Data Analysis on Loan Dataset

Aditya Kulshrestha | kulshresthaditya02@gmail.com | 8787252884

Data Description

We have been provided with the data which contains value of a loan details. Our objective is to do data analysis and further make a model using labels which will predict values whether the loan is good for the borrower or bad. The data does not contain any missing values; however, the labels are not in proper format, so we must do some manual cleaning of the data.

Further we have performed basic analysis of the data and did some visualization and brought import aspects to take care. We further did some feature engineering and removed few labels which doesn't have a significant impact on our target variable.

Nor we move to our mathematical modelling and here we used models like Logistic Regression, Gaussian Naïve Bayes and Random Forest Classifier. Apart from this we have also implemented deep neural network to see if it can give us better performance than our previous models.

Labels

Id, year, issue_d, final_d, emp_lenght_int, home_ownership, home_ownership_cat, income_category, annual_inc, income_cat, loan_amount, term, term_cat, application_type, application_type_cat, purpose, purpose_cat, interest_payments, interest_payments_cat, loan_condition, loan_condition_cat, interest_rate, grade, grade_cat, dti, total_pyment, total_rec_prncp, recoveries, installment, region

Keywords

Data Leakage, Logistic Regression, Gaussian Naïve Bayes, Random Forest Classifier, Deep Learning, Neural Network, OneHotEncoding

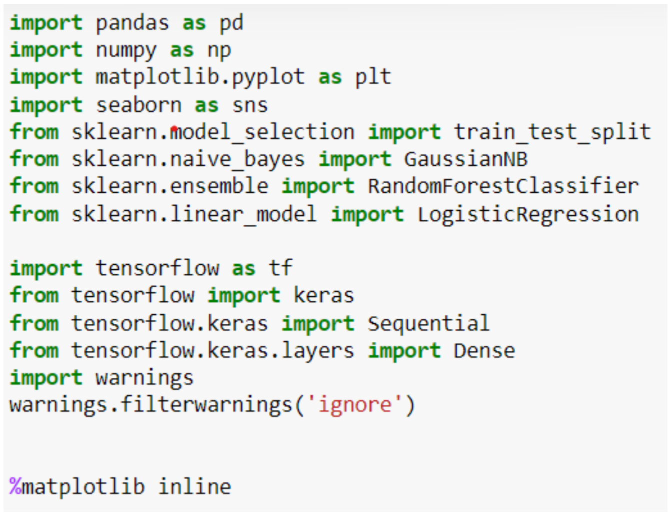
Use Case

Following dataset can be used by a service provider company or a bank to provide loan related services to their customers. The service provider can tell its client whether the loan is good for him based on his financial information like income, loan period, loan amount, loan interest, home ownership etc. The service can compare the loans of various of banks and come up with most suitable loan for its client based on the information given by the client.

Libraries used

- Two libraries for data manipulation - Pandas and Numpy has been used.

- Two libraries for data visualization – Matplotlib and Seaborn has been used
- Multiple sub libraries of sklearn are used for preprocessing and model making
- A deep learning framework – Tensorflow is also used to implement deep neural network



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')


