

TECHNOCOLABS MACHINE LEARNING APPLICATION DEVELOPER INTERNSHIP

PROJECT REPORT

TITLE: Identify Bad Loans using LendingClub Loan Dataset

AIM:

The principal focus of our project is to train a model using different machine learning algorithms and then deploy it to identify “bad loans” using LendingClub loan dataset in order to analyse whether or not a borrower will pay back their loan.

ABSTRACT:

LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub is the world's largest peer-to-peer lending platform.

On the LendingClub platform, people invest on other people's loans through an online secured system.

INTRODUCTION:

On these types of platforms, in most cases, the main criteria of giving loans to customers is solely based on their credit scores, so that a customer with lower credit score (more risky) gets a higher rate and

customers with higher credit score (less risky) get lower interest rate for their loan. From the investor point of view, the loans with higher interest rate are more attractive due to their higher return of investment. However, it also has a high risk of being not returned at all.

So, investing on such “Bad loans” may result in losing your asset, which is even worse than losing an opportunity to gain more profit.

The Machine Learning model will predict which of the high interest loans are more likely to be returned. This would bring added value by minimizing the associated risks. Also, using other factors along with credit score may help us to identify the high risky loans and minimize the investors loss of money more accurately.

OVERVIEW

- Data Segmentation and Data Cleaning
- Exploratory Data Analysis using python's data visualisation libraries
- Training the model based on the data available
- Deployment of the trained model

DATASET

The dataset we have examined has loan information and the goal is to find a better prediction model to prevent investing in "bad loans" which is based on the Lending club dataset which consists of the amount requested, risk score, debt-to-income ratio, employment length and target.

The original format of the dataset: XLSX

A brief explanation of every column in the dataset is as follows:

- **Amount Requested** :The amount the borrower promises to return, as outlined in the loan contract. The loan amount may surpass the initial amount demanded by the borrower if he or she chooses to add points and other upfront costs to the loan.
- **Risk Score** : A risk score is a mathematical score that assesses a person's creditworthiness based on their credit history. Lenders usually practice this number to estimate the possibility of debt repayment on the consumer's part. It extends from 300 to 850, and rationally the higher is the score, the higher is his/her financial trustworthiness. The credit score performs a crucial role in the lender's judgment to grant credit. Borrowers with credit scores beneath 700 are estimated as subprime borrowers. Lower credit scores can be among the foremost factors influencing your loan interest rates.
- **Debt To Income Ratio**: A debt-to-income ratio is an individual finance measure that analyses the amount of debt you have to your overall earnings. Lenders, practice it as a way to estimate your capacity to handle the payments you make every month and compensate for the money you have borrowed. A low debt-to-income ratio exhibits good stability between debt and income. In general, the lower the interest, the greater the possibility you will be able to get the loan you want. On the contrary, a high debt-to-income ratio shows that you may have a substantial debt for the income you have, and lenders see this as a signal that you would be inadequate to take on any additional responsibilities.
- **Employment Length** : A borrower who has less than a two-year employment history with their current employer

will only be able to use his/her hourly or monthly base salary to pass for a loan.

- **Target:** It indicates whether the person is fit for lending a loan or not. Here 1 means that a person is fit for lending a loan and 0 means it is a bad loan.

DATA CLEANING

- Using pandas data frame, we have calculated the mean of every column.
- By using the fillna we have filled all the cells with empty values.
- We have manually replaced the zeros in a column with the mean of the column.
- The original format of the file was XLSX. We have converted into CSV format and proceeded.

Overall the dataset from Lending club was relatively clean and required minimal updates to prepare it for modeling.

Missing Data

Replaced all missing values with NaN.

```
#1a.> First Lets mark all missing etc values as NaN
#Pandas automatically marks BLANK/NA values as NaN but fails to do so on 'n.a,n/a etc'
#So we need to mark all those obscure values as one generic 'NaN'.

#We pass all 'Missing Value Formats' in read.csv() method as a list to allow the Pandas to mark them as 'NaN'

#a list with all missing value formats

#missing_value_formats=["n.a.", "?", "NA", "n/a", "na", "--"]
#print one column
print(df['settlement_status'].head(10))
#df=pd.read_csv("accepted_2007_to_2018Q4.csv",na_values=missing_value_formats)

#print same column again and check if any values except NaN is present
print(df['settlement_status'])
```

Replacing Data

Replacing NaNs with a single constant value. We will use fillna() to replace missing values.

```
new_df['loan_amnt'].fillna(0,inplace=True)
new_df['funded_amnt'].fillna(0,inplace=True)
new_df['funded_amnt_inv'].fillna(0,inplace=True)
new_df['term'].fillna(0,inplace=True)
new_df['settlement_status'].fillna('No_Settlement_Status',inplace=True)
# To check changes
print(new_df['loan_amnt'].head(10))
print(new_df.isnull().sum())
```

Loan Status

Since loan status is independent variable throughout this project its kept binomial variable as follows:

- Fully paid :0
- Charged Off:1

```
# ONE HOT ENCODING - OHE
```

```
# creating initial dataframe
```

```
loan_status = ('Fully Paid', 'Charged Off')
```

```
loan_status_df = pd.DataFrame(loan_status, columns=['Laon_Status'])
```

```
# generate binary values using get_dummies
```

```
dum_df = pd.get_dummies(loan_status_df, columns=["Laon_Status"], prefix=["Type_is"] )
```

```
# merge with main df bridge_df on key values
```

```
loan_status_df = loan_status_df.join(dum_df)
```

```
loan_status_df
```

	Laon_Status	Type_is_Charged Off	Type_is_Fully Paid
0	Fully Paid	0	1
1	Charged Off	1	0

Overview of Each Feature:

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2260701 entries, 0 to 2260700
```

```
Columns: 151 entries, id to settlement_term
```

```
dtypes: float64(113), object(38)
```

```
memory usage: 2.5+ GB
```

Exploratory Data Analysis

Getting a snap of our data:

```
data.head()
```

	Unnamed: 0	Amount Requested	Risk_Score	Debt-To-Income Ratio	Employment Length	Target
0	0	3600.0	677.0	5.91	10	1
1	1	24700.0	717.0	16.06	10	1
2	2	20000.0	697.0	10.78	10	1
3	3	10400.0	697.0	25.37	3	1
4	4	11950.0	692.0	10.20	4	1

Then we changed the column name to make working easy: -

Before

