Project Information:

DOMAIN OF PROJECT	Finance
PROJECT TITLE	**Loan Disbursal risk**

Abstract:

- In today's competitive environment it has become very important for the Banks to know whether the customers they are receiving can repay or not.
- Banking processes use manual procedures to determine whether or not a borrower is suitable for a loan based on results.
- But in the case when the applications are more then it becomes very hectic for the Bank employees to identify the customers who will be able to repay or not.
- At that time, making a decision would take a long time. As a result, the loan prediction machine learning model can be used to assess a customer's loan status and build strategies. This model extracts and introduces the essential features of a borrower that influence the customer's loan status.

Objectives:

- To conduct an exploratory analysis to find out latent variables and understand which factors group together? Explore the significant variables.
- Identify the factors which influences the customer repayment. Perform classification algorithms to classify a customer being defaulter for loan or not.
- Perform customer segmentation based on the features provided and identify the insights. Based on that we will decide whether he should be offered with Bank services or not.

Industry Review:

- Nowadays there are many risks related to bank loans, especially for the banks so as to reduce their capital loss. The analysis of risks and assessment of default becomes crucial thereafter. Banks hold huge volumes of customer behavior related data from which they are unable to arrive at a judgment if an applicant can be defaulter or not.
- Data Mining is a promising area of data analysis which aims to extract useful knowledge from tremendous amount of complex data sets. In this project we aim to design a model that would predict whether the loan should be granted or not. The model is a decision treebased classification model.
- Prior to building the model, the dataset is pre-processed, reduced and made ready to provide efficient predictions. The final model is used for prediction with the test dataset.

- Credit Risk assessment is a crucial issue faced by Banks nowadays which helps them to evaluate if a loan applicant can be a defaulter at a later stage so that they can go ahead and grant the loan or not. This helps the banks to minimize the possible losses and can increase the volume of credits. The result of this credit risk assessment will be the prediction of Probability of Default (PD) of an applicant. Hence, it becomes important to build a model that will consider the various aspects of the applicant and produces an assessment of the Probability of Default of the applicant. This parameter PD, help the bank to make decision if they can offer the loan to the applicant or not.
- The aim of this work is to propose a data mining framework using python for predicting PD for the new loan applicants of a Bank. The data used for analysis contains many inconsistencies like missing values, outliers and inconsistencies and they have to be handled before being used to build the model. Only few of the customer parameters really contribute to the prediction of the defaulter. So, those parameters or features need to be identified before a model is applied. To classify if the applicant is a defaulter or not, the best data mining approach is the classification modelling using Decision Tree. The above said steps are integrated into a single model and prediction is done based on this model.

Current practices in Industry:

- In the improvement of banking sector many people are applied for bank loans but in bank has its limited benefit for given of limited people only, now searching out of which the loan can be given it will be a safe option for the bank is a classic process. So, in this project we try to reduce the risk factor behind collecting the safe people so to save lots of bank attempt or benefits.
- On the basis of this data/familiar, by using the machine learning model which give the most precise result. The main purpose of this project to forecast either imputing the loan given that person will be safe or not. In this, we are forecast loan data by using some machine learning algorithm like Decision Tree, Logical Regression and Classification.

Problem Statement:

- Our aim is to devise a model which will predict that the customer to whom we are providing loan will repay the loan amount on time or will default on it.

- This will help the company to identify the factors which makes a customer riskier for them and thus no loan should be given to them.

- Identify the factors which influences the customer repayment. Perform classification algorithms to classify a customer being defaulter for loan or not.

- Perform customer segmentation based on the features provided and identify the insights. Based on that we will decide whether he should be offered with Bank services or not.

Project Outcome:

- By implementing the resultant models built using the above methods, The solution to the problem is that we find out the data from the different types of surveys and then depending upon that find out the factors which are related to the customer turning out to be riskier or not.

Dataset and Domain:

Data Dictionary:

- The dataset has 2,52,000 records and 12 attributes
- The attribute/feature/column names are given below:

Features	Description	Data-Type
Income	Income of the user	Int64
age	Age of the user	int64
experience	Professional experience of the user in years	int64
married	Whether married or single	Object
house_ownership	Owned or rented or neither	Object
car_ownership	Does the person own a car	Object

Profession	profession of person	Object
CITY	City of residence	Object
STATE	State of residence	Object
Current_job_years	Years of experience in the current job	Int64
Current_house_years	Number of years in the current residence	Int64
Risk_Flag	Defaulted on a loan	Object

Variable categorization:

- There are 6 numerical columns and 06 categorical columns

Numerical Columns:

- ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS', 'Risk_Flag']

Categorical Columns:

- ['Married/Single', 'house_ownership', 'car_ownership', 'Profession', 'CITY', 'STATE'

```
In [4]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252000 entries, 1 to 252000
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Income             252000 non-null  int64
 1   Age                252000 non-null  int64
 2   Experience         252000 non-null  int64
 3   Married/Single     252000 non-null  object
 4   House_Ownership    252000 non-null  object
 5   Car_Ownership      252000 non-null  object
 6   Profession         252000 non-null  object
 7   CITY               252000 non-null  object
 8   STATE              252000 non-null  object
 9   CURRENT_JOB_YRS    252000 non-null  int64
 10  CURRENT_HOUSE_YRS  252000 non-null  int64
 11  Risk_Flag          252000 non-null  int64
dtypes: int64(6), object(6)
memory usage: 25.0+ MB
```

Data Pre-Processing:

count of missing/ null values, redundant columns

- The dataset has 0 missing values.

Check for Null Values

```
In [8]: df.isnull().sum()

Out[8]: Income             0
        Age                0
        Experience         0
        Married/Single     0
        House_Ownership    0
        Car_Ownership      0
        Profession         0
        CITY               0
        STATE              0
        CURRENT_JOB_YRS    0
        CURRENT_HOUSE_YRS  0
        Risk_Flag          0
        dtype: int64
```

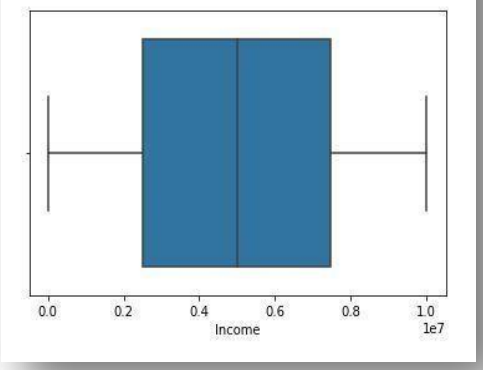
There are no null value in the dataset

Univariate Analysis:

Analysis of Numerical Variables:

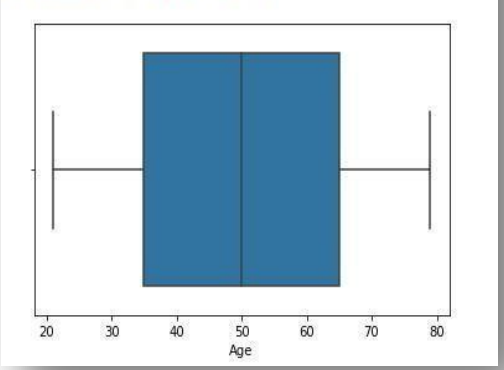
- From the below plots, each of the numerical variables are analyzed individually.

```
sns.boxplot(df['Income'])
<AxesSubplot:xlabel='Income'>
```



The variable does not have any outliers

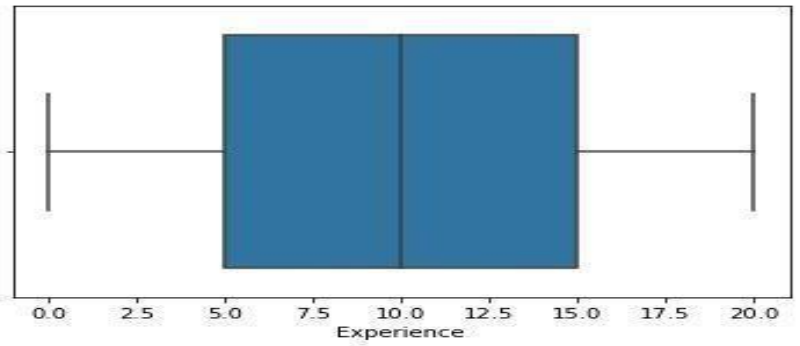
```
sns.boxplot(df['Age'])
<AxesSubplot:xlabel='Age'>
```



The mean age is 50Years.