%matplotlib inline
```

Splitting of dataset

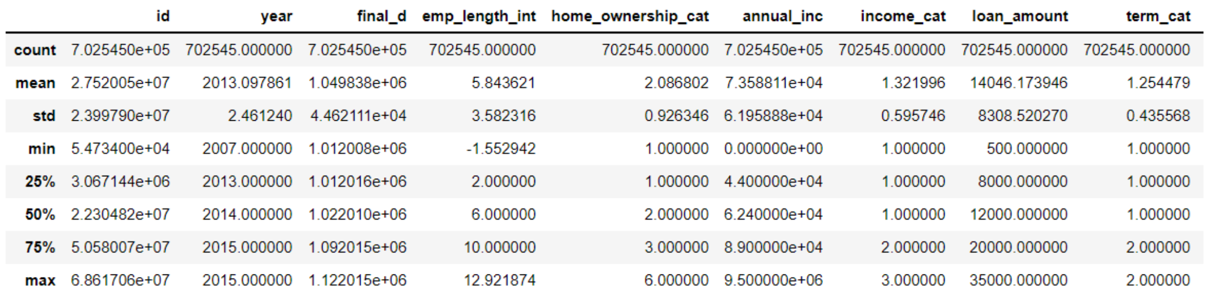
- What is data leakage?

Data leakage (or **leakage**) happens when your training data contains information about the target, but similar data will not be available when the model is used for prediction. This leads to high performance on the training set (and possibly even the validation data), but the model will perform poorly in production.

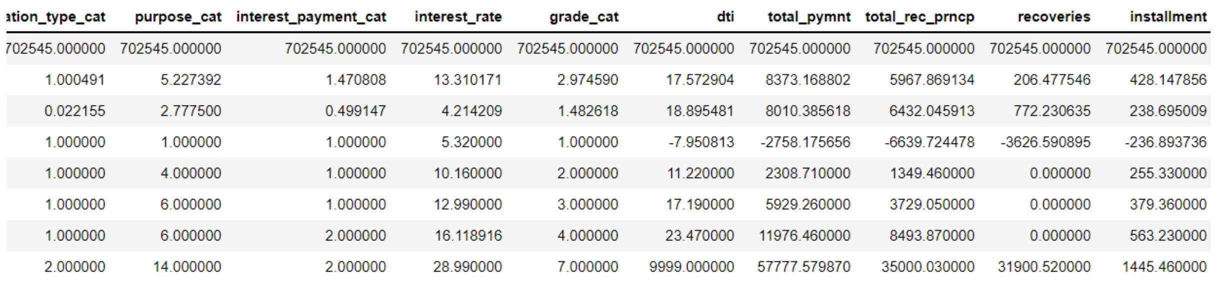
In other words, leakage causes a model to look accurate until you start making decisions with the model, and then the model becomes very inaccurate.

To avoid data leakage we generally split the data at the beginning of the data reading and separately perform data cleaning and feature engineering on the test set and the training set.

Basic Statistics of our data



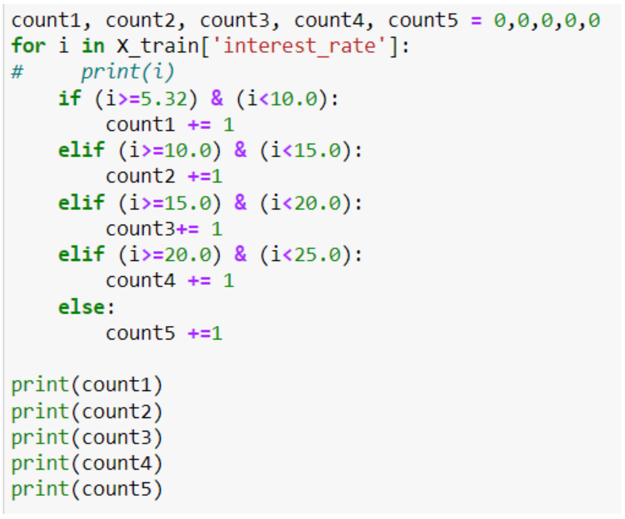
	id	year	final_d	emp_length_int	home_ownership_cat	annual_inc	income_cat	loan_amount	term_cat
count	7.025450e+05	702545.000000	7.025450e+05	702545.000000	702545.000000	7.025450e+05	702545.000000	702545.000000	702545.000000
mean	2.752005e+07	2013.097861	1.049838e+06	5.843621	2.086802	7.358811e+04	1.321996	14046.173946	1.254479
std	2.399790e+07	2.461240	4.462111e+04	3.582316	0.926346	6.195888e+04	0.595746	8308.520270	0.435568
min	5.473400e+04	2007.000000	1.012008e+06	-1.552942	1.000000	0.000000e+00	1.000000	500.000000	1.000000
25%	3.067144e+06	2013.000000	1.012016e+06	2.000000	1.000000	4.400000e+04	1.000000	8000.000000	1.000000
50%	2.230482e+07	2014.000000	1.022010e+06	6.000000	2.000000	6.240000e+04	1.000000	12000.000000	1.000000
75%	5.058007e+07	2015.000000	1.092015e+06	10.000000	3.000000	8.900000e+04	2.000000	20000.000000	2.000000
max	6.861706e+07	2015.000000	1.122015e+06	12.921874	6.000000	9.500000e+06	3.000000	35000.000000	2.000000



ation_type_cat	purpose_cat	interest_payment_cat	interest_rate	grade_cat	dti	total_pymnt	total_rec_prncp	recoveries	installment
702545.000000	702545.000000	702545.000000	702545.000000	702545.000000	702545.000000	702545.000000	702545.000000	702545.000000	702545.000000
1.000491	5.227392	1.470808	13.310171	2.974590	17.572904	8373.168802	5967.869134	206.477546	428.147856
0.022155	2.777500	0.499147	4.214209	1.482618	18.895481	8010.385618	6432.045913	772.230635	238.695009
1.000000	1.000000	1.000000	5.320000	1.000000	-7.950813	-2758.175656	-6639.724478	-3626.590895	-236.893736
1.000000	4.000000	1.000000	10.160000	2.000000	11.220000	2308.710000	1349.460000	0.000000	255.330000
1.000000	6.000000	1.000000	12.990000	3.000000	17.190000	5929.260000	3729.050000	0.000000	379.360000
1.000000	6.000000	2.000000	16.118916	4.000000	23.470000	11976.460000	8493.870000	0.000000	563.230000
2.000000	14.000000	2.000000	28.990000	7.000000	9999.000000	57777.579870	35000.030000	31900.520000	1445.460000

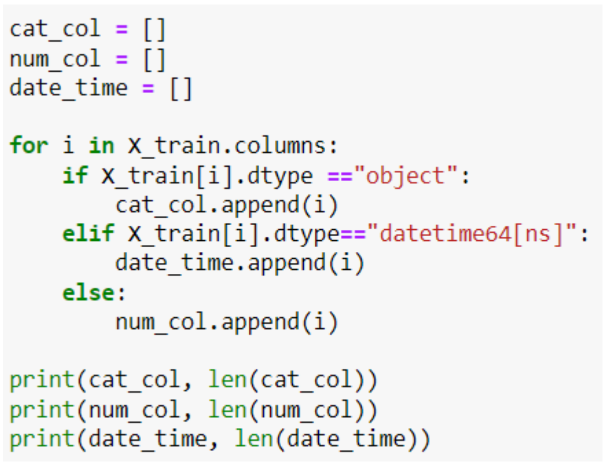
Basic EDA

None of the rows data have None values.



