LOAN PREDICTION ML MODEL BY EMMANUEL ADEITAN

In [ ]:
```
#standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1.Get The Data Ready

In [2]:
```
#import dataset

loan_default = pd.read_csv('Loan_Default.csv')

#view the data

loan_default.head()
```

Out[2]:

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	...	credit_type	Credit_Score	co-applicant_credit_type	age	submission_of_application	LTV	Regio
0	24890	2019	cf	Sex Not Available	nopre	type1	p1	l1	nopc	nob/c	...	EXP	758	CIB	25-34	to_inst	98.728814	sou
1	24891	2019	cf	Male	nopre	type2	p1	l1	nopc	b/c	...	EQUI	552	EXP	55-64	to_inst	NaN	Nor
2	24892	2019	cf	Male	pre	type1	p1	l1	nopc	nob/c	...	EXP	834	CIB	35-44	to_inst	80.019685	sou
3	24893	2019	cf	Male	nopre	type1	p4	l1	nopc	nob/c	...	EXP	587	CIB	45-54	not_inst	69.376900	Nor
4	24894	2019	cf	Joint	pre	type1	p1	l1	nopc	nob/c	...	CRIF	602	EXP	25-34	not_inst	91.886544	Nor

5 rows × 34 columns

In [4]:
```
loan_default.shape
```

Out[4]:
```
(148670, 34)
```

There are 148,670 Rolls and 34 Columns(features) in the dataframe.

The Features are;

ID = Customer ID of Applicant

year = Year of Application

loan limit = maximum avaliable amount of the loan allowed to be taken

Gender = sex type

approv_in_adv = Is loan pre-approved or not

loan_type = Type of loan

loan_purpose = the reason you want to borrow money

Credit_Worthiness = is how a lender determines that you will default on your debt obligations, or how worthy you are to receive new credit.

open_credit = is a pre-approved loan between a lender and a borrower. It allows the borrower to make repeated withdrawals up to a certain limit.

business_or_commercial = Usage type of the loan amount

loan_amount = The exact loan amount

rate_of_interest = is the amount a lender charges a borrower and is a percentage of the principal—the amount loaned.

Interest_rate_spread = the difference between the interest rate a financial institution pays to depositors and the interest rate it receives from loans

Upfront_charges = Fee paid to a lender by a borrower as consideration for making a new loan

term = the loan's repayment period

Neg_ammortization = refers to a situation when a loan borrower makes a payment less than the standard installment set by the bank.

interest_only = amount of interest only without principles

lump_sum_payment = is an amount of money that is paid in one single payment rather than in installments.

property_value = the present worth of future benefits arising from the ownership of the property

construction_type = Collateral construction type

occupancy_type = classifications refer to categorizing structures based on their usage

Secured_by = Type of Collatoral

total_units = number of unites

income = refers to the amount of money, property, and other transfers of value received over a set period of time

credit_type = type of credit

co-applicant_credit_type = is an additional person involved in the loan application process. Both applicant and co-applicant apply and sign for the loan

age = applicant's age

submission_of_application = Ensure the application is complete or not

LTV = life-time value (LTV) is a prognostication of the net profit

Region = applicant's place

Security_Type = Type of Collatoral

status = Loan status (Approved/Declined)

dtir1 = debt-to-income ratio

In [17]:
```
#The dataFrame is not displaying all of the 34 features.
#To solve this we set number of columns we want to display with pandas settings