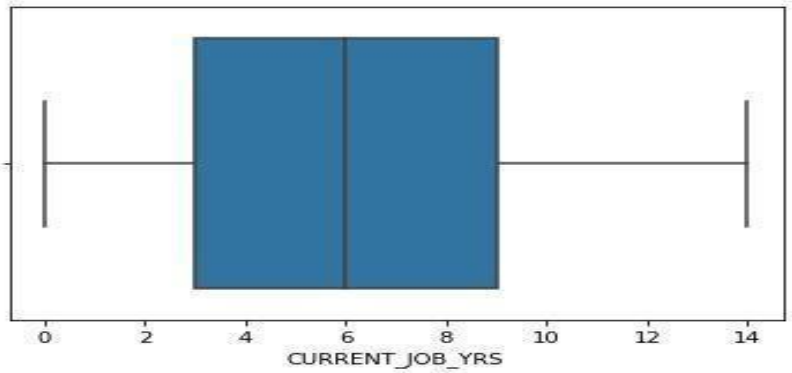
```
sns.boxplot(df['Experience'])
```

```
<AxesSubplot:xlabel='Experience'>
```



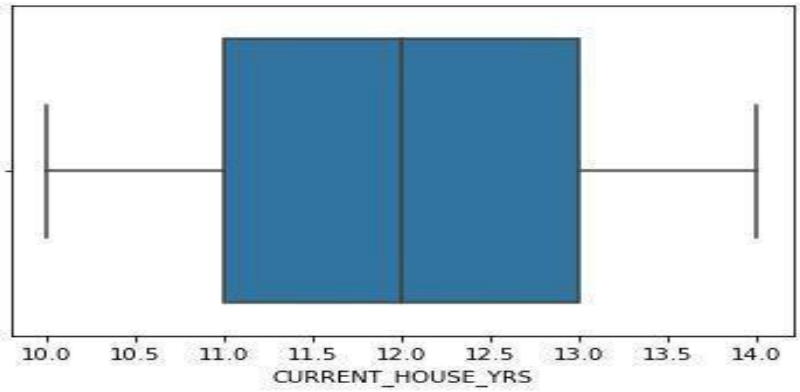

```
sns.boxplot(df['CURRENT_JOB_YRS'])
```

```
<AxesSubplot:xlabel='CURRENT_JOB_YRS'>
```




```
sns.boxplot(df['CURRENT_HOUSE_YRS'])
```

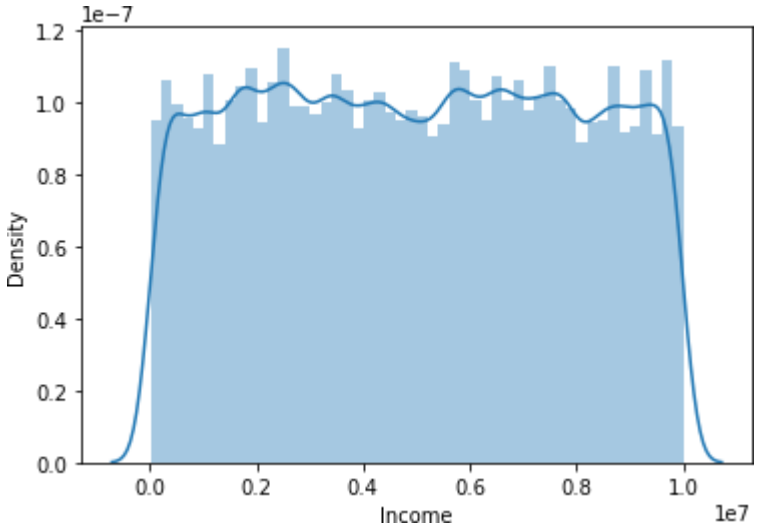
```
<AxesSubplot:xlabel='CURRENT_HOUSE_YRS'>
```



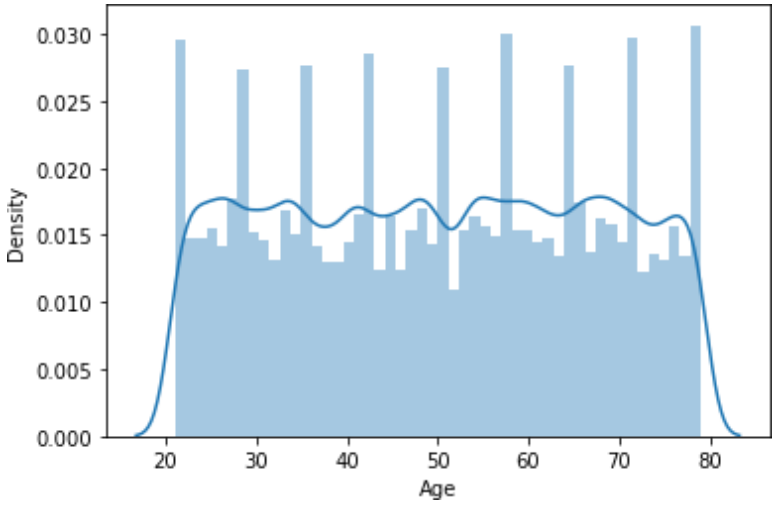
Distplot:

- A Distplot or distribution plot, depicts the variation in the data distribution. Seaborn Distplot represents the overall distribution of continuous data variables.
- The Seaborn module along with the Matplotlib module is used to depict the distplot with different variations in it. The Distplot depicts the data by a histogram and a line in combination to it.

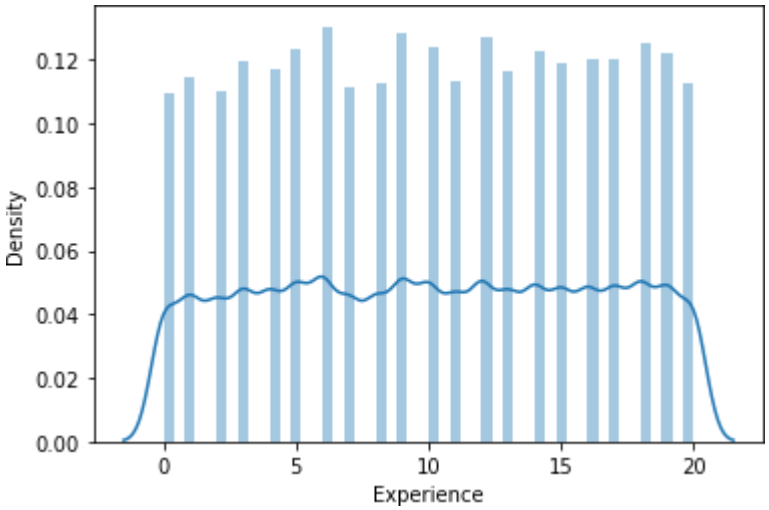
1. Income



2. Age



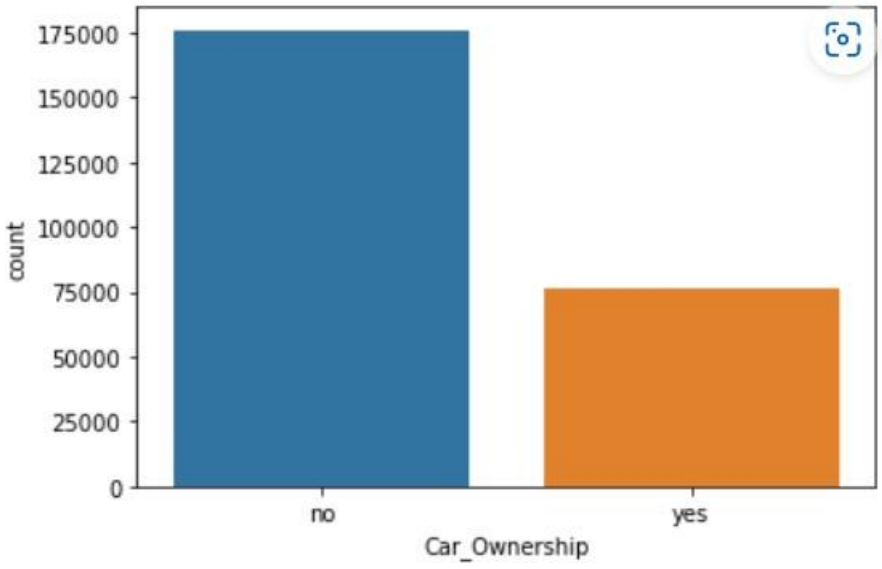
3. Experience



- From above three graph we can observe that in age income and experience columns there is no skewness and this is platykurtic distribution that infer the presence of extreme values.