```
[ ] data.columns
```

```
Index(['Unnamed: 0', 'Amount Requested', 'Risk_Score', 'Debt-To-Income Ratio',  
      'Employment Length', 'Target'],  
      dtype='object')
```

After (And we also removed the unnamed column)

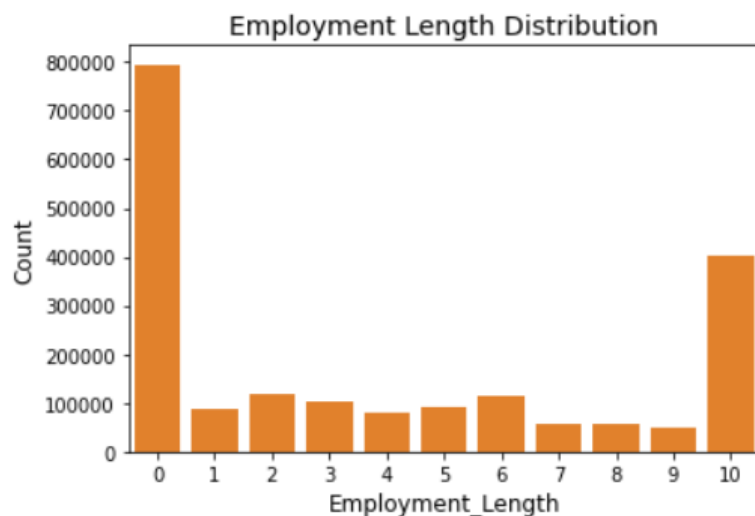
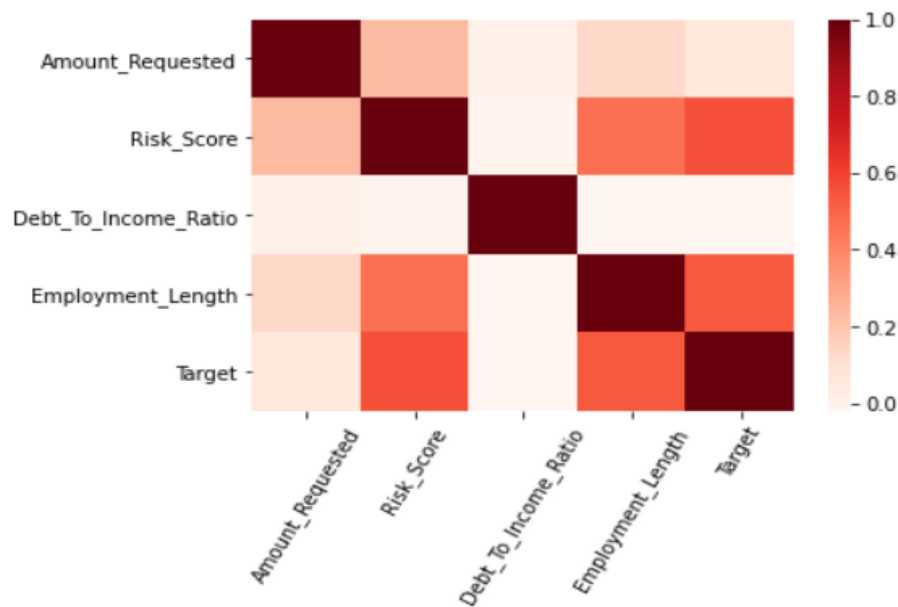
```
data.columns = data.columns.str.strip().str.lower().str.replace(' ', '_').str.replace('(', '').str.replace(')', '').str.replace('-', '_')  
data.columns
```

```
Index(['unnamed_0', 'amount_requested', 'risk_score', 'debt_to_income_ratio',  
      'employment_length', 'target'],  
      dtype='object')
```

Getting a snap of our processed data:

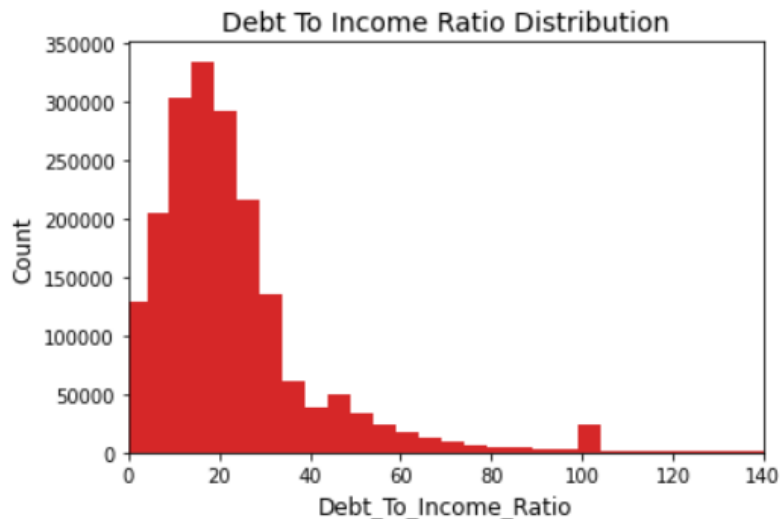
	amount_requested	risk_score	debt_to_income_ratio	employment_length	target
30	7000.0	732.0	13.06	10	1
31	21000.0	682.0	14.47	0	1
32	27500.0	702.0	6.79	10	1
33	7200.0	727.0	8.47	10	1
34	20000.0	692.0	12.45	6	1
35	20000.0	682.0	22.21	5	1
36	10000.0	687.0	35.70	2	0
37	20000.0	727.0	23.45	5	1
38	8650.0	752.0	7.28	0	1
39	21000.0	672.0	12.14	2	1

The plot of the correlation matrix for the dataset



The above plot shows that the employment length feature of the dataset range from 0 to 10:

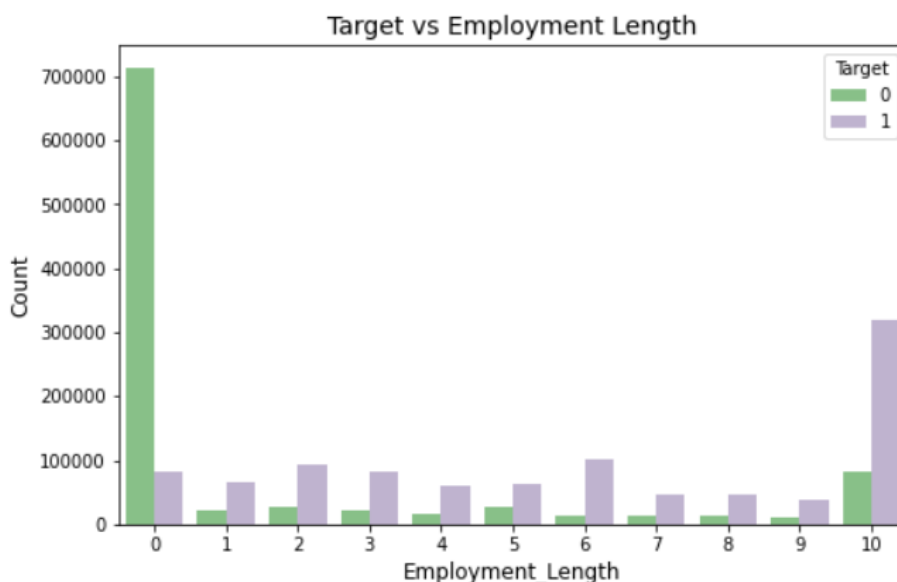
- The maximum frequency is given by employment length of 0 with a count of about 7 lakh.
- Employment length of 10 gives the second maximum count of about half the max (nearly 4lakh).
- Rest Employment length values(1-9) have counted in the range of 0-150000