pd.set_option("display.max_columns", loan_default.shape[-1])
loan_default.head()
```

Out[17]:

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	loan_amount	rate_of_interest	Interest_rate_spread	Upfront_charges	term	Neg_ammortization	inter
0	24890	2019	cf	Sex Not Available	nopre	type1	p1	l1	nopc	nob/c	116500	NaN	NaN	NaN	360.0	not_neg	
1	24891	2019	cf	Male	nopre	type2	p1	l1	nopc	b/c	206500	NaN	NaN	NaN	360.0	not_neg	
2	24892	2019	cf	Male	pre	type1	p1	l1	nopc	nob/c	406500	4.56	0.2000	595.0	360.0	neg_amm	
3	24893	2019	cf	Male	nopre	type1	p4	l1	nopc	nob/c	456500	4.25	0.6810	NaN	360.0	not_neg	
4	24894	2019	cf	Joint	pre	type1	p1	l1	nopc	nob/c	696500	4.00	0.3042	0.0	360.0	not_neg	

In [18]:
```
loan_default.columns
```

Out[18]:
```
Index(['ID', 'year', 'loan_limit', 'Gender', 'approv_in_adv', 'loan_type',
       'loan_purpose', 'Credit_Worthiness', 'open_credit',
       'business_or_commercial', 'loan_amount', 'rate_of_interest',
       'Interest_rate_spread', 'Upfront_charges', 'term', 'Neg_ammortization',
       'interest_only', 'lump_sum_payment', 'property_value',
       'construction_type', 'occupancy_type', 'Secured_by', 'total_units',
       'income', 'credit_type', 'Credit_Score', 'co-applicant_credit_type',
       'age', 'submission_of_application', 'LTV', 'Region', 'Security_Type',
       'Status', 'dtir1'],
      dtype='object')
```

In [19]:
```
#converting all features to lower case for easy accessibility.

loan_default.columns = loan_default.columns.str.lower()
loan_default.columns
```

Out[19]:
```
Index(['id', 'year', 'loan_limit', 'gender', 'approv_in_adv', 'loan_type',
       'loan_purpose', 'credit_worthiness', 'open_credit',
       'business_or_commercial', 'loan_amount', 'rate_of_interest',
       'interest_rate_spread', 'upfront_charges', 'term', 'neg_ammortization',
       'interest_only', 'lump_sum_payment', 'property_value',
       'construction_type', 'occupancy_type', 'secured_by', 'total_units',
       'income', 'credit_type', 'credit_score', 'co-applicant_credit_type',
       'age', 'submission_of_application', 'ltv', 'region', 'security_type',
       'status', 'dtir1'],
      dtype='object')
```

Filtering the data, since there are large number of unuseful columns which can be dropped without a bad effect on the performance of the model.

In [20]:
```
#we can drpo 'id' ,'loan_limit' , 'gender' ,'approv_in_adv' ,'loan_purpose' ,'open_credit' ,'business_or_commercial'  because they don't add any valiable information.
#we can drop 'year' as it includes only one year 2019 so we have the same year.