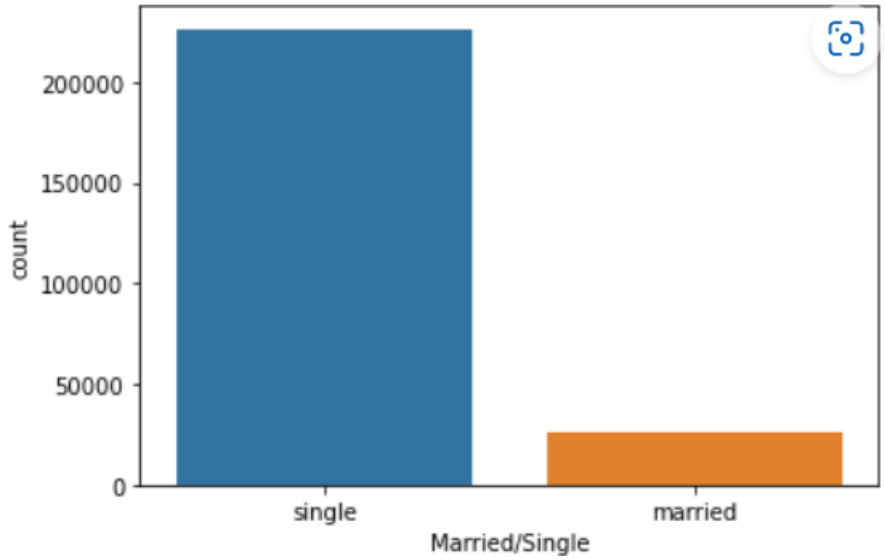
For Categorical values:

1. Car Ownership



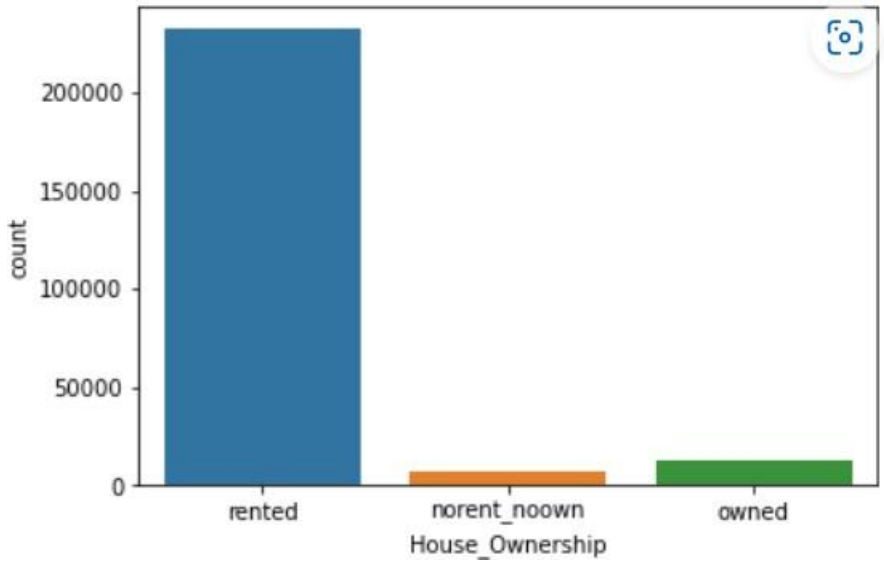
- From the above pie chart for categorical variable “Car_ownership”, we can observe that 69.84% of people don’t have car and only 30.16% of people own a car.

2. Married/ Single



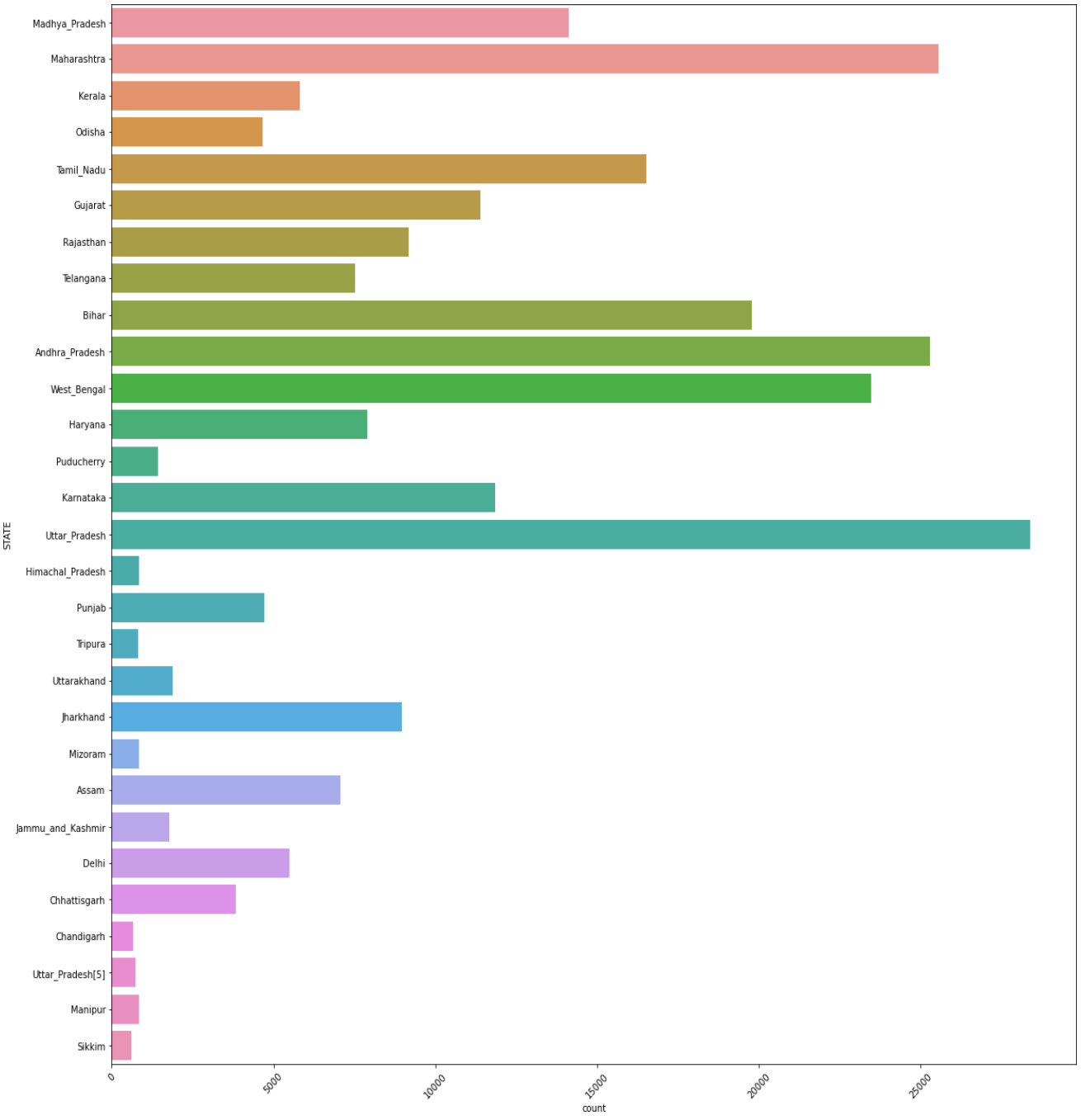
- From above pie chart for categorical variable “Married/Single”, we can observe that 89.79% of people are single and 10.21% of people are married.

