is to call data.info() to get all the columns, their data types so that we can decide which columns have unique values like IDs etc. so that we can drop them from our dataset.

```
[32]  data.info()
                                                                    Python
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Customer ID                   30000 non-null  object
 1   Name                          30000 non-null  object
 2   Gender                        29947 non-null  object
 3   Age                           30000 non-null  int64
 4   Income (USD)                  25424 non-null  float64
 5   Income Stability              28317 non-null  object
 6   Profession                    30000 non-null  object
 7   Type of Employment            22730 non-null  object
 8   Location                      30000 non-null  object
 9   Loan Amount Request (USD)     30000 non-null  float64
 10  Current Loan Expenses (USD)   29828 non-null  float64
 11  Expense Type 1                30000 non-null  object
 12  Expense Type 2                30000 non-null  object
 13  Dependents                    27507 non-null  float64
 14  Credit Score                  28297 non-null  float64
 15  No. of Defaults               30000 non-null  int64
 16  Has Active Credit Card        28434 non-null  object
 17  Property ID                   30000 non-null  int64
 18  Property Age                  25150 non-null  float64
 19  Property Type                 30000 non-null  int64
 20  Property Location             29644 non-null  object
 21  Co-Applicant                  30000 non-null  int64
 22  Property Price                30000 non-null  float64
 23  Loan Sanction Amount (USD)    29660 non-null  float64
dtypes: float64(8), int64(5), object(11)
memory usage: 5.5+ MB
```

Calling data.info() shows that there are many columns consisting of unique values that need to be dropped. It also tells us that there are 8 columns of float64 data type, 5 columns of int64 data type.

Columns such as Customer ID, name, Profession, Property ID, Property type have unique values. So they need to be dropped from the dataset. We are also going to drop the Gender column from the dataset as we are trying to prepare a machine learning model that is not gender biased. Also in the EDA part, we observed that the difference between male and female applicants is miniscule.

```
data.drop(columns=['Customer ID', 'Name', 'Gender', 'Profession', 'Property ID', 'Property Type', 'Property Age'], inplace=True)
data
```

[33] Python

Executing the above code drops all the columns specified in the columns list. So, we dropped all the unique and unnecessary data from the dataset. inplace=True ensures that the original dataset variable is altered.

	Age	Income (USD)	Income Stability	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	Expense Type 2	Dependents	Credit Score	No. of Defaults	Has Active Credit Card	Property Location	Co-Applicant	Property Price	Loan Sanction Amount (USD)
0	56	1933.05	Low	Sales staff	Semi-Urban	72809.58	241.08	N	N	3.0	809.44	0	NaN	Rural	1	119933.46	54607.18
1	32	4952.91	Low	NaN	Semi-Urban	46837.47	495.81	N	Y	1.0	780.40	0	Unpossessed	Rural	1	54791.00	37469.98
2	65	988.19	High	NaN	Semi-Urban	45593.04	171.95	N	Y	1.0	833.15	0	Unpossessed	Urban	0	72440.58	36474.43
3	65	NaN	High	NaN	Rural	80057.92	298.54	N	Y	2.0	832.70	1	Unpossessed	Semi-Urban	1	121441.51	56040.54
4	31	2614.77	Low	High skill tech staff	Semi-Urban	113858.89	491.41	N	Y	NaN	745.55	1	Active	Semi-Urban	1	208567.91	74008.28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	38	4969.41	Low	Managers	Urban	76657.90	722.34	Y	Y	2.0	869.61	0	Unpossessed	Urban	1	111096.56	68992.11
29996	20	1606.88	Low	Laborers	Semi-Urban	66595.14	253.04	N	N	3.0	729.41	0	Inactive	Urban	1	73453.94	46616.60
29997	49	NaN	Low	Sales staff	Urban	81410.08	583.11	N	Y	NaN	NaN	0	Active	Rural	1	102108.02	61057.56
29998	38	2417.71	Low	Security staff	Semi-Urban	142524.10	378.29	N	Y	3.0	677.27	1	Unpossessed	Urban	1	168194.47	99766.87
29999	63	3068.24	High	NaN	Rural	156290.54	693.94	N	Y	1.0	815.44	0	Active	Rural	1	194512.60	117217.90

30000 rows × 17 columns

The result obtained above shows us that the specified columns have been dropped. The dataset is reduced from 24 columns to 17 columns.

