Project Report
(Loan Default Prediction)

Introduction to Data Analytics - MS4610

Group 22:

1. Govind Shukla - NA18B016
2. Dilip Kumar - NA18B004
3. Hari Priya - NA18B020
4. Lokesh Lakhani - NA18B024
5. Monika Nathawat - NA18B027
6. Sanjay Vasnani - NA18B036

1. Problem Statement:

Using a dataset which consists of details on loans taken by customers, we are required to build a statistical learning model to predict whether a loan will go default or not, and to understand which features are crucial in prediction.

Following details of customer are given in the dataset:

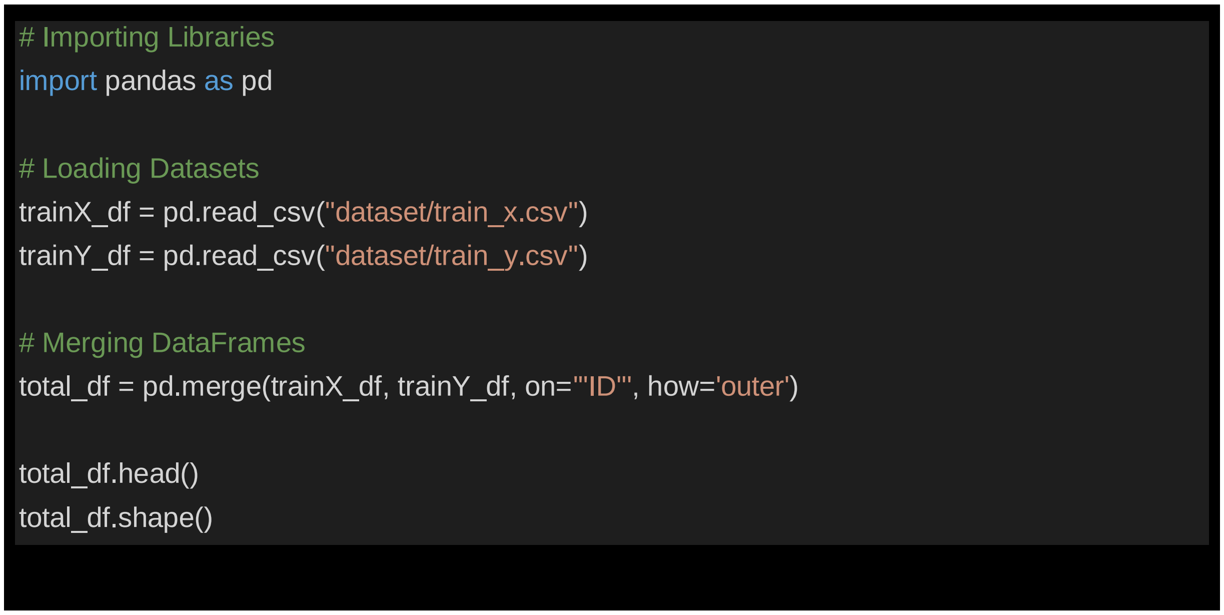
1. **ID**: Unique identifier.
2. **Loan Type:** There are two types: A and B.
3. **Occupation Type:** Occupation type of the customer. There are three types.
4. **Income:** A continuous variable indicative of the annual income.
5. **Expense:** A continuous variable indicative of the annual expense.
6. **Age:** The age of customer: 0 for < 50 and 1 for > 50.
7. **Score1-5**: Represents five different metrics calculated by the organization.
8. **Label:** 0: Non-default, 1: Default.

2. Exploratory Data Analysis (EDA):

Exploratory Data Analysis includes the preliminary conclusions which we can draw from the given data.

i. Head and Shape Functions:

To get a better understanding about the data we have used the head() function to have a look at the first five observations. The dimensions of the DataFrame are given the shape() function.