columns= ['id', 'year', 'loan_limit', 'gender', 'approv_in_adv','loan_purpose', 'credit_worthiness', 'open_credit',
          'business_or_commercial','interest_rate_spread', 'upfront_charges', 'neg_ammortization','interest_only',
          'lump_sum_payment','construction_type', 'occupancy_type', 'secured_by', 'total_units','credit_type',
          'co-applicant_credit_type','submission_of_application', 'ltv', 'region', 'security_type']
loan_default.drop(columns,axis=1, inplace =True)
```

In [21]:
```
loan_default.columns
```

Out[21]:
```
Index(['loan_type', 'loan_amount', 'rate_of_interest', 'term',
       'property_value', 'income', 'credit_score', 'age', 'status', 'dtir1'],
      dtype='object')
```

In [22]:
```
loan_default.shape
```

Out[22]:
```
(148670, 10)
```

In [23]:
```
pd.set_option("display.max_columns", loan_default.shape[-1])
loan_default.head()
```

Out[23]:

	loan_type	loan_amount	rate_of_interest	term	property_value	income	credit_score	age	status	dtir1
0	type1	116500	NaN	360.0	118000.0	1740.0	758	25-34	1	45.0
1	type2	206500	NaN	360.0	NaN	4980.0	552	55-64	1	NaN
2	type1	406500	4.56	360.0	508000.0	9480.0	834	35-44	0	46.0
3	type1	456500	4.25	360.0	658000.0	11880.0	587	45-54	0	42.0
4	type1	696500	4.00	360.0	758000.0	10440.0	602	25-34	0	39.0

CHECKING FOR DUPLICATES

In [25]:
```
loan_default.duplicated().sum()
```

Out[25]:
```
6
```

In [26]:
```
loan_default.drop_duplicates(inplace=True)
```

In [27]:
```
loan_default.duplicated().sum()
```

Out[27]:
```
0
```

In [28]:
```
loan_default.shape
```

Out[28]:
```
(148664, 10)
```

CHECKING FOR NAN VALUES

In [30]:
```
loan_default.isna().sum()
```

Out[30]:
```
loan_type               0
loan_amount             0
rate_of_interest    36437
term                   41
property_value      15096
income               9146
credit_score            0
age                   200
status                  0
dtir1               24115
dtype: int64
```

In [32]:
```
loan_default.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148664 entries, 0 to 148669
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   loan_type         148664 non-null  object 
 1   loan_amount       148664 non-null  int64  
 2   rate_of_interest  112227 non-null  float64
 3   term              148623 non-null  float64
 4   property_value    133568 non-null  float64
 5   income            139518 non-null  float64
 6   credit_score      148664 non-null  int64  
 7   age               148464 non-null  object 
 8   status            148664 non-null  int64  
 9   dtir1             124549 non-null  float64
dtypes: float64(5), int64(3), object(2)
memory usage: 12.5+ MB
```

In [33]:
```
#filling the null in columns with int and floats.
from sklearn.impute import SimpleImputer
imputer = SimpleImputer()
loan_default[['rate_of_interest', 'term',
    'property_value','income','dtir1']] = imputer.fit_transform(loan_default[['rate_of_interest',
                                                                 'term','property_value',
                                                                 'income','dtir1']])
```

In [34]:
```
#filling the null in columns with objects.
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='most_frequent')
loan_default[['age']] = imputer.fit_transform(loan_default[['age']])
```

In [35]:
```
loan_default.isna().sum()
```

Out[35]:
```
loan_type           0
loan_amount         0
rate_of_interest    0
term                0
property_value      0
income              0
credit_score        0
age                 0
status              0
dtir1               0
dtype: int64
```

In [36]:
```
#Dealing with non numerical values
#All data must be numerical to be used in the machine learning model