```
df.query("Debt_To_Income_Ratio < 140").size / df.size
```

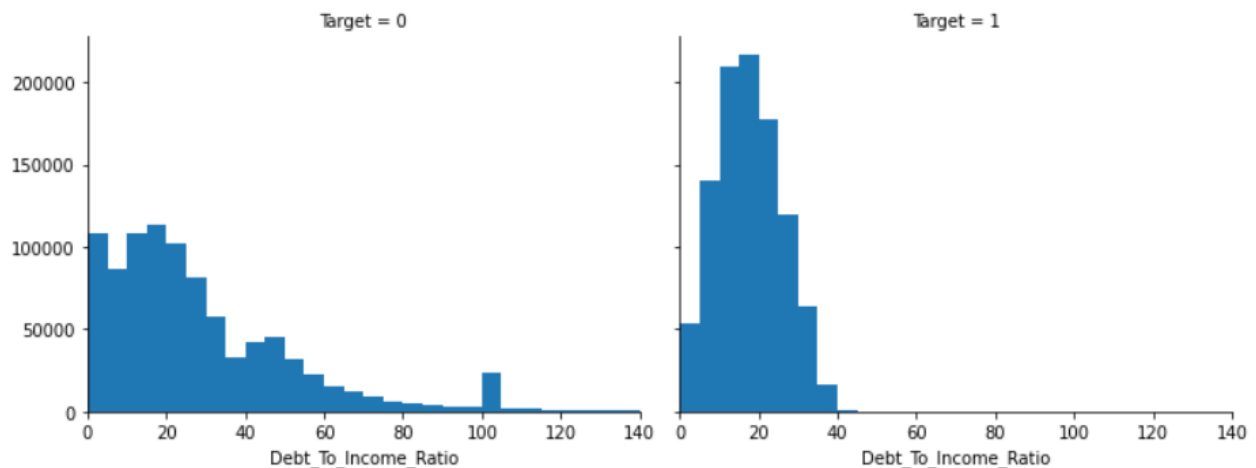
0.9832339411857139

- The above plot is for the Debt to income ratio and ranges from 0 to 140.
- In the above range, 98% of the dataset has been focused to get better visuals.
- Follows a right-skewed curve with a maximum of less than 350000 at the value of 20.



The above plot represents the relationship of Target with Employment length :

- There is just one case where the count of Target equals 0 exceeds that of target equals 1 when Employment length is equal to 0 and also with a great margin.
- For employment length ranging from 1 to 10, there are more no of the target of 1 as compared to the target of 0.

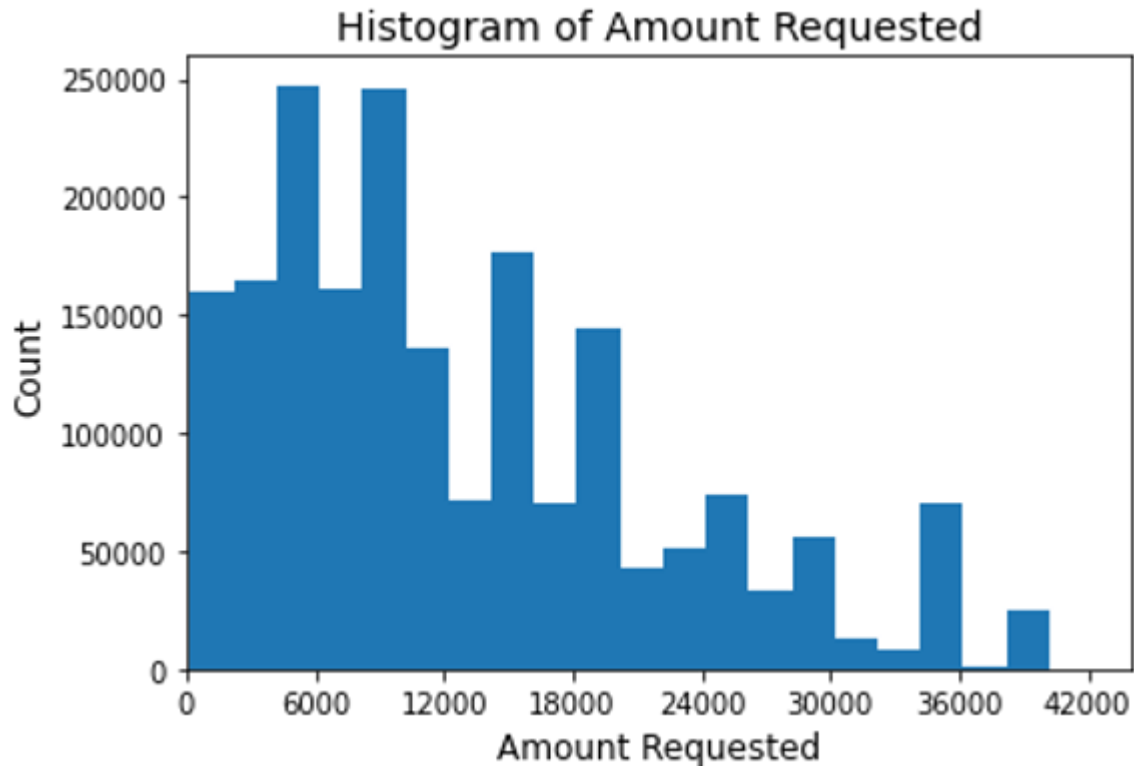


The above plot is a Facet Grid plot of Target vs Debt to Income Ratio:

- For target 0, the Debt to income ratio follows a right-skewed curve
- For target 1, the Debt to income ratio follows a curve normalized near 20.

Amount Requested(amount_requested):-

The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

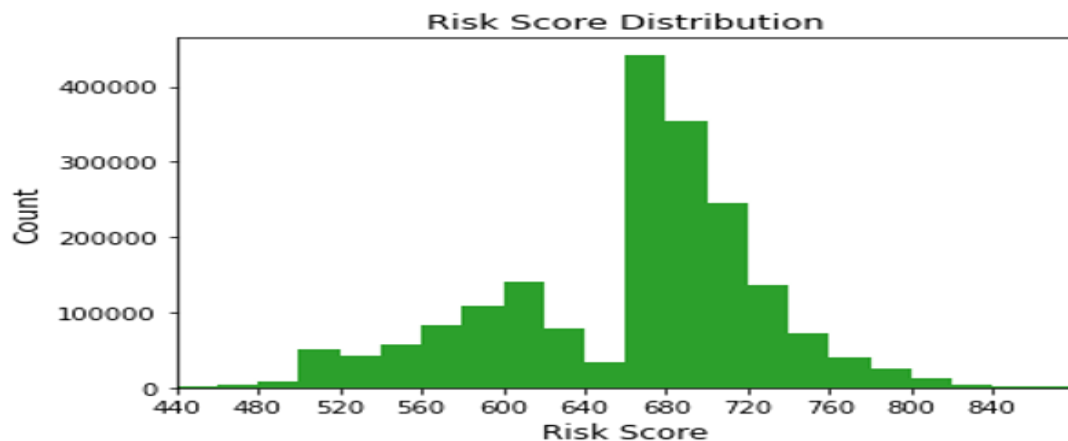


The above plot represents the different range of the values which are being requested by the customers. Observing the histogram, we can see that we have a nicely distributed graph which is very nice to have as we don't know our future requirements that what type of amount or big or small amount may be requested by our customer so our machine learning model will be getting nice exposure of the scenarios that is the amount to given in this case because of which our model will be able to predict the with greater precision that the amount requested by the person is likely to be paid off or not. The data consists of the amount starting from 200 and ranging up to 1400000 and seeing the nature of the histogram we can conclude that our model will be performing quite well in the densely populated areas of the values which is from 200 to 18000 because we almost have more values in that region.

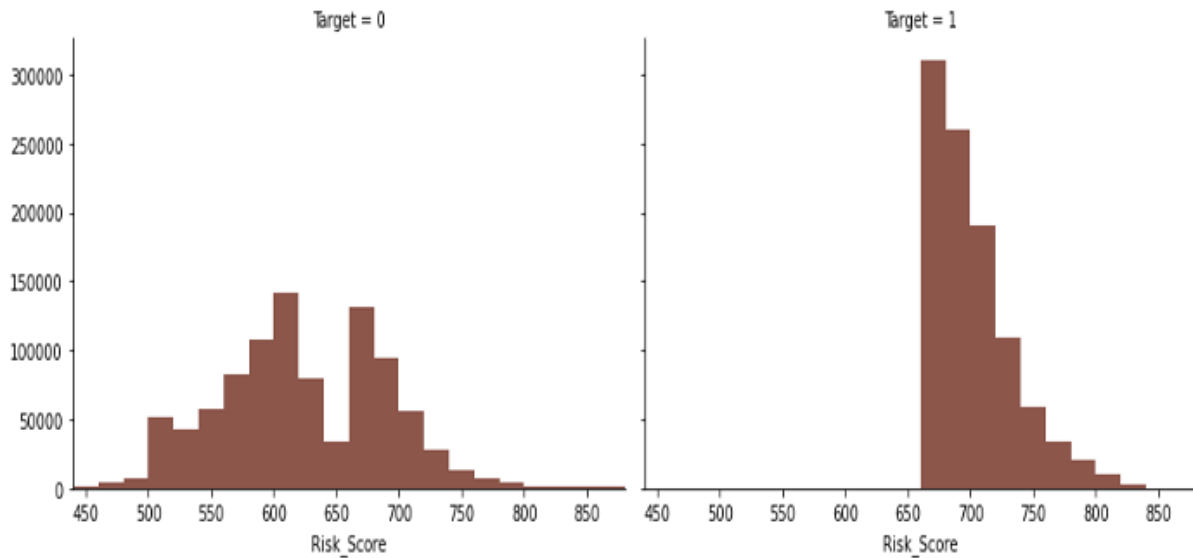


Due to the huge amount of distribution, it is not possible to understand the graph if we plot it for each value of the amount so for making the relation between amount requested and target variable, we have taken the mean of the amount which taken and paid back v/s the values which are not paid in time so we segregate the values and took mean and tried to plot the graph against target. From the above graph it is clear that we have fairly equal entries for both the conditions. Our model will be efficiently making some relation between both the features.

Risk Score(risk_score):-



The above plot is the distribution of another important feature which is risk score. Risk score is the probability of getting back the loan. This feature plays a very important role in the prediction and it is the combination of many features. From the above graph we can observe that the distribution of the values is very fair and we have enough values in each range so our model will be trained with more precision and accuracy however there are more values in region from 600 to 760 so our model prediction will be more accurate in this region and the remaining regions are having very less amount of the data distribution so our model might get some errors in predicting values dropping in these regions. However, there can be another case as well of overfitting of our model and may learn instead of evaluate these regions and may act anonymously in making predictions as if it learns the values or maybe the case never happens and we can control that by changing the weights of the model. This maybe not be the grate problem.



The above two graphs are the histogram of the distribution of risk score over the target variable which we are going to predict at the end with the help of our model. Here 0 means that we have not gotten back the amount which we have landed to the person as a loan, and 1 means that this person has returned the loan and this range is probably safe to be lending. Observing the above graph we can see clearly that the people in range or above 650 are likely to pay back the loan and the people having the risk score in range 400 to 700 are likely not to be paid back.

However, this single correlation is not sufficient in answering the question of who is going to pay back as we can observe there is a great conflict in the data falling between 600 to 700 there are a remarkable number of people in both the categories so this area is highly contractionary and confusing. Some values of paid ones and not paid ones also fall out of the noticeable range like we can have someone whose risk score is 200 but he might have paid off his dues, Since this depends on future conditions of the person not the present.

For the conclusion we have enough exposure of the values, the range of the data is very vast and our model gets more exposure and will be able to predict nicely when this feature is clubbed with another feature.

MODELLING

For modelling to work user needs to download the below mentioned libraries via

‘pip install “library name”’

Total number of libraries we are using are:

1. OS
2. PANDAS
3. NUMPY
4. MATPLOTLIB
5. SEABORN
6. SKLEARN
 - a. TRAIN_TEST_SPLIT
 - b. MINMAXSCALER
 - c. CLASSIFICATION_REPORT
 - d. ACCURACY_SCORE
 - e. CONFUSION_MATRIX
 - f. CLASSIFICATION_REPORT
 - g. ROC_AUC_SCORE
 - h. ROC_CURVE
 - i. AUC
 - j. MODEL_SELECTION
 - k. GRIDSEARCHCV
 - l. RANDOMFORESTCLASSIFIER
7. TENSORFLOW
 - a. SEQUENTIAL
 - b. DROPOUT
 - c. DENSE
8. XGBOOST
 - a. XGBCLASSIFIER

After importing libraries to make sure that all the libraries are installed we write print **“Import successful”**