Finding and handling NaN values

```
[34]  data.isnull().sum()
                                                        Python
```

```
Age                                0
Income (USD)                    4576
Income Stability                1683
Type of Employment              7270
Location                           0
Loan Amount Request (USD)          0
Current Loan Expenses (USD)      172
Expense Type 1                     0
Expense Type 2                     0
Dependents                      2493
Credit Score                    1703
No. of Defaults                    0
Has Active Credit Card          1566
Property Location                356
Co-Applicant                       0
Property Price                     0
Loan Sanction Amount (USD)       340
dtype: int64
```

From the result above, we can see that there are null values in many fields and they need to be handled properly. Many of the numerical columns and categorical data have null values which means that they are not available. Proper

handling of NaN/null values is the difference between an accurate model and an inaccurate one, so this step is important.

```
data = data.fillna(method='bfill')

# data = data.dropna()
data.isnull().sum()
```

[35] Python

```
Age                              0
Income (USD)                     0
Income Stability                 0
Type of Employment               1
Location                         0
Loan Amount Request (USD)        0
Current Loan Expenses (USD)      0
Expense Type 1                   0
Expense Type 2                   0
Dependents                       0
Credit Score                     0
No. of Defaults                  0
Has Active Credit Card           0
Property Location                0
Co-Applicant                     0
Property Price                   0
Loan Sanction Amount (USD)       0
dtype: int64
```

We are handling the NaN values by filling them with another value. We can go the route of dropping the rows with NaN values but that will result a huge loss of data as there are many rows with NaN values.

Here, we are fill the data by specifying the method as ‘bfill’, i.e backfill which fills the NaN value with the next valid observation.

After filling all the columns with NaN values with the next valid observations, we get the above result. Notice that ‘type of employment’ column has one NaN value. This is because of the applicants who are pensioners don’t have a type of employment. This will be also encoded by the encoder that we use.

Encoding categorical data columns

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

cols = data.select_dtypes(include=['object'])
cols.head(10)

for col in cols:
    data[col] = le.fit_transform(data[col])

data
```

[36] Python

Encoding is a process of converting the unique values in a column into integer data such that each value in the column corresponds to a integer.

To encode our data we use LabelEncoder from SKLearn. This will encode target labels from values ranging from 0 to n – 1 where n is the number of unique values in the column.

	Age	Income (USD)	Income Stability	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	Expense Type 2	Dependents	Credit Score	No. of Defaults	Has Active Credit Card	Property Location	Co-Applicant	Property Price	Loan Sanction Amount (USD)
0	56	1933.05	1	14	1	72809.58	241.08	0	0	3.0	809.44	0	2	0	1	119933.46	54607.18
1	32	4952.91	1	6	1	46837.47	495.81	0	1	1.0	780.40	0	2	0	1	54791.00	37469.98
2	65	988.19	0	6	1	45593.04	171.95	0	1	1.0	833.15	0	2	2	0	72440.58	36474.43
3	65	2614.77	0	6	0	80057.92	298.54	0	1	2.0	832.70	1	2	1	1	121441.51	56040.54
4	31	2614.77	1	6	1	113858.89	491.41	0	1	2.0	745.55	1	0	1	1	208567.91	74008.28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	38	4969.41	1	10	2	76657.90	722.34	1	1	2.0	869.61	0	2	2	1	111096.56	68992.11
29996	20	1606.88	1	8	1	66595.14	253.04	0	0	3.0	729.41	0	1	2	1	73453.94	46616.60
29997	49	2417.71	1	14	2	81410.08	583.11	0	1	3.0	677.27	0	0	0	1	102108.02	61057.56
29998	38	2417.71	1	16	1	142524.10	378.29	0	1	3.0	677.27	1	2	2	1	168194.47	99766.87
29999	63	3068.24	0	18	0	156290.54	693.94	0	1	1.0	815.44	0	0	0	1	194512.60	117217.90

30000 rows × 17 columns

After encoding operation is done, the above is the result we get. We can see that all the categorical data has been encoded into an integer ranging from 0 to n-1.

Handling negative values

```python
cols_with_negative_vals = ['Current Loan Expenses (USD)', 'Co-Applicant', 'Property Price', 'Loan Sanction Amount (USD)']

for col in cols_with_negative_vals:
    data[col] = np.where(data[col] <= 0, np.nan, data[col])

data['Co-Applicant'] = np.where(data['Co-Applicant'] <= 0, data['Co-Applicant'].mean(), data['Co-Applicant'])