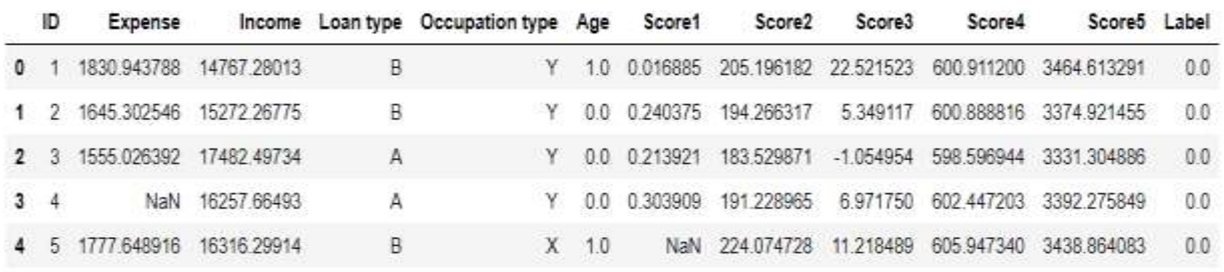


```
# Importing Libraries
import pandas as pd

# Loading Datasets
trainX_df = pd.read_csv("dataset/train_x.csv")
trainY_df = pd.read_csv("dataset/train_y.csv")

# Merging DataFrames
total_df = pd.merge(trainX_df, trainY_df, on='"ID"', how='outer')

total_df.head()
total_df.shape()
```



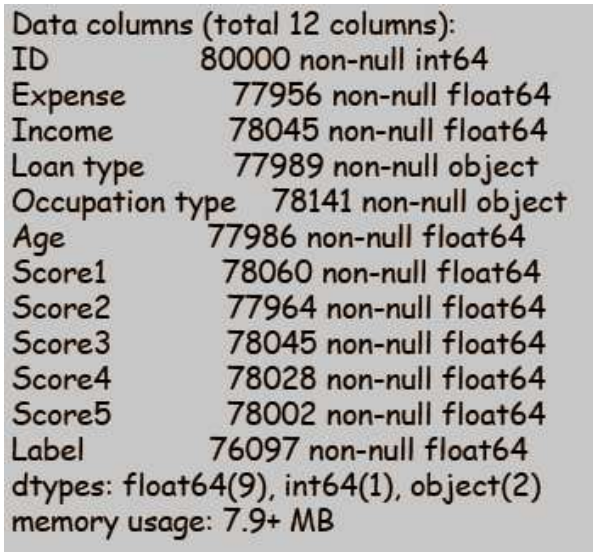
	ID	Expense	Income	Loan type	Occupation type	Age	Score1	Score2	Score3	Score4	Score5	Label
0	1	1830.943788	14767.28013	B	Y	1.0	0.016885	205.196182	22.521523	600.911200	3464.613291	0.0
1	2	1645.302546	15272.26775	B	Y	0.0	0.240375	194.266317	5.349117	600.888816	3374.921455	0.0
2	3	1555.026392	17482.49734	A	Y	0.0	0.213921	183.529871	-1.054954	598.596944	3331.304886	0.0
3	4	NaN	16257.66493	A	Y	0.0	0.303909	191.228965	6.971750	602.447203	3392.275849	0.0
4	5	1777.648916	16316.29914	B	X	1.0	NaN	224.074728	11.218489	605.947340	3438.864083	0.0

Fig 1. A glimpse into the raw data

(Dimensions are 80,000 x 12)

ii. Info Function:

We can get more information about the data frame using the info() function.



```
Data columns (total 12 columns):
ID              80000 non-null int64
Expense           77956 non-null float64
Income            78045 non-null float64
Loan type         77989 non-null object
Occupation type    78141 non-null object
Age             77986 non-null float64
Score1            78060 non-null float64
Score2            77964 non-null float64
Score3            78045 non-null float64
Score4            78028 non-null float64
Score5            78002 non-null float64
Label           76097 non-null float64
dtypes: float64(9), int64(1), object(2)
memory usage: 7.9+ MB
```