```
count1, count2, count3, count4, count5 = 0,0,0,0,0
for i in X_train['interest_rate']:
#     print(i)
    if (i>=5.32) & (i<10.0):
        count1 += 1
    elif (i>=10.0) & (i<15.0):
        count2 +=1
    elif (i>=15.0) & (i<20.0):
        count3+= 1
    elif (i>=20.0) & (i<25.0):
        count4 += 1
    else:
        count5 +=1

print(count1)
print(count2)
print(count3)
print(count4)
print(count5)
```

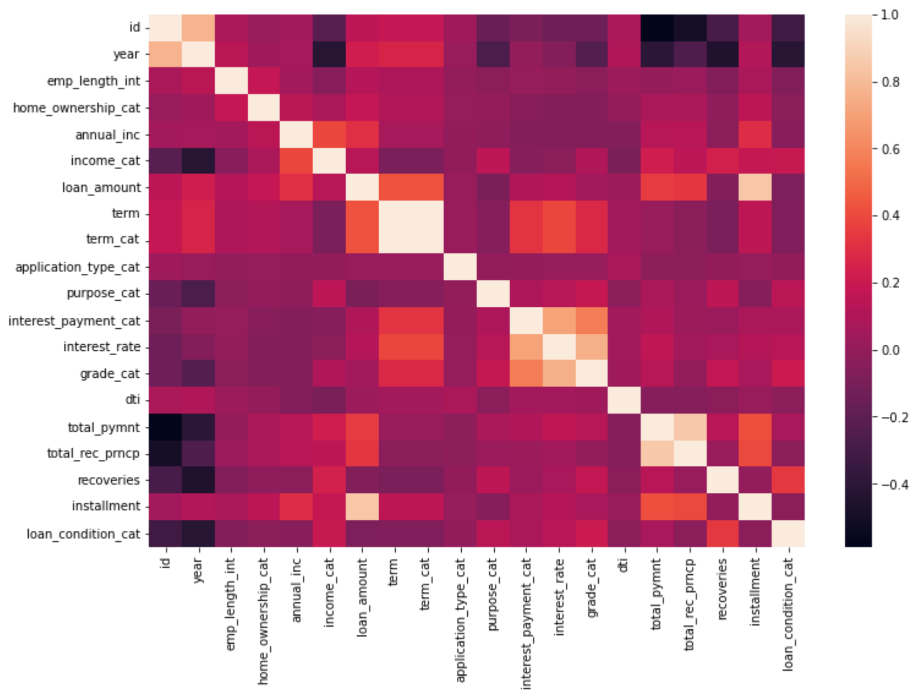
- 168007 rows have interest rate in the range of 5.32 to 10.0
- 313558 rows have interest rate in the range of 10.0 to 15.0
- 178556 rows have interest rate in the range of 15.0 to 20.0
- 38111 rows have interest rate in the range of 20.0 to 25.0
- 4313 rows have interest rate higher than 25.0



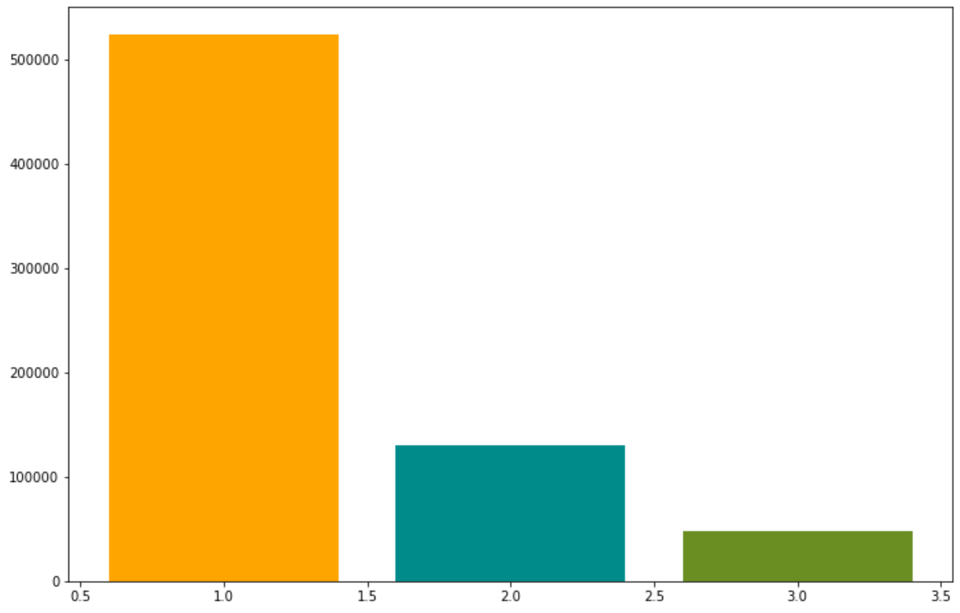
```
cat_col = []
num_col = []
date_time = []