```
import os
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.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
print("Imports Successful")
```

Imports Successful

Fig.1. Coding of importing libraries in python

Later we import a dataset file whose format is “.CSV”, which is the best format for applying machine learning models.

Via using Pandas library we import dataset file “combinedByYasin.csv”

```
data = pd.read_csv('/content/drive/MyDrive/LendingClub/combinedByYasin.csv')
data.shape
```

Data.shape display the shape of the data which as m*n, columns vs rows of the .csv file

```
data.columns = data.columns.str.strip().str.lower().str.replace(' ', '_').str.replace('(', '').str.replace(')', '').str.replace('-', '_')
del data['unnamed: 0']
```


data.columns

So our training data and validation data is successfully separated. Lets just quickly check the shape of our data.

```
df = data.copy()

X = df.loc[:, df.columns != 'target']
y = df.target.values

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

Fig.2. this block of code creates test, train and split model.

Now we will define a function to print the score of the model accuracy score, classification code, confusion matrix using libraries in python of sklearn such as accuracy_score, confusion_matrix, classification_report

```
def print_score(true, pred, train=True):
    if train:
        clf_report = pd.DataFrame(classification_report(true, pred, output_dict=True))
        print("Train Result:\n=====")
        print(f"Accuracy Score: {accuracy_score(true, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(true, pred)}\n")
    elif train==False:
        clf_report = pd.DataFrame(classification_report(true, pred, output_dict=True))
        print("Test Result:\n=====")
        print(f"Accuracy Score: {accuracy_score(true, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(true, pred)}\n")
```

Fig.3. defining function for scores of the model.

Now with the previous steps we are ready to build a Neural network model using train-test-split and scale data before training.

```
model1 = Sequential()
model1.add(Dense(units=78,activation='relu'))
model1.add(Dense(units=39,activation='relu'))
model1.add(Dense(units=19,activation='relu'))
model1.add(Dense(units=8,activation='relu'))
model1.add(Dense(units=4,activation='relu'))
model1.add(Dense(units=1,activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fig.4. Initializing model.

By using the above codes we have just initialized the model, which we will save and use it later. With increasing epochs accuracy of the model will be increased.

```
model1.fit(x=X_train,
          y=y_train,
          epochs=15,
          batch_size=32,
          validation_data=(X_test, y_test), verbose=1)
```

Epoch 15/15
51821/51821 [=====] - 135s 3ms/step - loss: 0.3088 - accuracy: 0.8792 - val_loss: 0.2996
- val_accuracy: 0.8837
<tensorflow.python.keras.callbacks.History at 0x7f424e19a810>

Fig.5.6. shows value accuracy of the model.

Above results can be more accurate changing the Learning rate and Batch size and Epoch of the model.

Now as the model is created, we start evaluating the model by using some graphs to display certain values.

Value losses:

we use below code on python compiler to display value loss.