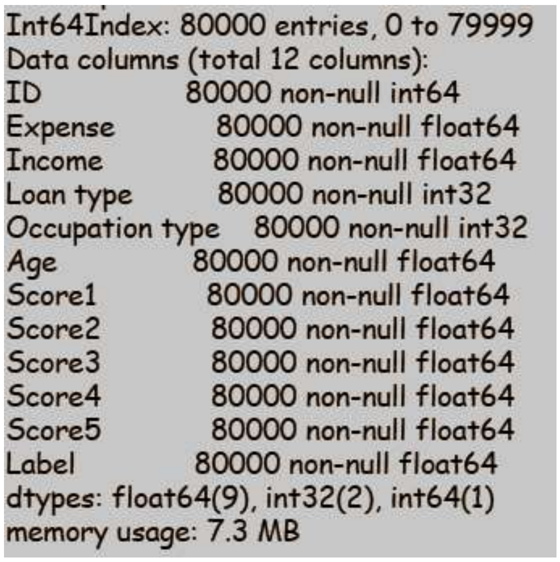
From this, we can understand that there are missing values which have to be filled and also columns with object data types which have to be changed to either float or int to go ahead with the analysis.We have replaced the null values with mode values and changed categorical variables into numerical.

Fig 2. Info on the raw data

```
# Filling Null Values
trainX_df.Expense.fillna(trainX_df.Expense.mode()[0], inplace = True)
trainX_df.Income.fillna(trainX_df.Income.mode()[0], inplace = True)
trainX_df['Loan type'].fillna(trainX_df['Loan type'].mode()[0], inplace = True)
trainX_df['Occupation type'].fillna(trainX_df['Occupation type'].mode()[0], inplace = True)
trainX_df.Age.fillna(trainX_df.Age.mode()[0], inplace = True)
trainX_df.Score1.fillna(trainX_df.Score1.mode()[0], inplace = True)
trainX_df.Score2.fillna(trainX_df.Score2.mode()[0], inplace = True)
trainX_df.Score3.fillna(trainX_df.Score3.mode()[0], inplace = True)
trainX_df.Score4.fillna(trainX_df.Score4.mode()[0], inplace = True)
trainX_df.Score5.fillna(trainX_df.Score5.mode()[0], inplace = True)
trainY_df.Label.fillna(trainY_df.Label.mode()[0], inplace = True)

# Replacing Qualitative Values
trainX_df['Loan type'].replace(['A', 'B'], [0, 1], inplace = True)
trainX_df['Occupation type'].replace(['X', 'Y', 'Z'], [0, 1, 2], inplace = True)
```

After processing the DataFrame, we recheck using the info() function.

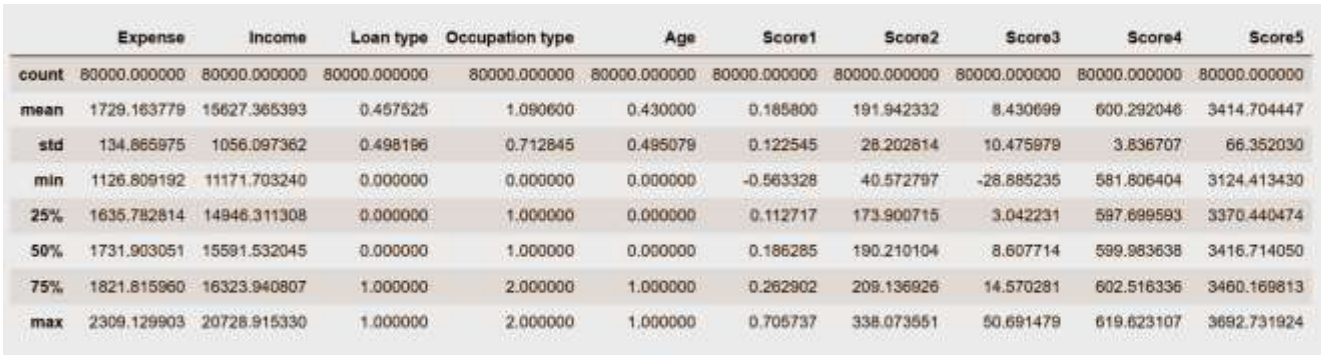


```
Int64Index: 80000 entries, 0 to 79999
Data columns (total 12 columns):
ID                80000 non-null int64
Expense           80000 non-null float64
Income            80000 non-null float64
Loan type         80000 non-null int32
Occupation type   80000 non-null int32
Age               80000 non-null float64
Score1            80000 non-null float64
Score2            80000 non-null float64
Score3            80000 non-null float64
Score4            80000 non-null float64
Score5            80000 non-null float64
Label             80000 non-null float64
dtypes: float64(9), int32(2), int64(1)
memory usage: 7.3 MB
```

Fig 3. Info on the processed data

iii. Describe Function:

The describe() function provides more information about the DataFrame, but from a statistical perspective.