3. House Ownership



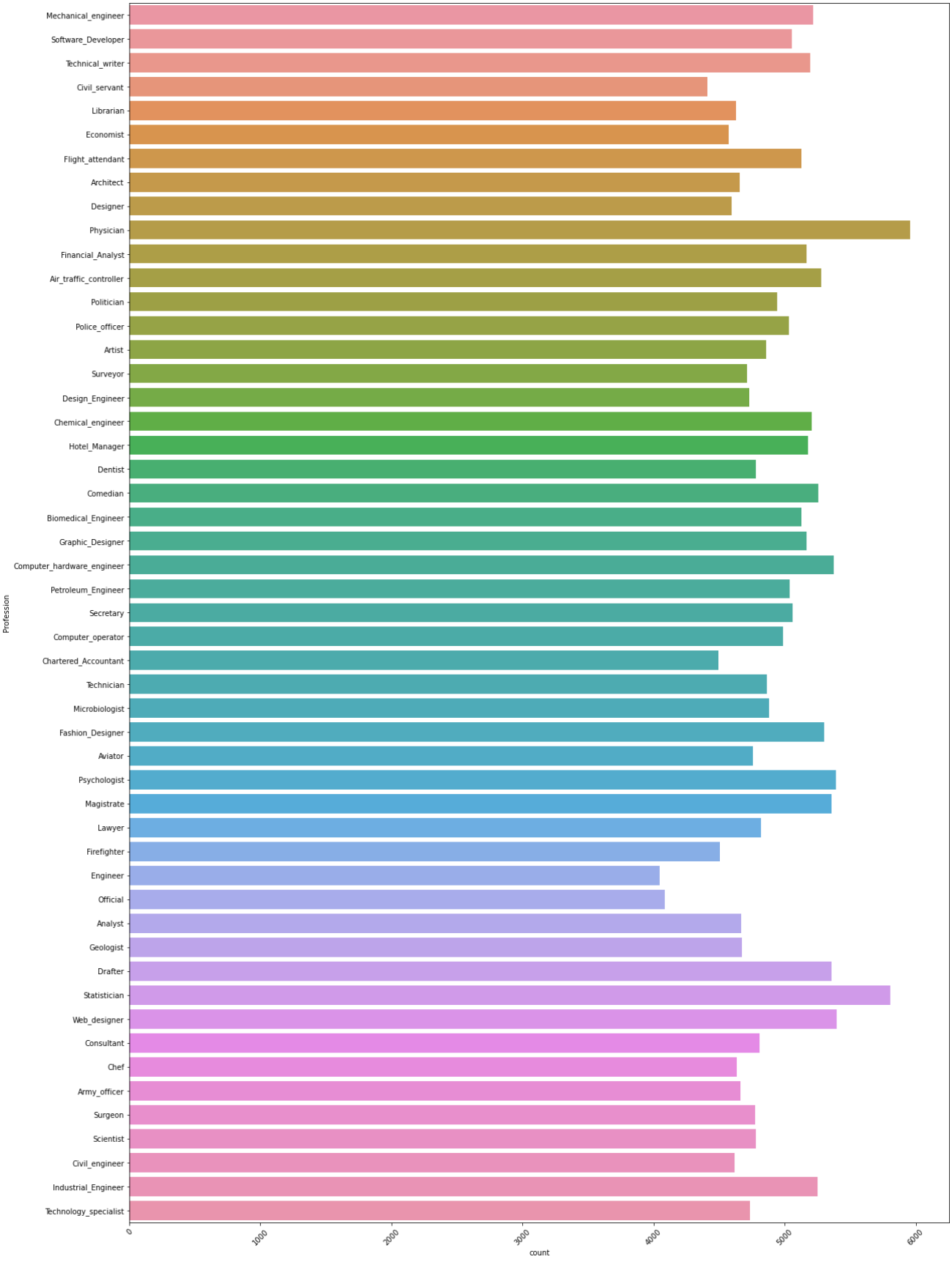
- From the above pie chart we can observe that 92.02% of people live in Rented apartments whereas 5.13% live in Owned apartments and 2.85% of people neither have rented or owned house.

4. STATE



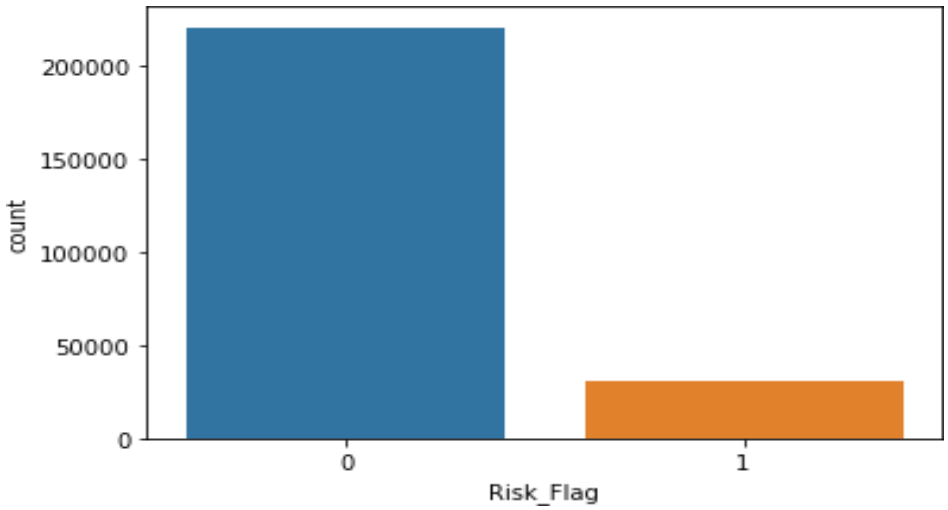
- maximum people reside in Uttar Pradesh and minimum in Sikkim.

5. Profession



- Maximum people are physicians and minimum are engineer

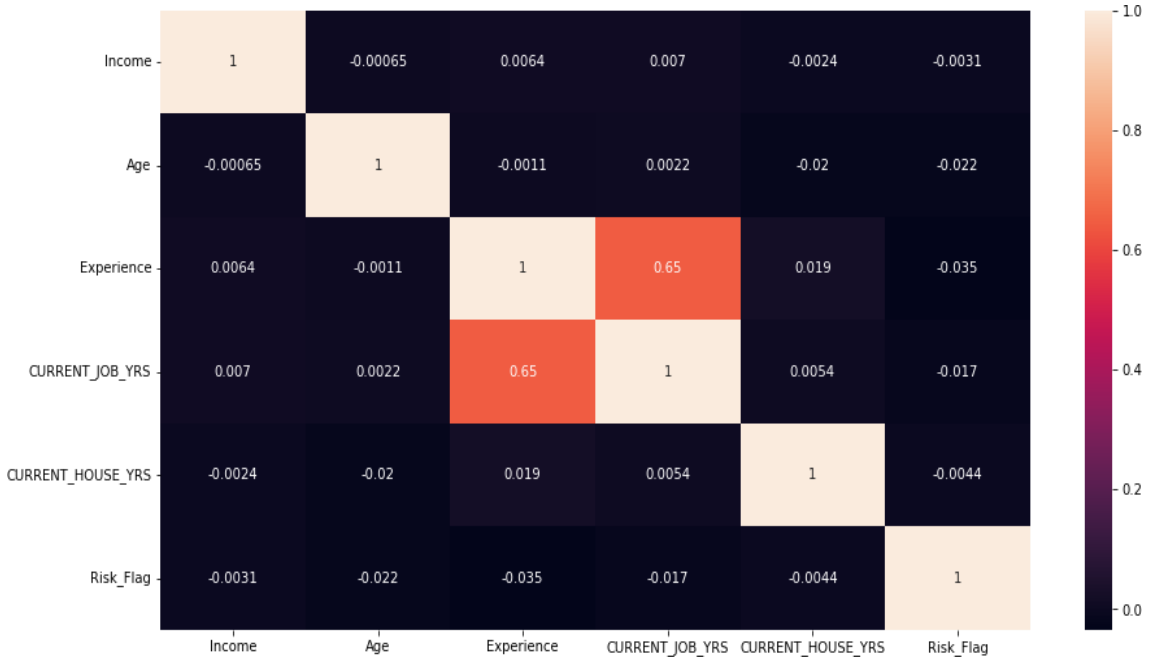
6. Target variable: Risk_Flag



- from the above graph we can say that most of the records in the risk flag variable is falls under zero class that yes (87%).