loan_default.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148664 entries, 0 to 148669
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   loan_type         148664 non-null  object 
 1   loan_amount       148664 non-null  int64  
 2   rate_of_interest  148664 non-null  float64
 3   term              148664 non-null  float64
 4   property_value    148664 non-null  float64
 5   income            148664 non-null  float64
 6   credit_score      148664 non-null  int64  
 7   age               148664 non-null  object 
 8   status            148664 non-null  int64  
 9   dtir1             148664 non-null  float64
dtypes: float64(5), int64(3), object(2)
memory usage: 12.5+ MB
```

In [37]:
```
loan_default = pd.get_dummies(loan_default, columns=['loan_type','age'], drop_first=True)
```

In [38]:
```
loan_default.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148664 entries, 0 to 148669
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   loan_amount       148664 non-null  int64  
 1   rate_of_interest  148664 non-null  float64
 2   term              148664 non-null  float64
 3   property_value    148664 non-null  float64
 4   income            148664 non-null  float64
 5   credit_score      148664 non-null  int64  
 6   status            148664 non-null  int64  
 7   dtir1             148664 non-null  float64
 8   loan_type_type2   148664 non-null  uint8  
 9   loan_type_type3   148664 non-null  uint8  
 10  age_35-44         148664 non-null  uint8  
 11  age_45-54         148664 non-null  uint8  
 12  age_55-64         148664 non-null  uint8  
 13  age_65-74         148664 non-null  uint8  
 14  age_<25           148664 non-null  uint8  
 15  age_>74           148664 non-null  uint8  
dtypes: float64(5), int64(3), uint8(8)
memory usage: 11.3 MB
```

In [39]:
```
pd.set_option("display.max_columns", loan_default.shape[-1])
loan_default.head()
```

Out[39]:

	loan_amount	rate_of_interest	term	property_value	income	credit_score	status	dtir1	loan_type_type2	loan_type_type3	age_35-44	age_45-54	age_55-64	age_65-74	age_<25	age_>74
0	116500	4.045482	360.0	118000.000000	1740.0	758	1	45.000000	0	0	0	0	0	0	0	0
1	206500	4.045482	360.0	497900.200647	4980.0	552	1	37.732932	1	0	0	0	1	0	0	0
2	406500	4.560000	360.0	508000.000000	9480.0	834	0	46.000000	0	0	1	0	0	0	0	0
3	456500	4.250000	360.0	658000.000000	11880.0	587	0	42.000000	0	0	0	1	0	0	0	0
4	696500	4.000000	360.0	758000.000000	10440.0	602	0	39.000000	0	0	0	0	0	0	0	0

In [41]:
```
loan_default.shape
```

Out[41]:
```
(148664, 16)
```

1. Choose the model/estimator Split Data to train and test sets

In [42]:
```
# Create X (all the feature columns)
x = loan_default.drop('status', axis=1)
# Create y (the target column)
y = loan_default['status']
```

In [43]:
```
x
```

Out[43]:

	loan_amount	rate_of_interest	term	property_value	income	credit_score	dtir1	loan_type_type2	loan_type_type3	age_35-44	age_45-54	age_55-64	age_65-74	age_<25	age_>74
0	116500	4.045482	360.0	118000.000000	1740.0	758	45.000000	0	0	0	0	0	0	0	0
1	206500	4.045482	360.0	497900.200647	4980.0	552	37.732932	1	0	0	0	1	0	0	0
2	406500	4.560000	360.0	508000.000000	9480.0	834	46.000000	0	0	1	0	0	0	0	0
3	456500	4.250000	360.0	658000.000000	11880.0	587	42.000000	0	0	0	1	0	0	0	0
4	696500	4.000000	360.0	758000.000000	10440.0	602	39.000000	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
148665	436500	3.125000	180.0	608000.000000	7860.0	659	48.000000	0	0	0	0	1	0	0	0
148666	586500	5.190000	360.0	788000.000000	7140.0	569	15.000000	0	0	0	0	0	0	0	0
148667	446500	3.125000	180.0	728000.000000	6900.0	702	49.000000	0	0	0	1	0	0	0	0
148668	196500	3.500000	180.0	278000.000000	7140.0	737	29.000000	0	0	0	0	1	0	0	0
148669	406500	4.375000	240.0	558000.000000	7260.0	830	44.000000	0	0	0	1	0	0	0	0

148664 rows × 15 columns

In [44]:
```
y
```

Out[44]:
```
0         1
1         1
2         0
3         0
4         0
         ..
148665    0
148666    0
148667    0
148668    0
148669    0
Name: status, Length: 148664, dtype: int64
```

In [45]:
```
x.shape
```

Out[45]:
```
(148664, 15)
```

In [46]:
```
y.shape
```

Out[46]:
```
(148664,)
```

In [57]:
```
# Split the data into training and test sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

View the data shapes

In [58]:
```
x_train
```

Out[58]:

	loan_amount	rate_of_interest	term	property_value	income	credit_score	dtir1	loan_type_type2	loan_type_type3	age_35-44	age_45-54	age_55-64	age_65-74	age_<25	age_>74
59491	666500	3.990000	360.0	848000.000000	9060.000000	651	44.000000	0	0	0	1	0	0	0	0
105728	166500	3.750000	180.0	238000.000000	4380.000000	854	44.000000	0	0	0	0	0	0	0	0
71354	136500	3.750000	360.0	198000.000000	2760.000000	775	30.000000	0	0	0	1	0	0	0	0
60811	226500	3.500000	180.0	378000.000000	5160.000000	683	40.000000	0	0	0	0	0	0	0	1
28296	406500	4.250000	360.0	838000.000000	6240.000000	510	47.000000	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
101798	566500	3.990000	360.0	828000.000000	9180.000000	519	41.000000	0	0	0	1	0	0	0	0
49440	136500	3.990000	360.0	208000.000000	3360.000000	831	45.000000	0	0	0	1	0	0	0	0
34886	276500	3.375000	360.0	318000.000000	6957.392164	884	37.732932	1	0	1	0	0	0	0	0
52828	166500	4.045482	360.0	497900.200647	3000.000000	509	37.732932	0	1	0	1	0	0	0	0
70000	166500	4.045482	240.0	497900.200647	3060.000000	784	37.732932	0	0	1	0	0	0	0	0