# data['Co-Applicant'].min()
data.describe()
data.dropna(how='any', axis=0)

data.isnull().sum()
```

Python

```
Age                              0
Income (USD)                     0
Income Stability                 0
Type of Employment               0
Location                         0
Loan Amount Request (USD)        0
Current Loan Expenses (USD)    179
Expense Type 1                   0
Expense Type 2                   0
Dependents                       0
Credit Score                     0
No. of Defaults                  0
Has Active Credit Card           0
Property Location                0
Co-Applicant                  4484
Property Price                 352
Loan Sanction Amount (USD)    8302
dtype: int64
```

We saw earlier that some columns had negative values that had to be processed. For this dataset and the given scenario, having negative values is not ideal because it does not reflect the real world where those values will never be negative. So we will handle these values by first converting all the negative values into null/NaN values and then dropping them.

After the conversion of negative values to null values has been done, we can see that there are a lot of negative values. Especially we need to look at Loan Sanction Amount column which has 8302 rows with negative values. This data is simply incorrect as there is never a negative loan sanctioned in the real world.

```python
data = data.dropna(how='any', axis=0)
data.isnull().sum()
```

```
Age                             0
Income (USD)                    0
Income Stability                0
Type of Employment              0
Location                        0
Loan Amount Request (USD)       0
Current Loan Expenses (USD)     0
Expense Type 1                  0
Expense Type 2                  0
Dependents                      0
Credit Score                    0
No. of Defaults                 0
Has Active Credit Card          0
Property Location               0
Co-Applicant                    0
Property Price                  0
Loan Sanction Amount (USD)      0
dtype: int64
```

After the conversion of negative values into null values, we will drop all the null values by executing the above code. This will get rid of all the rows with negative values which were converted into null values.

```python
# saving the data to a csv file
data.to_csv('./encoded data/enc1.csv')
```

After all the data cleaning and encoding operation has been done, we will save the processed data into a new csv file. We will call the file as ‘enc1.csv’.

Training the machine learning models

After the data has been processed, we can use that data to train the machine learning models. The steps taken while training a model are:

- Identifying the type of task given i.e classification or regression.
- Choosing the appropriate models.
- Identifying the dependent and independent variables.
- Splitting the data into training and testing data.
- Training the model.
- Using the test data to predict the output.
- Evaluating the trained model's accuracy by comparing with the test output and the predicted output.

Identifying the task given

Here, our task is to predict the amount of loan to be sanctioned to an applicant by the bank. This falls under the category of regression as we are predicting the actual amount to be sanctioned. We can use regression models for our application.

Choosing the appropriate models

Choosing the correct model is a crucial step in the model training and testing because it will decide the accuracy of the trained model. Not all models are the same and each model is built for a specific set of tasks. So, we need to choose the appropriate models for our task to maximize the accuracy of our trained models.

For this application, we will choose to train 3 regression models available in SKLearn:

- Linear regression: Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

- Bayesian Ridge: Bayesian regression allows a natural mechanism to survive insufficient data or poorly distributed data by formulating linear regression using probability distributors rather than point estimates. The output or response ‘y’ is assumed to drawn from a probability distribution rather than estimated as a single value.

- Decision Tree Regressor: Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Identifying the dependent and independent variables

We need to identify the dependent variables and independent variables in our data to train the model. Here, the dependent variable is 'Loan Sanction Amount (USD)'. We can isolate it and store it in a variable.

```python
# defining the dependent variable
Y = data['Loan Sanction Amount (USD)']
Y
```

Python

```
0         54607.18
1         37469.98
2         56040.54
3         74008.28
4         22382.57
           ...
20058     68992.11
20059     46616.60
20060     61057.56
20061     99766.87
20062    117217.90
Name: Loan Sanction Amount (USD), Length: 20063, dtype: float64
```

The independent variables are all the columns except the dependent variable in this case. So, we isolate the dependent variables and store them in a variable.