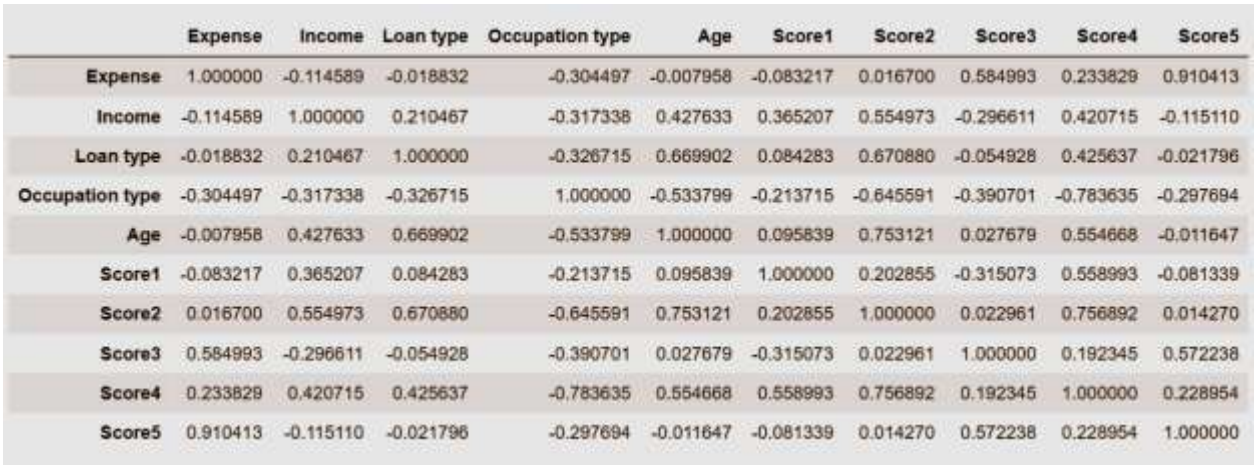


	Expense	Income	Loan type	Occupation type	Age	Score1	Score2	Score3	Score4	Score5
count	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000
mean	1729.163779	15627.365393	0.457525	1.090600	0.430000	0.185800	191.942332	8.430699	600.292046	3414.704447
std	134.865975	1056.097362	0.498196	0.712845	0.495079	0.122545	28.202814	10.475979	3.836707	66.352030
min	1126.809192	11171.703240	0.000000	0.000000	0.000000	-0.563328	40.572797	-28.885235	581.806404	3124.413430
25%	1635.782814	14946.311308	0.000000	1.000000	0.000000	0.112717	173.900715	3.042231	597.699593	3370.440474
50%	1731.903051	15591.532045	0.000000	1.000000	0.000000	0.186285	190.210104	8.607714	599.983638	3416.714050
75%	1821.815960	16323.940807	1.000000	2.000000	1.000000	0.262902	209.136926	14.570281	602.516336	3460.169813
max	2309.129903	20728.915330	1.000000	2.000000	1.000000	0.705737	338.073551	50.691479	619.623107	3692.731924

Fig 4. Descriptive statistics of the data

iv. Corr Function:

The corr() function provides the correlation matrix and subsequently, seaborn.heatmap is used to plot the heat map of the matrix.