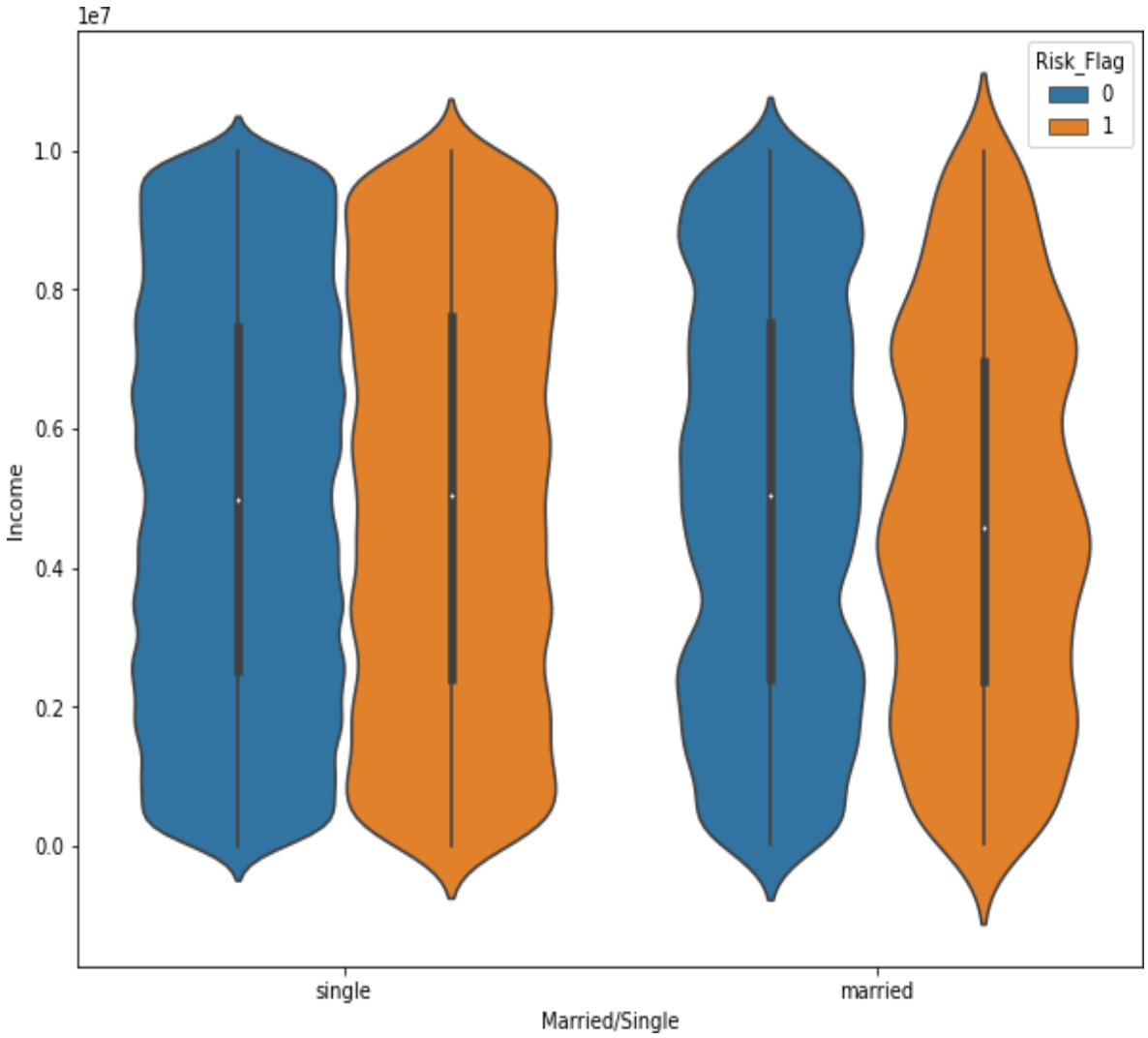
Heatmap for finding the correlation b/w the variable

- Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect



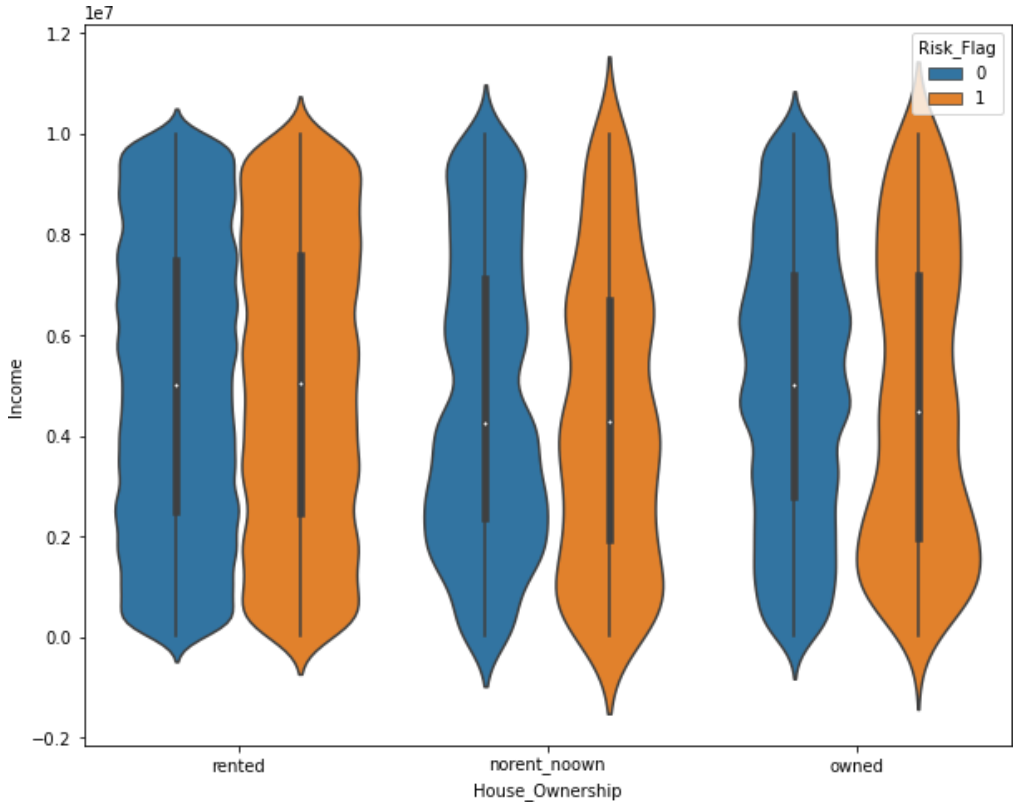
Bivariate analysis (numerical and categorical variables)

1. Married/Single, Income and Risk_Flag



- from above violin plot we can observe that risk of giving lone to single people is comparatively same but in married people the risk of giving loan is higher

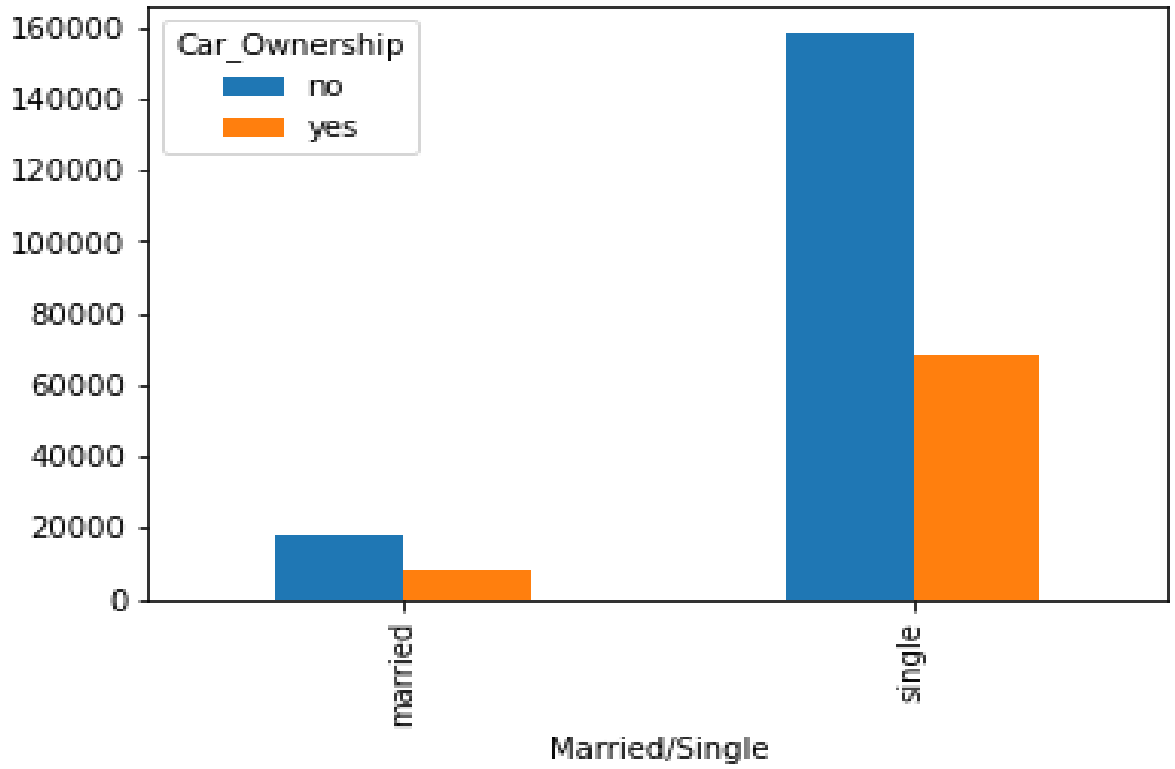
2. house_ownership, Income and Risk_Flag



- From the above plot we can observe in rented people the risk of giving loan is comparatively equal but in non-rented and owned risk is comparatively higher

Categorical to Categorical Bivariate Analysis:

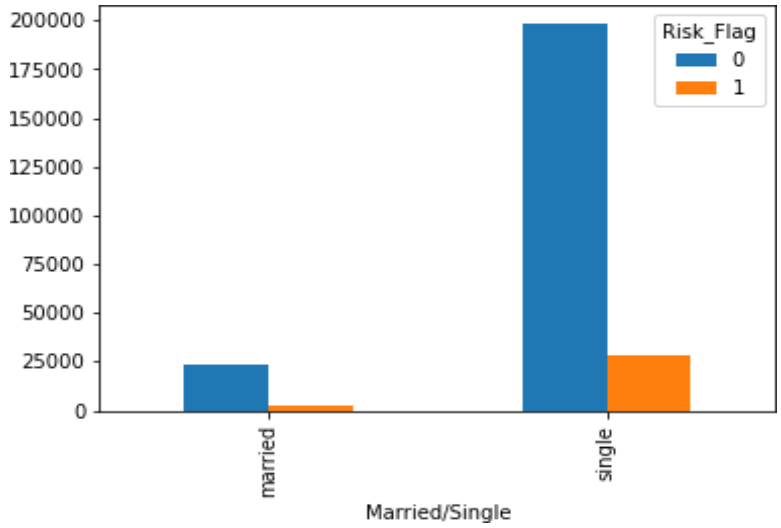
1. Married/Single and car_ownership



Car_Ownership	no	yes
Married/Single		
married	7.147222	3.062302
single	62.694048	27.096429

- From the above graph we can observe that Married people have less car ownership and for Single persons the car ownership is also observed as less. Most of the persons do not own a car.