```
losses = pd.DataFrame(model1.history.history)
losses[['loss', 'val_loss']].plot()
```

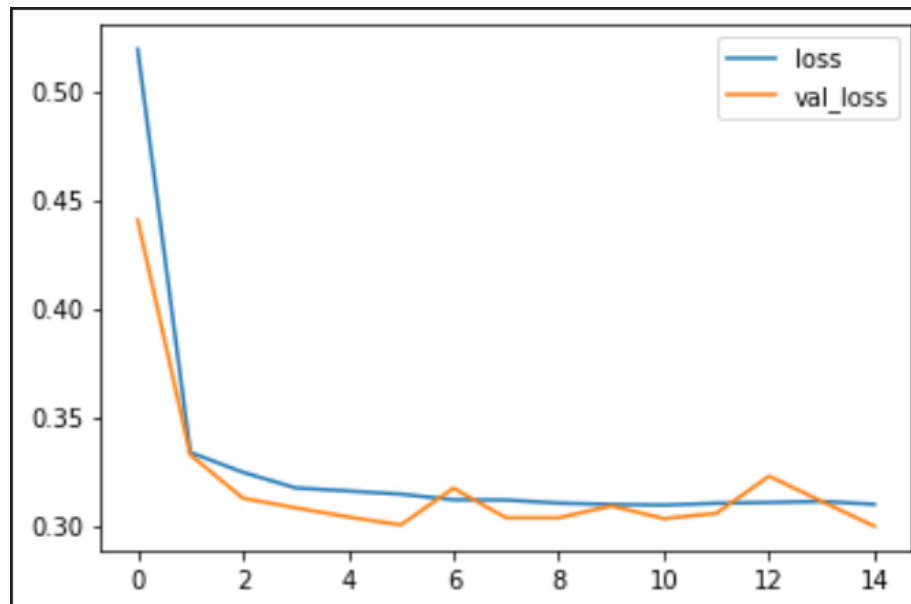


Fig.7. shows value loss

Value Accuracy:

We use the below code to get value accuracy on python

```
losses[['accuracy', 'val_accuracy']].plot()
```

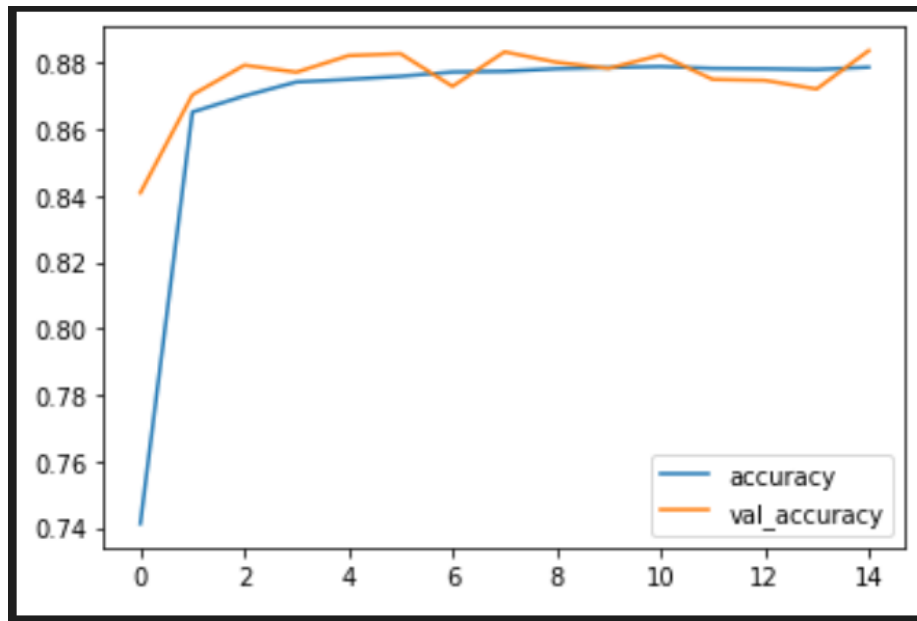


Fig. 8. Shows value accuracy

Now we use prediction lines of code to use predictions.

It shows the accuracy of the code as 90%, macro average of the model is 88%, weighted average of the mode is 88%.

There exists 2 classes for the output, namely 0 no as and 1 as yes, hence for 0 precision is 100%, recall is 77%, f1-score 87% and support is 207499 and for 1 precision is 81%, recall is 100%, f1-score 90% and support is 207063.

```

> ML
predictions = (model1.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, predictions))

```

	precision	recall	f1-score	support
0	1.00	0.77	0.87	207499
1	0.81	1.00	0.90	207063
accuracy			0.88	414562
macro avg	0.90	0.88	0.88	414562
weighted avg	0.90	0.88	0.88	414562

Fig. 9. Shows accuracies, precision, recall, fi-score, support for both the classes.

Now by using the above line

```
model1.save('/content/drive/MyDrive/LendingClub/Saved Model/neural_network.h5')
```

```
#Loading Model
```

```
loaded_model1 = tf.keras.models.load_model('/content/drive/MyDrive/LendingClub/Saved Model/neural_network.h5')
```

we save the model, so that we can use it whenever we need it such as deployment, to run it without taking much time and to kickstart the code, as load using the loaded_model line

Other classification methods:

In the same way how we worked with neural networks, we have used different algorithms such as XGBOOST CLASSIFIER and RANDOM FOREST CLASSIFIER

XGBOOST CLASSIFIER:

XGBoost algorithm is an efficient open source working implementation of “Gradient trees algorithm”, basically used for regression, it’s a tree based combination used to assemble Machine learning algorithms and it stands for Extreme Gradient **Boosting**.

This is the basic structure and code block of XGBoost that we have used in our model.

In order for xgboost to work we need to install libraries such as “xgboost”

Like in Neural networks here also we define classifiers, then we create GridSearchCv with parameters as first we provide classifier address, then params in grid, then we give CV as 3, scoring as roc_auc, n_jobs as -1 finally verbose as 1.

Now we indicate best params out of all

Then we fit the model to the dataset using the fit function which is predefined using x_train and y_train then we indicate outputs via printing the x train prediction and y train prediction. As shown in the below figure:

```

from xgboost import XGBClassifier
n_estimators = [50, 100, 200]
learning_rate = [0.05, 0.01, 0.5, 0.1, 1]
tree_method = ['gpu_hist']
params_grid = { 'n_estimators': n_estimators, #'learning_rate': learning_rate,
}
xgb_clf = XGBClassifier()
xgb_cv = GridSearchCV(xgb_clf, params_grid, cv=3, scoring='roc_auc', n_jobs=-1, verbose=1)
# xgb_cv.fit(X_train, y_train) # best_params = xgb_cv.best_params_ # best_params['tree_method'] =
'gpu_hist'
best_params = {'n_estimators': 50, 'tree_method': 'gpu_hist'}
print(f"Best Parameters: {best_params}")
xgb_clf = XGBClassifier(**best_params)
xgb_clf.fit(X_train, y_train)
y_train_pred = xgb_clf.predict(X_train)
y_test_pred = xgb_clf.predict(X_test)
print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

```

Fig.10. xgboost algorithm

Output of xgboost is:

```

Best Parameters: {'n_estimators': 50, 'tree_method': 'gpu_hist'}
Train Result:
=====
Accuracy Score: 88.78%

CLASSIFICATION REPORT:

```

	0	1	accuracy	macro avg	weighted avg
precision	0.999966	0.816805	0.887819	9.083852e-01	9.083611e-01
recall	0.775605	0.999973	0.887819	8.877892e-01	8.878187e-01
f1-score	0.873610	0.899155	0.887819	8.863828e-01	8.863862e-01
support	828904.000000	829340.000000	0.887819	1.658244e+06	1.658244e+06

```

Confusion Matrix:
[[642902 186002]
 [    22 829318]]

Test Result:
=====
Accuracy Score: 88.79%

CLASSIFICATION REPORT:

```

	0	1	accuracy	macro avg	weighted avg
precision	0.999994	0.816674	0.887877	0.908334	0.908430
recall	0.775994	0.999995	0.887877	0.887995	0.887877
f1-score	0.873868	0.899085	0.887877	0.886476	0.886463
support	207499.000000	207063.000000	0.887877	414562.000000	414562.000000

```

Confusion Matrix:
[[161018 46481]
 [     1 207062]]

```

Fig.11. output of xgboost

Finally accuracy of xgboost is 88.79%

For 0 precision is 99%, recall is 77.59%, f1_score is 87.38%, and support is 207499.

For 1 precision is 81.66%, recall is 99.99%, f1_score is 89.90%, and support is 207063.

From this we observe that the values are almost similar to neural network algorithms.