for i in X_train.columns:
    if X_train[i].dtype =="object":
        cat_col.append(i)
    elif X_train[i].dtype=="datetime64[ns]":
        date_time.append(i)
    else:
        num_col.append(i)

print(cat_col, len(cat_col))
print(num_col, len(num_col))
print(date_time, len(date_time))
```

- 8 columns are categorical columns
- 19 columns are numerical columns
- 2 columns have datetime format

Visualization and Analysis of data

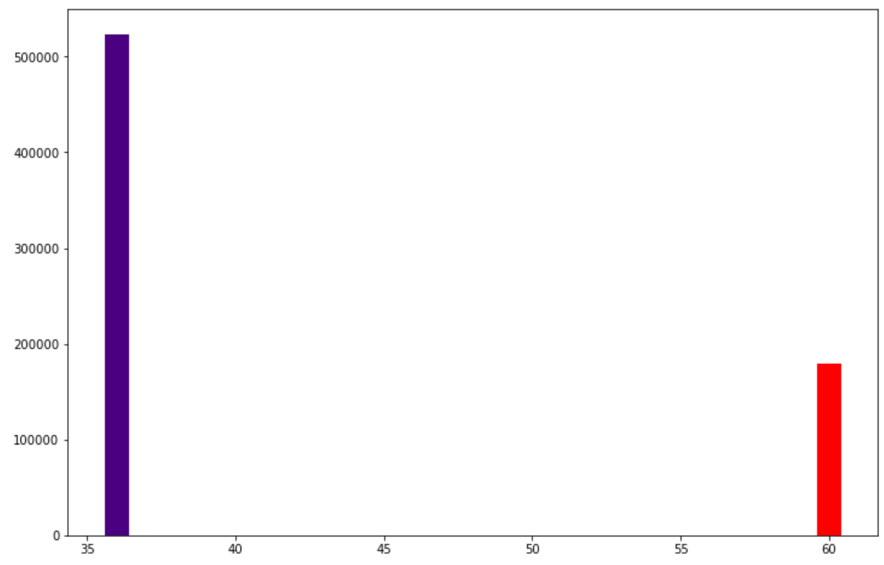


Looking across the correlation image, we can draw out that year, income_cat, purpose_cat, interest_rate grade_cat recoveries have a higher impact on the loan_category than other variable



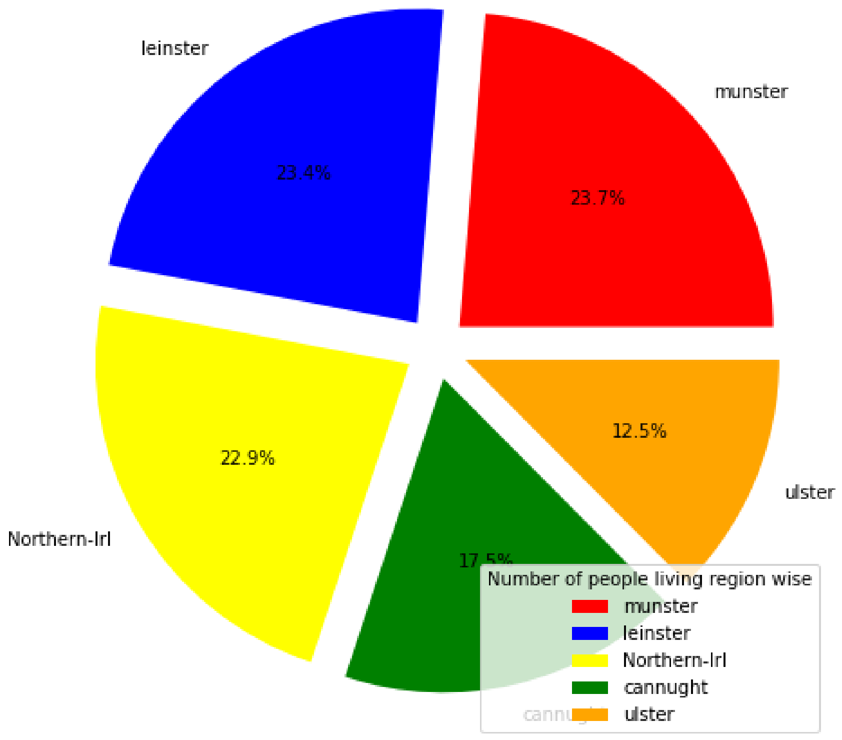
We observe that most of the loan have been applied by the person whose income lies in the category of 1 i.e. having income in the range of approx. 0 to 60000 USD

Thus the service provider can invest and target more people with income that lies within that range.



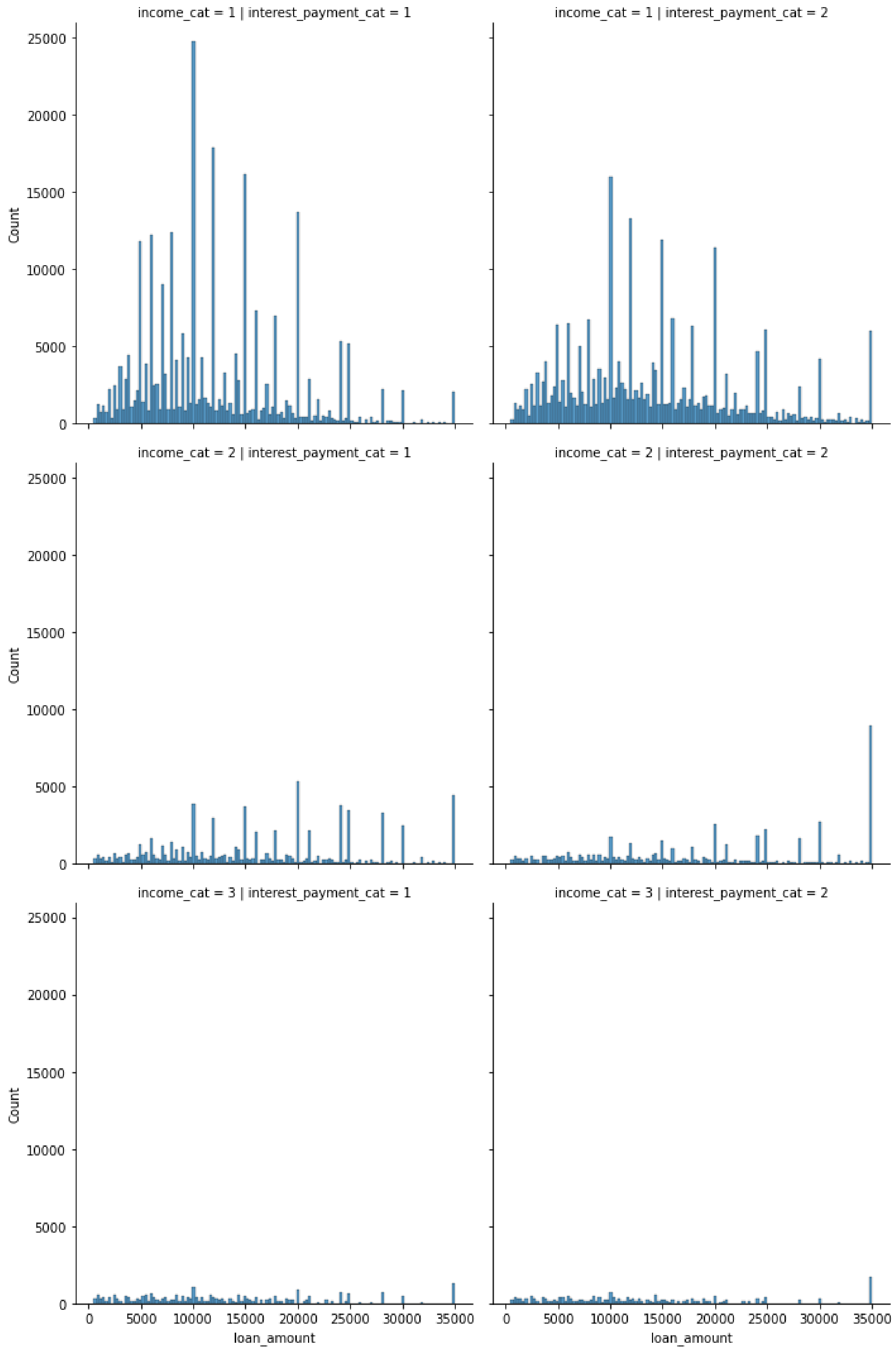
Here is a visualization of duration of loan vs the number of loans.

Clearly people preferred short duration of loans i.e. 36 months instead of long duration that is 60 months which is also pretty evident by the amount they will pay as interest will increase during long period of loan.