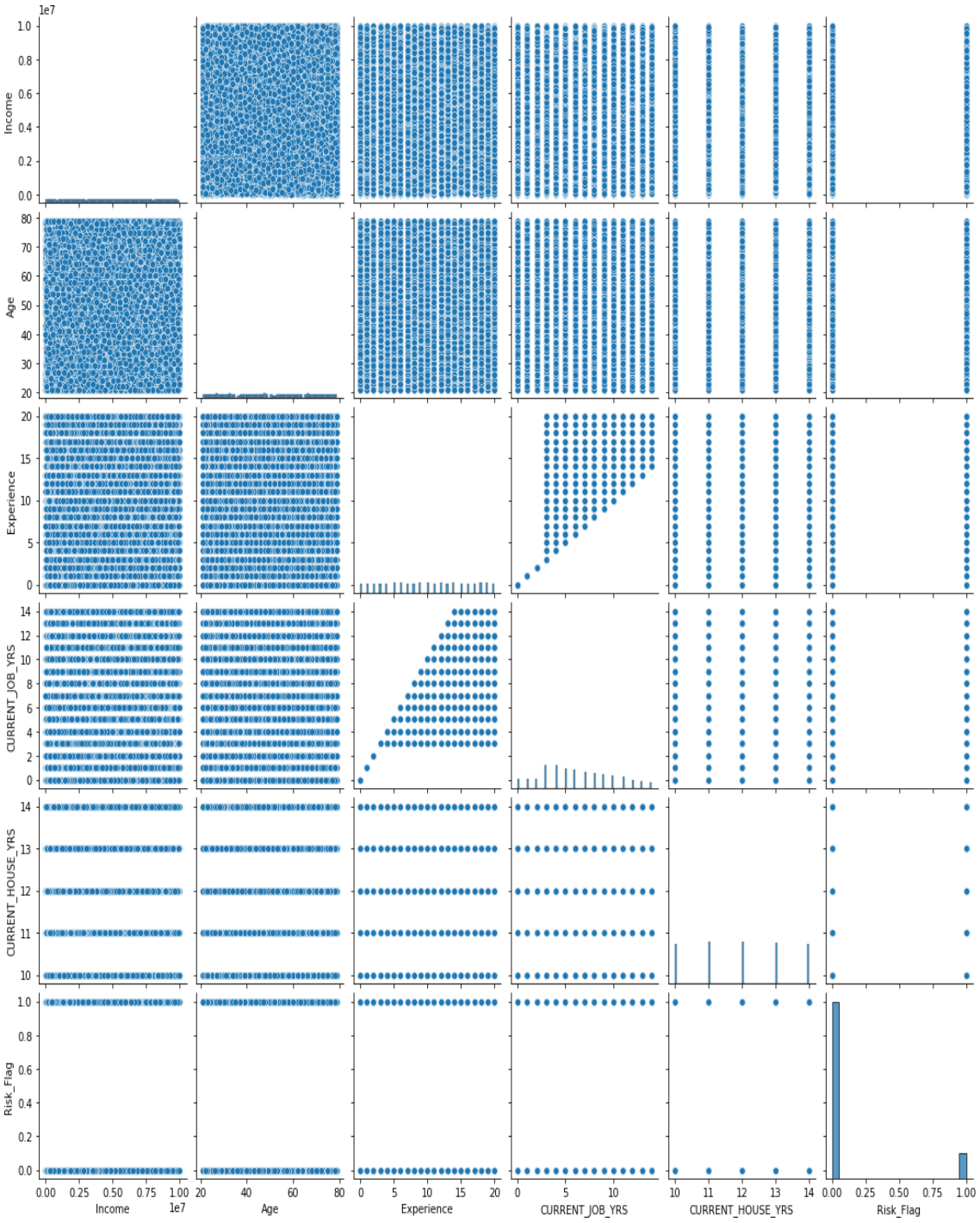
2. Married/Single and Risk Flag



Risk_Flag	0	1
Married/Single		
married	9.163492	1.046032
single	78.536508	11.253968

- From the above graph we can observe that Married people have less defaulter than single persons i.e., giving loan to single person is more risky

Pair plot:



- The above pair plot shows the pairwise relationship between all the numerical variables.
- Current job years and experience has a positive mild correlation.

Model Building:

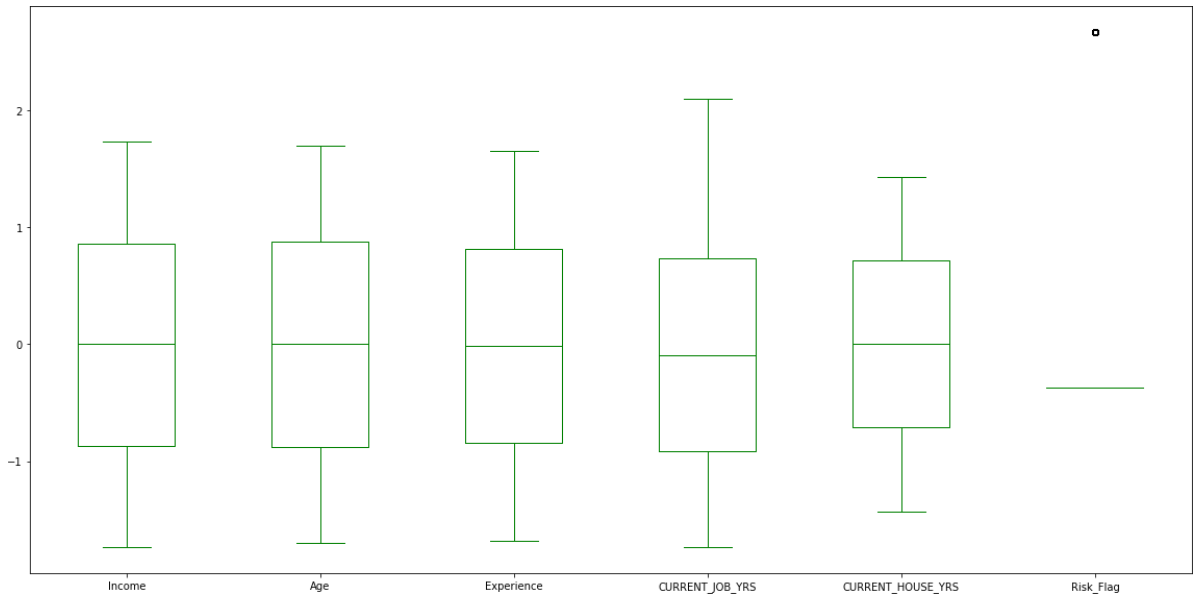
- First of all, we have dropped the risk flag column as it is the target variable and then we have moved towards scaling of the features as there are no outliers in our dataset so no outlier treatment has been done.
- Using the Standard Scaler from sklearn.preprocessing library we have scaled all the numerical variables that are present in our dataset.