```
# Saving the model
```

```
xgb_clf.save_model('/content/drive/MyDrive/LendingClub/Saved Model/XGBoost.model')
```

```
#Loading the model
```

```
loaded_xgb = xgb.Booster({'nthread': 4}) # init model
```

```
loaded_xgb.load_model('/content/drive/MyDrive/LendingClub/Saved Model/XGBoost.model') # load data
```

We save the model, so that we can use it whenever we need it such as deployment, to run it without taking much time and to kickstart the code, as load using the loaded_xgb line.

RANDOM FOREST CLASSIFIER:

Random forests are an ensemble learning method for regression, classification and some other tasks which operate via building a multiple of decision trees at outputting and training time, the class that is the mode of the mean/average prediction (regression) of the individual trees or . classes (classification).

This is the basic structure of a random forest classification algorithm.

The algorithm is simpler than the previous 2 classifiers, here we just import randomforestclassifier from sklearn.classifier then declare the model, then fit it to the classifier and predict using xtrain and ytrain.

Overall accuracy score is 85.81%

For 0 precision is 90.21%, recall is 80.37%, f1_score is 85.01%, and support is 207499.

For 1 precision is 82.27%, recall is 91.25%, f1_score is 86.53%, and support is 207063.

Accuracy is 85.81%, macro average of the mode is 86.24% and weighted average of the model is 87.77%

```

from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)
y_train_pred = rf_clf.predict(X_train)
y_test_pred = rf_clf.predict(X_test)
print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

```

Train Result:

=====

Accuracy Score: 99.34%

CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.997390	0.989382	0.993353	9.933862e-01	9.933851e-01
recall	0.989291	0.997412	0.993353	9.933515e-01	9.933526e-01
f1-score	0.993324	0.993381	0.993353	9.933525e-01	9.933525e-01
support	828904.000000	829340.000000	0.993353	1.658244e+06	1.658244e+06

Confusion Matrix:

```
[[820027  8877]
 [ 2146 827194]]
```

Confusion Matrix:

```
[[820027  8877]
 [ 2146 827194]]
```

Test Result:

=====

Accuracy Score: 85.81%

CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.902100	0.822723	0.858123	0.862411	0.862453
recall	0.803773	0.912587	0.858123	0.858180	0.858123
f1-score	0.850102	0.865328	0.858123	0.857715	0.857707
support	207499.000000	207063.000000	0.858123	414562.000000	414562.000000

Confusion Matrix:

```
[[166782  40717]
 [ 18100 188963]]
```

Fig. 12. Random forest classification algorithm code

Now let's compare all the three models using the above code snippet:


```

ML
ml_models = {
    'Random Forest': rf_clf,
    'XGBoost': xgb_clf,
    'ANNs': model1
}
for model in ml_models:
    print(f"{model.upper():{30}} roc_auc_score: {roc_auc_score(y_test, ml_models[model].predict(X_test))
    :.3f}")
RANDOM FOREST          roc_auc_score: 0.858
XGBOOST               roc_auc_score: 0.888
ANNs                  roc_auc_score: 0.887

```

Fig.13. shows the accuracies of all the 3 models.

And lets analysis it via graph:

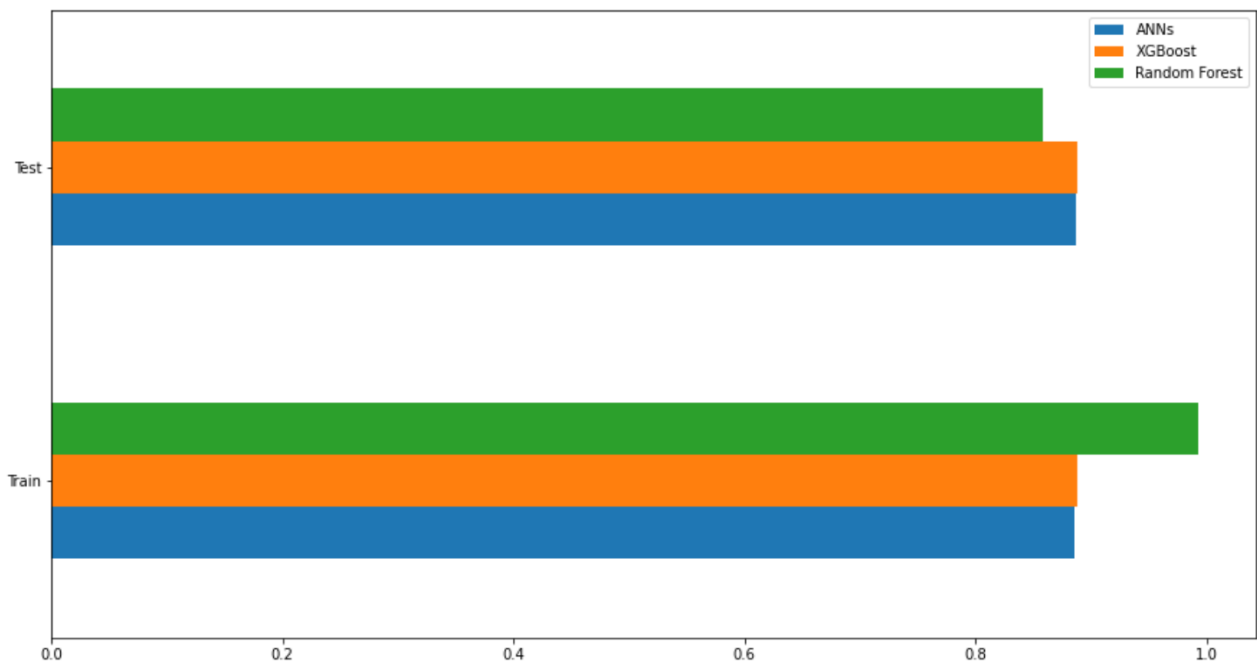


Fig.14. Graph for comparing 3 models.

DEPLOYMENT

After the model is made and evaluated on the basis of accuracy it needs to be deployed on the cloud platform, in doing so we followed some steps which are as follow:

1. Making Pickle file(.pkl)

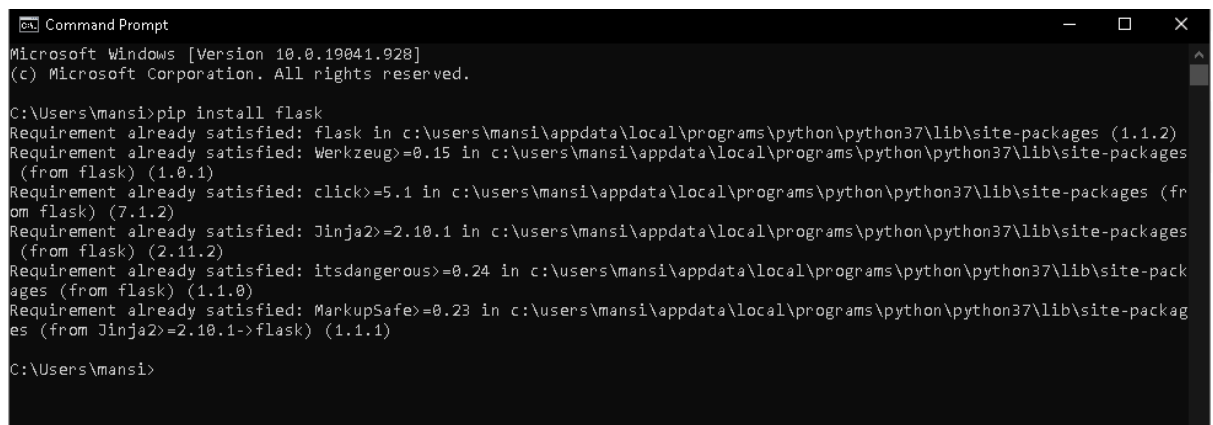
```
[ ] 1 import pickle
    2 pickle.dump(classifier, open('model.pkl','wb'))