104064 rows × 15 columns

In [59]:
```
x_train.shape
```

Out[59]:
```
(104064, 15)
```

In [60]:
```
x_test
```

Out[60]:

	loan_amount	rate_of_interest	term	property_value	income	credit_score	dtir1	loan_type_type2	loan_type_type3	age_35-44	age_45-54	age_55-64	age_65-74	age_<25	age_>74
142373	456500	3.875000	360.0	7.580000e+05	7800.000000	774	29.000000	0	0	1	0	0	0	0	0
73668	286500	4.375000	360.0	3.080000e+05	6300.000000	513	28.000000	0	0	1	0	0	0	0	0
23085	516500	4.375000	360.0	7.180000e+05	7140.000000	763	41.000000	0	0	0	0	1	0	0	0
77959	476500	3.375000	180.0	4.980000e+05	20340.000000	894	28.000000	0	0	0	1	0	0	0	0
86151	236500	4.045482	180.0	4.979002e+05	6900.000000	656	37.732932	0	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
74666	426500	3.625000	360.0	4.980000e+05	8460.000000	515	39.000000	0	0	0	0	1	0	0	0
105323	726500	3.500000	360.0	1.258000e+06	14940.000000	611	40.000000	0	0	0	0	1	0	0	0
1706	236500	4.500000	360.0	3.480000e+05	4560.000000	774	59.000000	1	0	0	0	0	1	0	0
136778	316500	4.875000	360.0	4.280000e+05	6360.000000	707	38.000000	0	0	0	1	0	0	0	0
113676	546500	3.000000	360.0	6.780000e+05	6957.392164	773	37.732932	0	1	0	0	1	0	0	0

44600 rows × 15 columns

In [61]:
```
x_test.shape
```

Out[61]:
```
(44600, 15)
```

In [62]:
```
y_train
```

Out[62]:
```
59491     0
105728    0
71354     0
60811     0
28296     0
         ..
101798    0
49440     0
34886     0
52828     1
70000     1
Name: status, Length: 104064, dtype: int64
```

In [63]:
```
y_train.shape
```

Out[63]:
```
(104064,)
```

In [64]:
```
y_test
```

Out[64]:
```
142373    0
73668     0
23085     0
77959     0
86151     1
         ..
74666     0
105323    0
1706      0
136778    0
113676    0
Name: status, Length: 44600, dtype: int64
```

In [65]:
```
y_test.shape
```

Out[65]:
```
(44600,)
```

In [66]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

1. Fit the model to the data and use it to make a prediction

A model will (attempt to) learn the patterns in a dataset by calling the fit() function on it and passing it the data.

In [68]:
```
model.fit(x_train, y_train)
```

Out[68]:
```
RandomForestClassifier()
```

Once a model has learned patterns in data, you can use them to make a prediction with the predict() function.

In [69]:
```
# Make predictions
y_preds = model.predict(x_test)
```

In [70]:
```
# This will be in the same format as y_test
y_preds
```

Out[70]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

1. Evaluate the model

A trained model/estimator can be evaluated by calling the score() function and passing it a collection of data.

In [77]:
```
# On the training set
print(f'training accuaracy: {model.score(x_train,y_train)}')
```

```
training accuaracy: 1.0
```

In [78]:
```
# On the test set (unseen)
print(f'testing accuaracy: {model.score(x_test,y_test)}')
```

```
testing accuaracy: 0.9995739910313901
```

we got accuracy of 100% for training score and 99% for testing score.