	Expense	Income	Loan type	Occupation type	Age	Score1	Score2	Score3	Score4	Score5
Expense	1.000000	-0.114589	-0.018832	-0.304497	-0.007958	-0.083217	0.016700	0.584993	0.233829	0.910413
Income	-0.114589	1.000000	0.210467	-0.317338	0.427633	0.365207	0.554973	-0.296611	0.420715	-0.115110
Loan type	-0.018832	0.210467	1.000000	-0.326715	0.669902	0.084283	0.670880	-0.054928	0.425637	-0.021796
Occupation type	-0.304497	-0.317338	-0.326715	1.000000	-0.533799	-0.213715	-0.645591	-0.390701	-0.783635	-0.297694
Age	-0.007958	0.427633	0.669902	-0.533799	1.000000	0.095839	0.753121	0.027679	0.554668	-0.011647
Score1	-0.083217	0.365207	0.084283	-0.213715	0.095839	1.000000	0.202855	-0.315073	0.558993	-0.081339
Score2	0.016700	0.554973	0.670880	-0.645591	0.753121	0.202855	1.000000	0.022961	0.756892	0.014270
Score3	0.584993	-0.296611	-0.054928	-0.390701	0.027679	-0.315073	0.022961	1.000000	0.192345	0.572238
Score4	0.233829	0.420715	0.425637	-0.783635	0.554668	0.558993	0.756892	0.192345	1.000000	0.228954
Score5	0.910413	-0.115110	-0.021796	-0.297694	-0.011647	-0.081339	0.014270	0.572238	0.228954	1.000000