[ ] 1 model = pickle.load(open('model.pkl','rb'))
    2 print(model.predict([[3600.0,677.0, 5.91,10]]))
```

2. Installing Flask:

Command:

pip install flask



```
Command Prompt
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

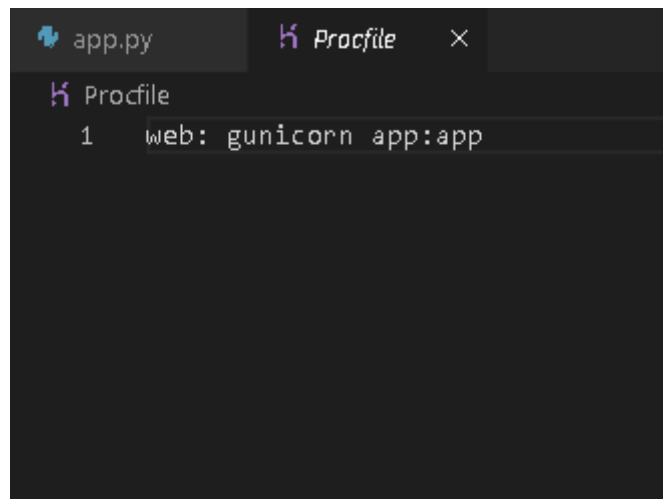
C:\Users\mansi>pip install flask
Requirement already satisfied: flask in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (1.1.2)
Requirement already satisfied: Werkzeug>=0.15 in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (from flask) (1.0.1)
Requirement already satisfied: click>=5.1 in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (from flask) (7.1.2)
Requirement already satisfied: Jinja2>=2.10.1 in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (from flask) (2.11.2)
Requirement already satisfied: itsdangerous>=0.24 in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (from flask) (1.1.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\mansi\appdata\local\programs\python\python37\lib\site-packages (from Jinja2>=2.10.1->flask) (1.1.1)

C:\Users\mansi>
```

3. Creating a folder with model.pkl,HTML file,app.py(Flask).

4. Then making a Procfile and Requirements.txt file .

Procfile:The ProcFile contains the command line for starting your application on heroku.



```
app.py Procfile X
Procfile
1 web: gunicorn app:app
```

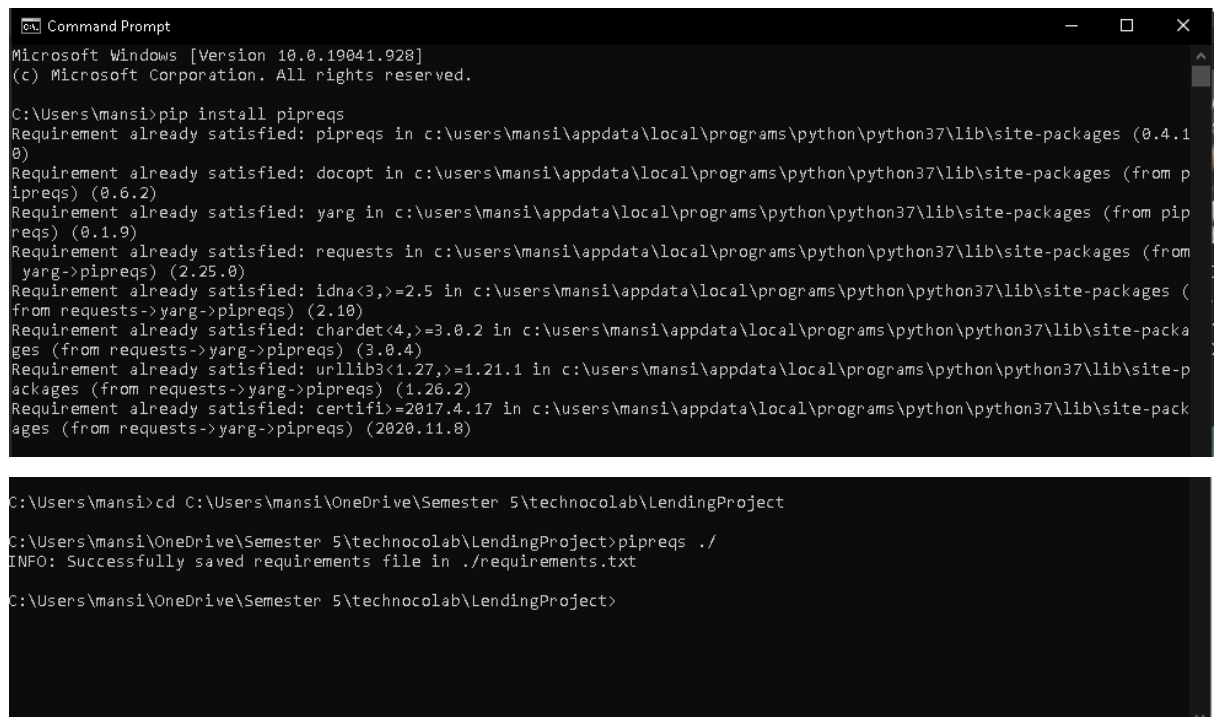
Requirements.txt: These requirements. txt file is used for specifying what python packages are required to run the project you are looking at. Typically the requirements. txt file is located in the root directory of your project

Command:

pip install pipreqs

cd \project\folder\path

pipreqs ./

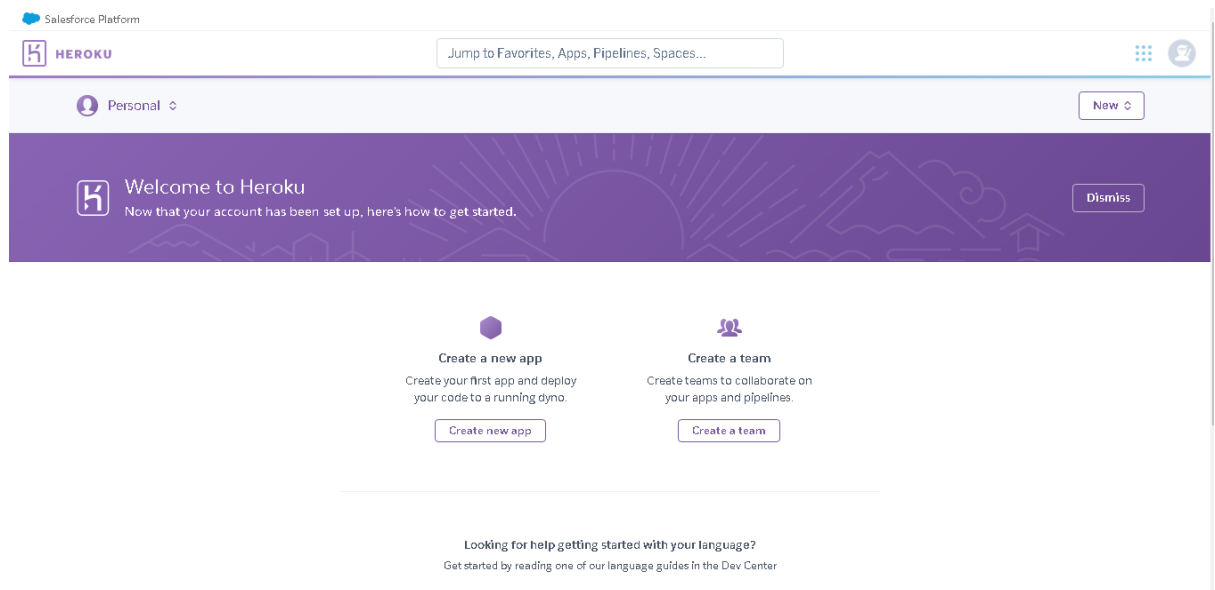