```python
# defining the independent variables
X = data.drop(['Loan Sanction Amount (USD)', 'Unnamed: 0'], axis=1)
X
```

[24] Python

	Age	Income (USD)	Income Stability	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	Expense Type 2	Dependents	Credit Score	No. of Defaults	Has Active Credit Card	Property Location	Co-Applicant	Property Price
0	56	1933.05	1	14	1	72809.58	241.08	0	0	3.0	809.44	0	2	0	1.0	119933.46
1	32	4952.91	1	6	1	46837.47	495.81	0	1	1.0	780.40	0	2	0	1.0	54791.00
2	65	2614.77	0	6	0	80057.92	298.54	0	1	2.0	832.70	1	2	1	1.0	121441.51
3	31	2614.77	1	6	1	113858.89	491.41	0	1	2.0	745.55	1	0	1	1.0	208567.91
4	60	1234.92	1	15	0	34434.72	181.48	0	0	2.0	684.12	1	1	0	1.0	43146.82
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20058	38	4969.41	1	10	2	76657.90	722.34	1	1	2.0	869.61	0	2	2	1.0	111096.56
20059	20	1606.88	1	8	1	66595.14	253.04	0	0	3.0	729.41	0	1	2	1.0	73453.94
20060	49	2417.71	1	14	2	81410.08	583.11	0	1	3.0	677.27	0	0	0	1.0	102108.02
20061	38	2417.71	1	16	1	142524.10	378.29	0	1	3.0	677.27	1	2	2	1.0	168194.47
20062	63	3068.24	0	18	0	156290.54	693.94	0	1	1.0	815.44	0	0	0	1.0	194512.60

20063 rows × 16 columns

Splitting the data into training and testing

After we decide which models to train, we need to split the data into training and testing data so that most of the data is used to train the model and the remaining data can be used to test the model for it's accuracy.

To split the data, we will use 'train_test_split' from sklearn's model selection. Then we pass the dependent and independent variables to the train_test_split method along with a test size and random_state. Test_size determines the percentage of the data to be used as test data. Random_state controls the shuffling applied to the data before applying the split.

```python
# splitting the dataset into training and testing data
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)

# test size determines the split between training and testing data.
#  here 20% of the data is the testing data.
#  remaining 80% of the data is the training data.
```

Python

Train_test_split returns 4 values: X_train, X_test, Y_train, Y_test. The *_train data is used to train the model and the *_test data is used to test the model after training it.

Training, testing and evaluating the models

After the train and test data is ready, we can move on to training and testing the models. The process of training and testing is as follows:

- The model is imported from the library and instantiated.
- Then it is trained using the training data.
- The trained model is used to predict the output using the test input.
- Then the model is evaluated by analyzing the r2 score by using the predicted output and the test output.

All the 3 selected models are trained, tested and evaluated according the steps mentioned above.

Model 1: Linear Regression

```python
# importing and creating the object of the regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Python

```python
# training the model
model.fit(X_train, Y_train)
```

Python

```
LinearRegression()
```

```python
# testing the model
predictions = model.predict(X_test)
```

Python

```python
# getting the r2 score of the trained model
from sklearn.metrics import r2_score
r2_score(Y_test, predictions)
```

Python

```
0.9693971794711593
```

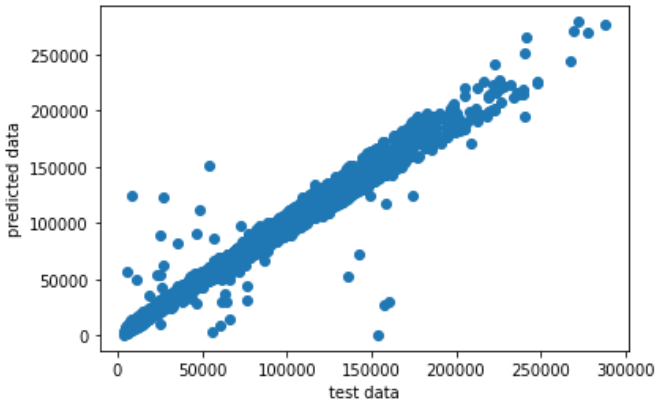
```python
# plotting a scatter plot of the test output vs the predicted output

import matplotlib.pyplot as plt

plt.scatter(Y_test, predictions)
plt.xlabel('test data')
plt.ylabel('predicted data')

plt.show()
```

Python



Linear regression model is successfully trained, tested and evaluated. The r2 score of the trained model is 0.9693971794711593. The scatter plot of the test output vs predicted output can be seen above.