Fig 5. Correlation matrix of the data

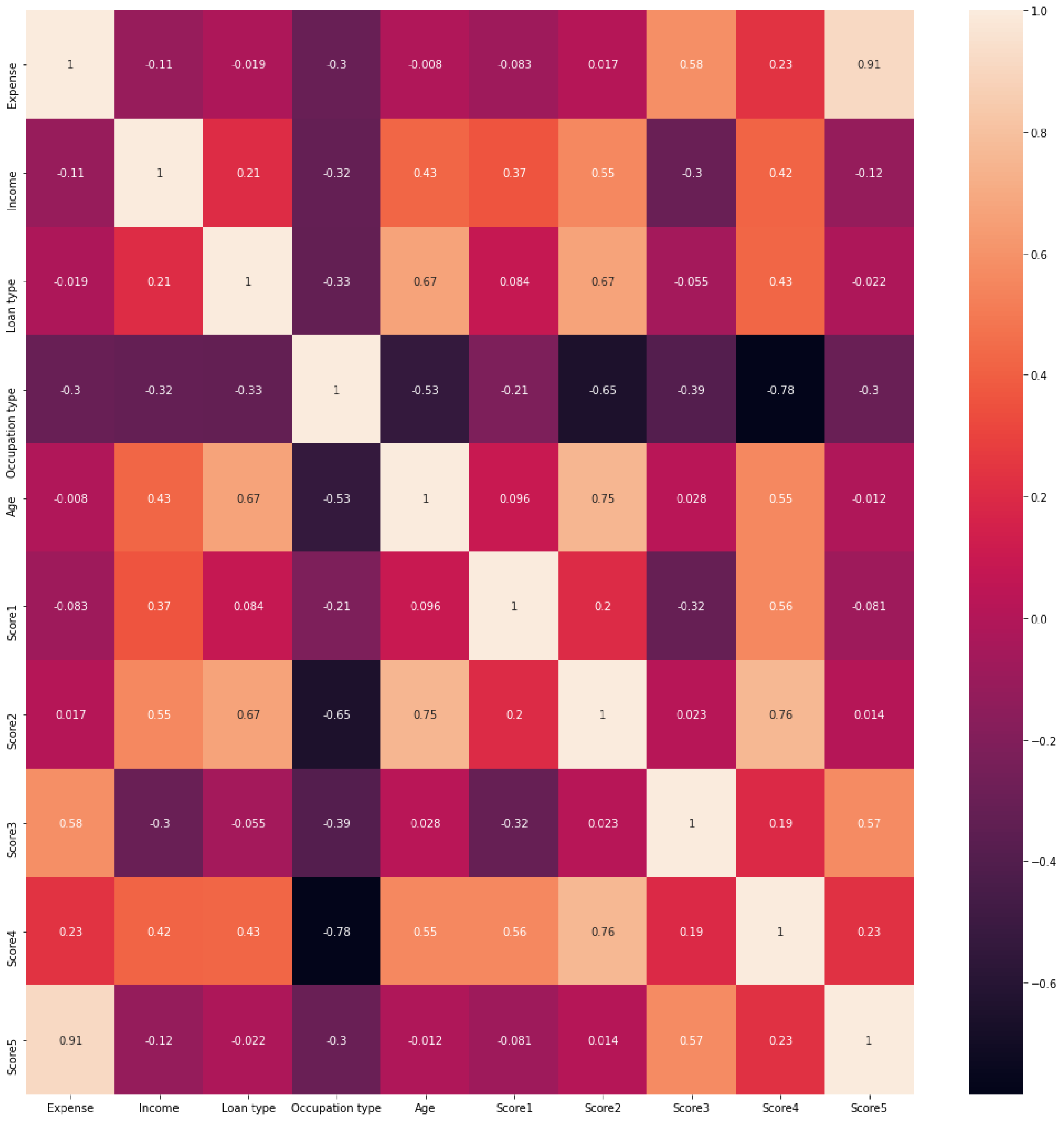


Fig 6. Heatmap for the Correlation matrix

v. Distribution Plots:

The distribution plots for all the variables are generated using the matplotlib library.

```python
import matplotlib as plt

total.hist(bins = 30, figsize = (10,10), color = 'r')
```