```
Command Prompt
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mans1>pip install pipreqs
Requirement already satisfied: pipreqs in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (0.4.10)
Requirement already satisfied: docopt in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from pipreqs) (0.6.2)
Requirement already satisfied: yarg in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from pipreqs) (0.1.9)
Requirement already satisfied: requests in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from yarg->pipreqs) (2.25.0)
Requirement already satisfied: idna<3,>=2.5 in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from requests->yarg->pipreqs) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from requests->yarg->pipreqs) (3.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from requests->yarg->pipreqs) (1.26.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\mans1\appdata\local\programs\python\python37\lib\site-packages (from requests->yarg->pipreqs) (2020.11.8)

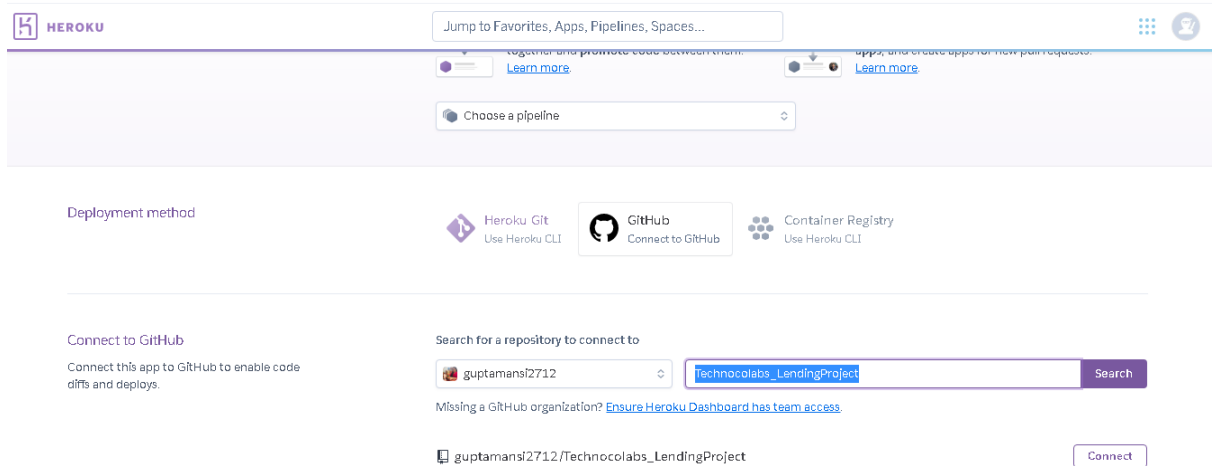
C:\Users\mans1>cd C:\Users\mans1\OneDrive\Semester 5\technocolab\LendingProject
C:\Users\mans1\OneDrive\Semester 5\technocolab\LendingProject>pipreqs ./
INFO: Successfully saved requirements file in ./requirements.txt
C:\Users\mans1\OneDrive\Semester 5\technocolab\LendingProject>
```

5. Creating an account on **heroku**.



6. Uploading folder on GitHub.

Name	Date modified	Type	Size
.git	11-05-2021 12:34 PM	File folder	
static	01-05-2021 07:29 PM	File folder	
templates	01-05-2021 07:29 PM	File folder	
app	06-05-2021 09:02 PM	Python Source File	1 KB
model.pkl	06-05-2021 06:17 PM	PKL File	253 KB
model	06-05-2021 06:17 PM	Python Source File	2 KB
Procfile	29-11-2020 11:26 PM	File	1 KB
requirements	11-05-2021 09:59 AM	Text Document	1 KB

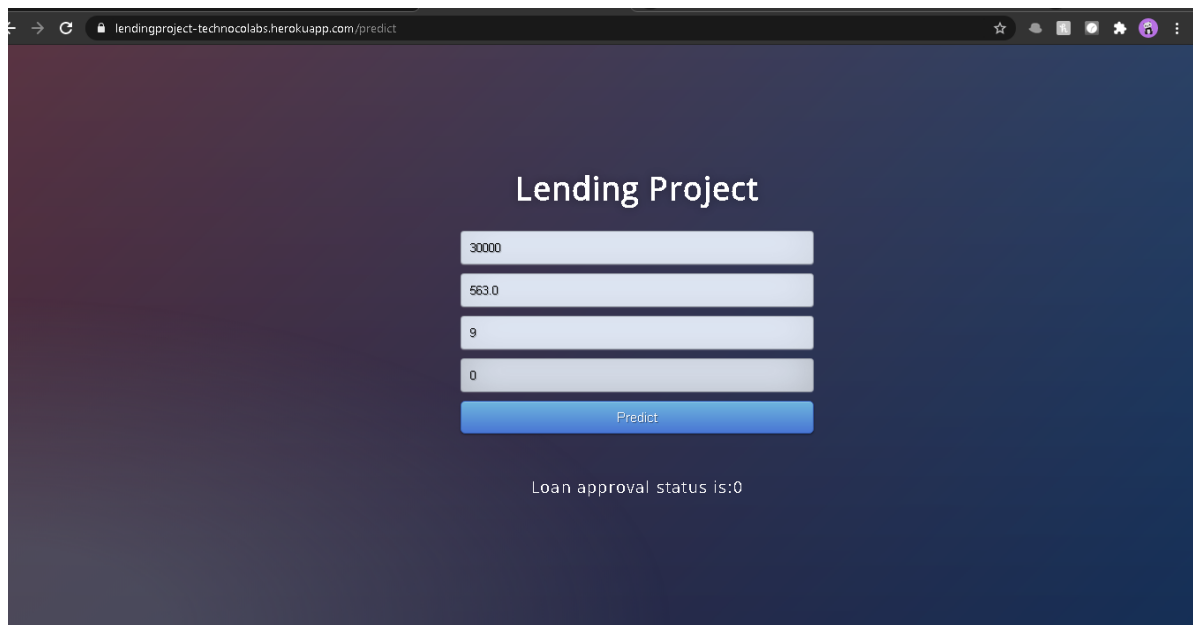


9. Finally, deploy the web app

- Enable Automatic Deploy
- Deploy main branch

Link to the Web application:

<https://lendingproject-technocolabs.herokuapp.com/>



Contributors:

- 1.Somya Jaiswal
- 2.Luv Gupta
- 3.Mansi Gupta
- 4.Aparajita
- 5.Anisha D’cunha
- 6.Shraddha Padhiari
- 7.Nidhi Singh
- 8.Hrithik Sagar Rachakonda