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
sc = sc.fit_transform(df.select_dtypes('int','float'))
```

- After that the Box plot is done for all the numerical variables to find out whether there are outliers or not.



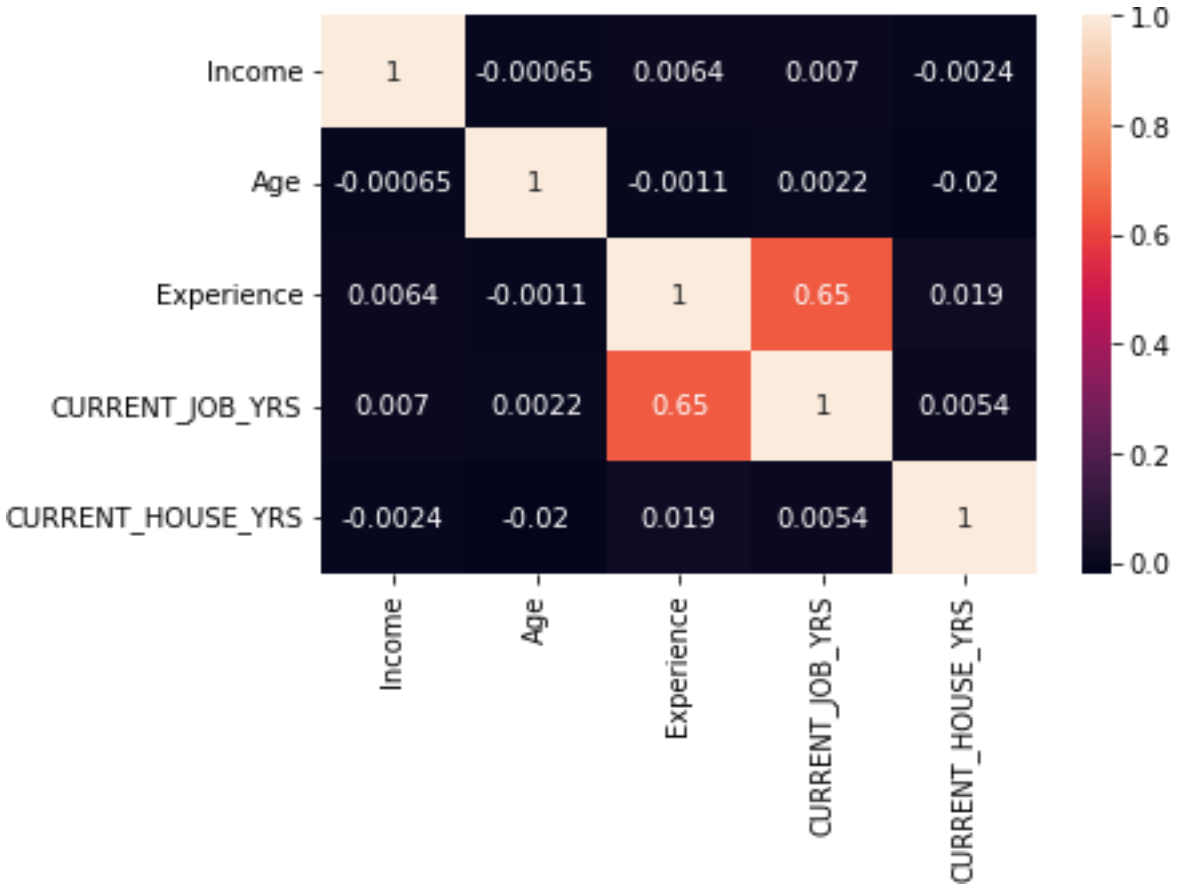
- Moving on, we have dropped the column CITY as it has no significance and it is adding to the complexity of the model.
- For checking the normality of the variables we have checked for the skewness of the variables and the observation is as:

```
df.skew()
```

```
Income             0.005958
Age               -0.005316
Experience        -0.012779
CURRENT_JOB_YRS    0.273146
Risk_Flag          2.295734
dtype: float64
```

- We can observe that all over variables are normally distributed and for the Risk flag we have dropped that column being the target variable.

1. **<u>For checking the collinearity between the independent variables we have plotted heatmap and filtered the values as:</u>**



- And we have observed that there is moderate correlation between Current_Job_yrs and Experience variables.

2. Variance Inflation Factor (VIF):

- Variance inflation factor (VIF) is a measure of the amount of multicollinearity in a set of multiple regression variables. Mathematically, the VIF for a regression model variable is equal to the ratio of the overall model variance to the variance of a model that includes only that single independent variable.

	feat	predictors
4	13.281953	CURRENT_HOUSE_YRS
1	8.556036	Age
3	6.830915	CURRENT_JOB_YRS
2	6.541820	Experience
0	3.879021	Income

- From the above values of VIF we can observe that the variables have moderate collinearity and only one variable has high collinearity and i.e., Current_House_yrs and thus we are dropping it.

- Moving ahead, we have done the encoding for our categorical variables
- Label Encoder has been used for encoding Profession and State as they have many unique values and for rest of the categorical variables one hot encoding has been done.

- After that the dataset has been scaled and dataset is splited into train and test dataset for model building.

- We have built a base model with Logistic Regression and has given accuracy score of 0.87

- **Since our dataset is highly imbalanced, moving forward first we will be using SMOTE for balancing of the data** and then we will make transformations in our model depending upon the requirement.

Smote:

- SMOTE stands for Synthetic Minority Oversampling Technique. At the heart of SMOTE, the construction of Minority Class Exists.
- Creates Artificial Observations based on feature space similar to minority samples. In simple words, it generates random set of minority class observations to shift the learning bias
- The SMOTE Samples are the linear combination of two similar samples from the minority class.

```
In [243]: # SMOTE as our target variable is imbalanced

          sm = SMOTE(sampling_strategy='minority', random_state=1)
```

```
df_smotted['Risk_Flag'].value_counts()
```

```
0    221004
1    221004
Name: Risk_Flag, dtype: int64
```

- Now our target variable is balanced we will proceed further with the model building

Model building after balancing the data:

- Splitting the data into train and test

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

```
# shape of X_train

X_train.shape
```

```
(309405, 10)
```

- Creating a base model: **Logistic Regression**

```
lr = LogisticRegression()

Mod_base = lr.fit(X_train,y_train)
```

- Classification report - train set:

	precision	recall	f1-score	support
0	0.54	0.50	0.52	154643
1	0.53	0.57	0.55	154762
accuracy			0.54	309405
macro avg	0.54	0.54	0.53	309405
weighted avg	0.54	0.54	0.53	309405

- Classification report - test set:

	precision	recall	f1-score	support
0	0.54	0.50	0.52	66361
1	0.53	0.57	0.55	66242
accuracy			0.53	132603
macro avg	0.53	0.53	0.53	132603
weighted avg	0.53	0.53	0.53	132603

- **storing the results into one data frame:**

```
results_df = pd.DataFrame({'Score': [r1_test]},index=['Logistic Regression'])
results_df
```

]:

	Score
Logistic Regression	0.534392

RFE (Recursive Feature Elimination):

- Feature ranking with recursive feature elimination. Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

```
rfe_model = RFE(RandomForestClassifier(),n_features_to_select= 7)
rfe_model= rfe_model.fit(X_train,y_train)
```

- **From RFE, these are the desired set of features:**

  Income, Age, Experience, CURRENT_JOB_YRS, Profession, STATE', car_ownership_yes

- We further use the Random Forest Classifier for getting a better model.

Random Forest Classifier:

- The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree

```
# assigning the significant features received from RFE to new train and test

X_train_new = X_train[significant_features]
X_test_new= X_test[significant_features]
```

```
# Random forest classifier

rf= RandomForestClassifier()
rf.fit(X_train_new,y_train)
```

```
RandomForestClassifier()
```

- classification report of train:

	precision	recall	f1-score	support
0	1.00	0.91	0.96	154643
1	0.92	1.00	0.96	154762
accuracy			0.96	309405
macro avg	0.96	0.96	0.96	309405
weighted avg	0.96	0.96	0.96	309405

- classification report of test:

	precision	recall	f1-score	support
0	0.96	0.88	0.92	66361
1	0.89	0.96	0.92	66242
accuracy			0.92	132603
macro avg	0.92	0.92	0.92	132603
weighted avg	0.92	0.92	0.92	132603

- **storing the results into one data frame**

	Score
Logistic Regression	0.534392
Random Forest	0.941623

confusion matrix of train: Random Forest

- A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

```
cm = confusion_matrix(y_train, y_pred_rf_train)
cm
```

```
array([[141396,  13247],
       [     0, 154762]], dtype=int64)
```

```
# confusion matrix of test : Random Forest

cm = confusion_matrix(y_test, y_pred_rf_test)
cm
```

```
array([[60130,  6231],
       [ 1510, 64732]], dtype=int64)
```

Decision tree classifier:

- A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

```
dtf= DecisionTreeClassifier()
dtf.fit(X_train_new,y_train)
```

```
DecisionTreeClassifier()
```

- classification report of train: Decision Tree classifier

```
              precision    recall  f1-score   support

           0       1.00      0.91      0.96    154643
           1       0.92      1.00      0.96    154762

    accuracy                           0.96    309405
   macro avg       0.96      0.96      0.96    309405
weighted avg       0.96      0.96      0.96    309405
```

- classification report of test: Decision Tree classifier

	precision	recall	f1-score	support
0	0.96	0.88	0.92	66361
1	0.89	0.96	0.92	66242
accuracy			0.92	132603
macro avg	0.92	0.92	0.92	132603
weighted avg	0.92	0.92	0.92	132603

```
# train score of Decision Tree

r3_train=dtf.score(X_train_new,y_train)
r3_train
```

```
0.9571855658441202
```

```
# test score of Decision Tree

r3_test=dtf.score(X_test_new,y_test)
r3_test
```

```
0.919481459695482
```

- **storing the result in data frame:**

	Score
Logistic Regression	0.534392
Random Forest	0.941623
Decision Tree	0.919481

AdaBoost:

- Adaboost is basically a forest of stumps. It is sensitive to noisy data and outliers. Usually, decision trees are used for modelling Multiple sequential models are created, each correcting the errors from the last model. AdaBoost works by weighing the observations, putting more weight on difficult to classify instances and less on those already handled well

```
ada = AdaBoostClassifier(base_estimator=dtf,random_state=1)
```

```
tuned_param= [ { 'n_estimators':[10],
                 'learning_rate':[0.1,0.01,0.001,0.15,0.015] } ]
```

- GridSearchCV: Exhaustive search over specified parameter values for an estimator.