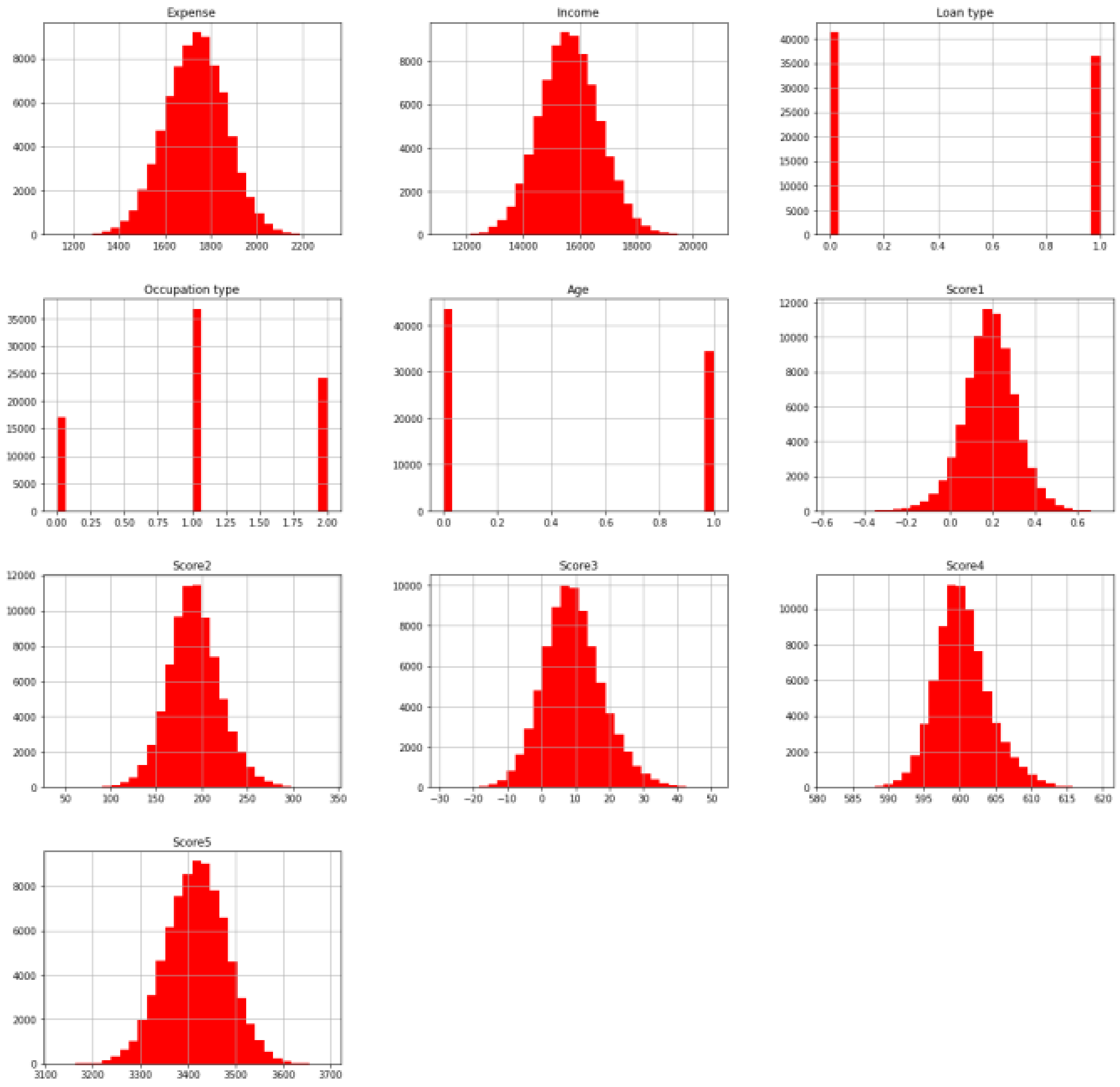


Fig 7. Distribution plots of all the variables

3. Data Scaling:

Since all the qualitative variables have a roughly Gaussian distribution, we'll use the StandardScaler function to turn them into a normal distribution with mean zero and unit variance.

```
from sklearn.preprocessing import StandardScaler
# Feature Scaling
scaler.fit(X_train)
X_train = scaler.transform(X_train)

scaler.fit(X_test)
X_test = scaler.transform(X_test)
```

4. Cross Validation:

Cross validation is a method used to evaluate statistical learning models. In cross validation, we split the training dataset into two sets. The larger set is used for training and the complementary set is used to test the model. This process is repeated multiple times and the metric to be evaluated is averaged.
In this program, a 10-Fold CV is used, and it is repeated 5 times. The metric we calculate is accuracy.

```
from sklearn.model_selection import RepeatedKFold, cross_val_score

# Cross Validation
cv = RepeatedKFold(n_splits = 10, n_repeats = 5, random_state = rd.seed())
scores = cross_val_score(model, X_train, y_train, scoring = 'accuracy', cv = cv, n_jobs = -1)
print("Accuracy: %.4f" % (np.mean(scores)))
```

5. Training and Predicting using Different Models:

All accuracy results are obtained from 10-fold Cross Validation over the training set, and the remaining metrics are obtained by comparing with the test labels.

i. Decision Tree:

Accuracy	96.7%
Precision	73.74%
Recall	73.84%
F1 Score	73.79%

ii. Random Forest:

Accuracy	97.56%
Precision	91.04%
Recall	69.34%
F1 Score	78.72%

iii. Logistic Regression:

Accuracy	95.56%
Precision	84.55%
Recall	39.87%
F1 Score	54.19%

iv. K-Nearest Neighbors (14 Neighbours):

Accuracy	97.53%
Precision	92.01%
Recall	68.68%
F1 Score	78.65%

v. XGBoost:

Accuracy	97.90%
Precision	89.17%
Recall	75.76%
F1 Score	81.92%

6. Final Model Selection and Prediction:

After comparing the results seen above, we settle on XGBoost as the best choice because it maximizes all the metrics we are evaluating.

XGBoost is a variant of a gradient boosted decision tree. It is preferred over other gradient boosted decision tree algorithms because it provides an advantage in computing time and accuracy.

```
from xgboost import XGBClassifier

# XGBoost
model = XGBClassifier(verbosity = 0, use_label_encoder = False, objective = 'binary:logistic',
booster = 'gbtree' )
model.fit(X, y)
labels = model.predict(X_test)
labels = [round(value) for value in labels]

# Exporting
y_pred = pd.DataFrame(testX_df["'ID_Test'"].to_numpy(), columns = ["'ID_Test'"])
y_pred['Label'] = labels
y_pred.to_csv('dataset/y_pred.csv', index = False)
```

After using the model to predict the labels for the test dataset, we export it into CSV form.