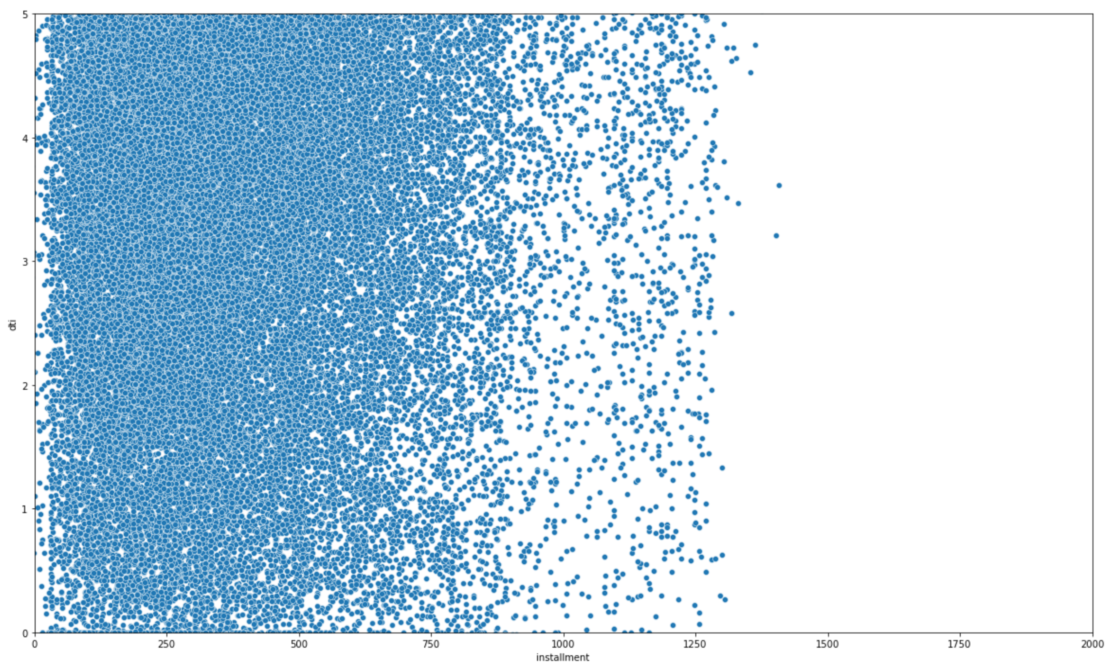
This is pie chart of people applied for loan vs their living area. From this chart we can infer that people from “Munster”, “Leinster” and “Norther-Irl” have applied most for the loan.

Hence people from these areas can be targeted more often than other areas.

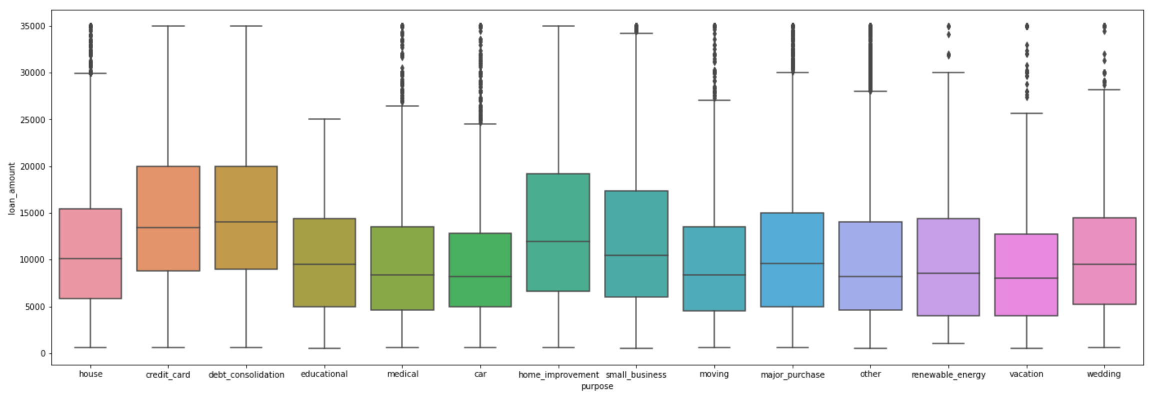

income_cat = 1 | interest_payment_cat = 1
income_cat = 1 | interest_payment_cat = 2
income_cat = 2 | interest_payment_cat = 1
income_cat = 2 | interest_payment_cat = 2
income_cat = 3 | interest_payment_cat = 1
income_cat = 3 | interest_payment_cat = 2
Count
0
5000
10000
15000
20000
25000
30000
35000
loan_amount

Above chart shows the relation between loan amount, income category and interest the borrower is paying on the loan amount. From the graph we can safely draw the following facts:

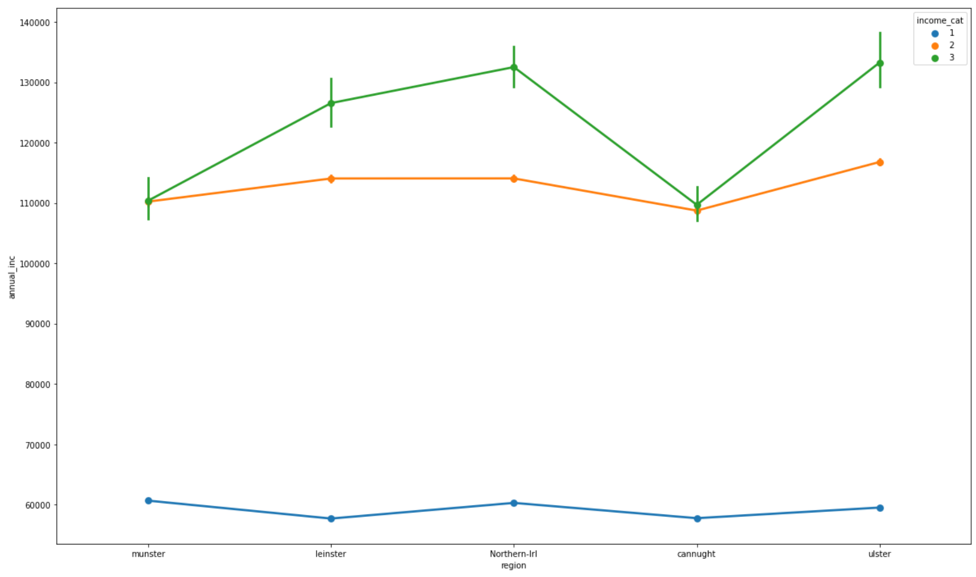
- A huge number of people having income category as 1 have applied for loan.
- People from income category 2 have agreed to pay a higher amount of interest than any other income category person.
- Most of the people have applied loan in the range of 10,000 USD to 20,000 USD. In this range people have mostly applied for 10,000 USD



This chart shows the relation between instalment and dti of the loan borrower. Clearly people preferred to have a instalment of less than 900 USD.



The people have applied for loan in high number for Credit Card, Debt consolidation and house improvement and small business purposes.



Here region vs annual income relation chart has been shown. People from munster and Northern-irl has a greater number of people in category 1 of annual income i.e., more people from this region has applied for the loan

Also, the people from Ulster have a greater number of people from income category 2 and 3. Hence there is lesser chance of people applying for loans.

OneHotEncoding

One hot encoding can be defined as the essential process of converting the categorical data variables to be provided to machine and deep learning algorithms which in turn improve predictions as well as classification accuracy of a model. One Hot Encoding is a common way of preprocessing categorical features for machine learning models.

Here we will use pandas inbuilt function pd.get dummies to convert categorical columns into numerical category columns