Model 2: Decision Tree Regressor

```python
# importing the decision tree regressor from sklearn
from sklearn.tree import DecisionTreeRegressor

DTR = DecisionTreeRegressor()
```

Python

```python
# training the model
DTR.fit(X_train, y_train)
```

Python

```
DecisionTreeRegressor()
```

```python
# making predictions using the test data
predictions = DTR.predict(X_test)
```

Python

```python
# measuring the r2 score of the trained model
from sklearn.metrics import r2_score

# score = DTR.score(X_test, y_test)

r2_score(y_test, predictions)
```

Python

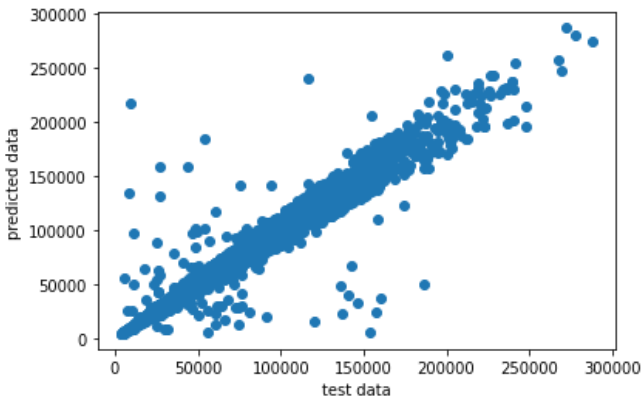
```
0.9362287012804633
```

```python
# plotting a scatter plot of the test output vs the predicted output
import matplotlib.pyplot as plt

plt.scatter(y_test, predictions)
plt.xlabel('test data')
plt.ylabel('predicted data')

plt.show()
```

Python



Decision Tree Regressor model is successfully trained, tested and evaluated. The r2 score of the trained model is 0.9362287012804633. The scatter plot of the test output vs predicted output can be seen above.

Model 3: Bayesian Ridge

```python
# importing the bayesian ridge model from sklearn models and instantiating
from sklearn.linear_model import BayesianRidge
BRmodel = BayesianRidge()
```

Python

```python
# training the model
BRmodel.fit(X_train, y_train)
```

Python

```
BayesianRidge()
```

```python
# predicting the output using the testing data
predictions = BRmodel.predict(X_test)
```

Python

```python
# checking the r2 score of the trained model
from sklearn.metrics import r2_score
r2_score(y_test, predictions)
```

Python

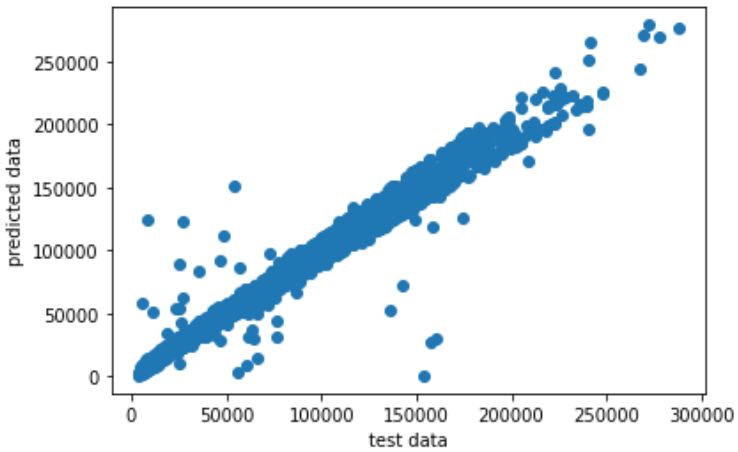
```
0.9694010419387209
```

```python
# plotting a scatter plot of test output vs the predicted output
import matplotlib.pyplot as plt

plt.scatter(y_test, predictions)
plt.xlabel('test data')
plt.ylabel('predicted data')

plt.show()
```

Python



Bayesian Ridge model is successfully trained, tested and evaluated. The r2 score of the trained model is 0.9694010419387209. The scatter plot of the test output vs predicted output can be seen above.

Machine Learning model chart

Sl no.	Algorithm name	r2 score
1	Bayesian Ridge	0.9694010419387209
2	Linear Regression	0.9693971794711593
3	Decision Tree Regressor	0.9362287012804633

Conclusion

To conclude this project, we can surely say that we have tackled the problem of predicting the loan sanction amount for an applicant by using machine learning and data science to take the workload off of the bank employees. We have successfully analyzed the data available, found patterns and using the data, trained and tested various models and ranked them on their scores. So we can surely say that these models can be used for prediction of the loan sanction amount.

Bibliography

- Geeks for geeks article on decision tree regression
- Bayesian ridge regression in scikit learn - tutorialspoint
- Pandas documentation
- seaborn documentation
- Matplotlib documentation