```
region_df_train = pd.get_dummies(X_train[['region']])
region_df_test = pd.get_dummies(X_test[['region']])

region_df_train.columns = X_train['region'].unique()
region_df_test.columns = X_test['region'].unique()

X_train = X_train.join(region_df_train,on=X_train.index)
X_test = X_test.join(region_df_test, on=X_test.index)
```

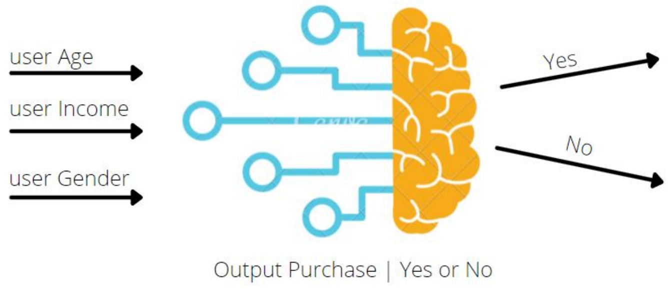
```
X_train.drop('region', axis = 1, inplace = True)
X_test.drop('region', axis = 1, inplace = True)
```

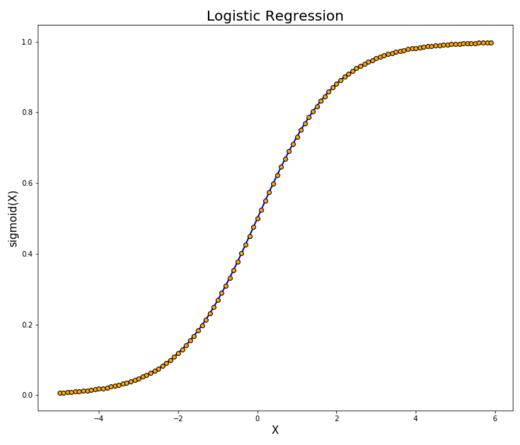
Along with the conversion we will drop the column on which OneHotEncoding has been implemented as categorical columns can not be used in training of machine learning models.

Training of machine learning models

Few of the variety of models that have been used in the training are

1. Logistic Regression



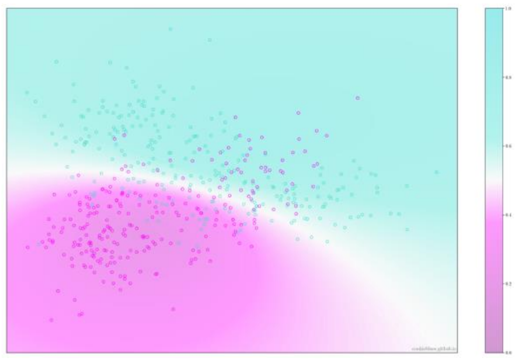


Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

Logit(pi) = 1/(1+ exp(-pi))

ln(pi/(1-pi)) = Beta_0 + Beta_1*X_1 + … + B_k*K_k

2. Gaussian Naïve Bayes



Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Bayes Theorem - It is the determination of the conditional probability of an event. This conditional probability is known as a hypothesis. This hypothesis is calculated through previous evidence or knowledge.

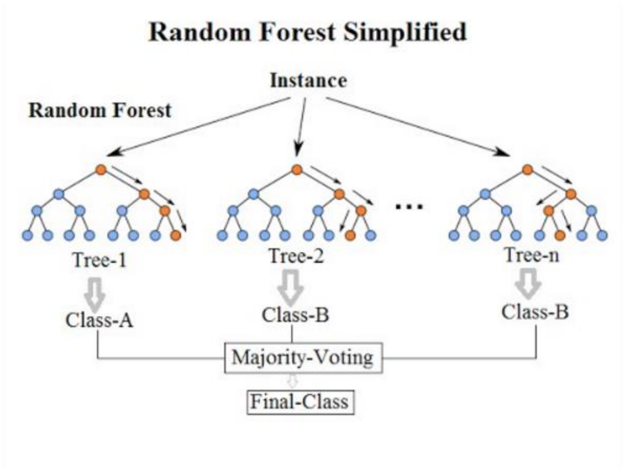
This conditional probability is the probability of the occurrence of an event, given that some other event has already happened.

The formula of Bayes' Theorem involves the posterior probability P(H | E) as the product of the probability of hypothesis P(E | H), multiplied by the probability of the hypothesis P(H) and divided by the probability of the evidence P(E).

Formulae for Bayes theorem

$$p(H \mid E) = \frac{p(E \mid H)\ p(H)}{p(E)}$$

3. Random Forest Classifier

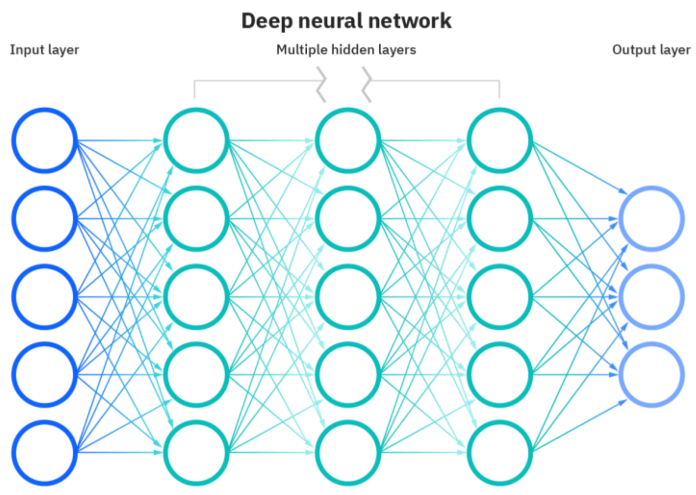


The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.

Steps involved in random forest algorithm:

Step 1: In Random forest n number of random records are taken from the data set having k number of records.
Step 2: Individual decision trees are constructed for each sample.
Step 3: Each decision tree will generate an output.
Step 4: Final output is considered based on Majority Voting

4. Deep Neural Network



Deep neural network represents the type of machine learning when the system uses many layers of nodes to derive high-level functions from input information. It means transforming the data into a more creative and abstract component. In a deep learning neural network every unit in every layer helps in computing result for next layer and after every iteration loss function is calculated and derivative of loss function with respect to particular weight is subtracted by the weight after it is multiplied by the learning rate.

Like this we are able to reach local optimum.

Several algorithms have been designed to optimize deep neural networks like rmsprop and adam which can be applied in any deep learning framework easily.

Evaluation of models

1. Logistic Regression –

```
log_reg = LogisticRegression(max_iter = 1000000)
log_reg.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=1000000)
```

```
scores['Logistic Regression']=log_reg.score(X_test, y_test)
```

```
random_forest = RandomForestClassifier()
random_forest.fit(X_train,y_train)
scores['Random Forest Classifier'] =  random_forest.score(X_test,y_test)
```

2. Gaussian Naïve Bayes –

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
scores['GaussianNB'] = gnb.score(X_test,y_test)
```

3. Random Forest Classifier –

```
random_forest = RandomForestClassifier()
random_forest.fit(X_train,y_train)
scores['Random Forest Classifier'] =  random_forest.score(X_test,y_test)
```

4. Deep Neural Network

```
model = Sequential()
model.add(Dense(100, activation = 'relu',input_shape=(24,)))
model.add(Dense(80,activation = 'relu'))
model.add(Dense(40, activation = 'relu'))
model.add(Dense(5, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
```

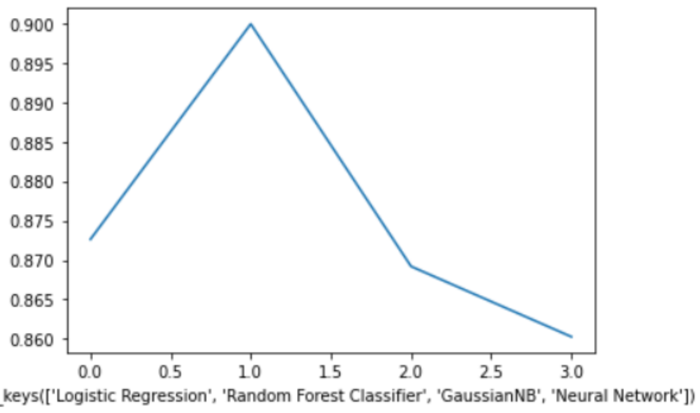
```
opt = keras.optimizers.Adam(learning_rate=0.9)
model.compile(optimizer=opt,
              loss=tf.losses.MeanSquaredError(),
              metrics=['accuracy'])
with tf.device('/gpu:0'):
    model.fit(X_train,y_train, epochs = 10)
```

```
score, acc = model.evaluate(X_test, y_test)
scores['Neural Network'] = acc
```

Plotting accuracies of the models

```
plt.plot(scores.values())
plt.xlabel(scores.keys())
```

```
Text(0.5, 0, "dict_keys(['Logistic Regression', 'Random Forest Classifier',
'GaussianNB', 'Neural Network'])")
```



Here we can observe that **Random Forest Classifier** has the highest accuracy among all the other models with an accuracy of approximately **90%**