```
ada_grid= GridSearchCV(estimator=ada, param_grid= tuned_param, cv=3, n_jobs=-1)
```

```
ada_grid.fit(X_train_new, y_train)
```

- best parameters from grid search:

  {'learning rate': 0.015, 'n_estimators':10}

```
ada = AdaBoostClassifier(base_estimator=dtf,random_state=1,n_estimators= 10,learning_rate=0.015)
```

```
# fitting the model

ada.fit(X_train_new,y_train)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), random_state=1)
```

- classification report of train: Ada Boost

	precision	recall	f1-score	support
0	1.00	0.91	0.96	154643
1	0.92	1.00	0.96	154762
accuracy			0.96	309405
macro avg	0.96	0.96	0.96	309405
weighted avg	0.96	0.96	0.96	309405

- classification report of test: Ada Boost

	precision	recall	f1-score	support
0	0.96	0.88	0.92	66361
1	0.89	0.97	0.93	66242
accuracy			0.93	132603
macro avg	0.93	0.93	0.93	132603
weighted avg	0.93	0.93	0.93	132603

```
# confusion matrix of train : Ada Boost

confusion_matrix(y_train,y_pred_ada_train)
```

```
array([[141396,  13247],
       [     0, 154762]], dtype=int64)
```

```
# confusion matrix of test : Ada Boost

confusion_matrix(y_test,y_pred_ada_test)
```

```
array([[58716,  7645],
       [ 2156, 64086]], dtype=int64)
```

```
# train score of Ada Boost

r4_train=ada.score(X_train_new,y_train)
r4_train
```

```
0.9571855658441202
```

- **storing the result in data frame:**

	Score
Logistic Regression	0.534392
Random Forest	0.941623
Decision Tree	0.919481
Ada Boost	0.926088
Ada Boost on DTF	0.926088

Gradient Boosting:

- Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.

```
gb_model= GradientBoostingClassifier(n_estimators=10,max_depth =5)
```

```
# fitting the model

gb_model.fit(X_train_new,y_train)
```

```
GradientBoostingClassifier(max_depth=5, n_estimators=10)
```

- classification report of train: Gradient Boosting

	precision	recall	f1-score	support
0	0.59	0.70	0.64	154643
1	0.63	0.52	0.57	154762
accuracy			0.61	309405
macro avg	0.61	0.61	0.60	309405
weighted avg	0.61	0.61	0.60	309405

- classification report of test: Gradient Boosting

	precision	recall	f1-score	support
0	0.59	0.70	0.64	66361
1	0.63	0.52	0.57	66242
accuracy			0.61	132603
macro avg	0.61	0.61	0.60	132603
weighted avg	0.61	0.61	0.60	132603

```
# confusion matrix of train : Gradient Boosting

confusion_matrix(y_train,y_pred_gb_train)
```

```
array([[108459,  46184],
       [ 74999,  79763]], dtype=int64)
```

```
# confusion matrix of test : Gradient Boosting

confusion_matrix(y_test,y_pred_gb_test)
```

```
array([[46249, 20112],
       [32113, 34129]], dtype=int64)
```

```
# score of train Gradient Boosting

r5_train=gb_model.score(X_train_new,y_train)
r5_train
```

```
0.6083353533394742
```

```
# score of test Gradient Boosting

r5_test=gb_model.score(X_test_new,y_test)
r5_test
```

```
0.6061552151912099
```

- **storing the results in data frame:**

	Score
Logistic Regression	0.534392
Random Forest	0.941623
Decision Tree	0.919481
Ada Boost	0.926088
Ada Boost on DTF	0.926088
Gradient Boost	0.606155

XG Boost:

- XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.

```
xgb_model = XGBClassifier(learning_rate = 0.015)
```

```
#fitting the model

xgb_model.fit(X_train_new,y_train)
```

- classification report of train: XG Boost

	precision	recall	f1-score	support
0	0.64	0.73	0.68	154643
1	0.68	0.58	0.63	154762
accuracy			0.66	309405
macro avg	0.66	0.66	0.65	309405
weighted avg	0.66	0.66	0.65	309405

- classification report of test: XG Boost

	precision	recall	f1-score	support
0	0.63	0.73	0.68	66361
1	0.68	0.58	0.63	66242
accuracy			0.65	132603
macro avg	0.66	0.65	0.65	132603
weighted avg	0.66	0.65	0.65	132603

```
confusion_matrix(y_train,y_pred_xg_train)
```

```
array([[112834,  41809],
       [ 64777,  89985]], dtype=int64)
```

```
confusion_matrix(y_test,y_pred_xg_test)
```

```
array([[48130, 18231],
       [27796, 38446]], dtype=int64)
```

```
# score of train XG Boost

r6_train=xgb_model.score(X_train_new,y_train)
r6_train
```

```
0.6555130007595223
```

```
# score of test XG Boost

r6_test=xgb_model.score(X_test_new,y_test)
r6_test
```

```
0.6528962391499438
```

- **storing the results in data frame:**

	Score
Logistic Regression	0.534392
Random Forest	0.941623
Decision Tree	0.919481
Ada Boost	0.926088
Ada Boost on DTF	0.926088
Gradient Boost	0.606155
XG Boost	0.652896

balanced bagging classifier:

- A Bagging classifier with additional balancing. This implementation of Bagging is similar to the scikit-learn implementation. It includes an additional step to balance the training set at fit time using a given sampler.

```
BalancedBaggingClassifier(base_estimator= DecisionTreeClassifier(),sampling_strategy='auto', replacement= False, random_state=1)
```

```
#fitting the model

bbc.fit(X_train_new,y_train)
```

```
BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
                          random_state=1)
```

- classification report of train: Balanced bagging

	precision	recall	f1-score	support
0	1.00	0.91	0.95	154643
1	0.92	1.00	0.96	154762
accuracy			0.96	309405
macro avg	0.96	0.96	0.96	309405
weighted avg	0.96	0.96	0.96	309405

- classification report of test: Balanced bagging

	precision	recall	f1-score	support
0	0.96	0.90	0.93	66361
1	0.91	0.97	0.94	66242
accuracy			0.93	132603
macro avg	0.94	0.93	0.93	132603
weighted avg	0.94	0.93	0.93	132603

```
# score of train Balanced bagging

r7_train=bbc.score(X_train_new,y_train)
r7_train
```

```
0.9559541701006771
```

```
# score of test Balanced bagging

r7_test=bbc.score(X_test_new,y_test)
r7_test
```

```
0.9330105653718241
```

- **storing the results in data frame:**

	Score
Logistic Regression	0.534392
Random Forest	0.941623
Decision Tree	0.919481
Ada Boost	0.926088
Ada Boost on DTF	0.926088
Gradient Boost	0.606155
XG Boost	0.652896
Balnced Bagging Classifier	0.933011

- **sorting the results data frame in descending order:**

	Score
Random Forest	0.941623
Balnced Bagging Classifier	0.933011
Ada Boost	0.926088
Ada Boost on DTF	0.926088
Decision Tree	0.919481
XG Boost	0.652896
Gradient Boost	0.606155
Logistic Regression	0.534392

The best results are given by Random Forest classifier and using that we are able to predict with 94% accuracy.

Model validation:

- Model validation refers to the process of confirming that the model actually achieves its intended purpose. In most situations, this will involve confirmation that the model is predictive under the conditions of its intended use

```
#model validation

cross_score= cross_val_score(estimator=RandomForestClassifier(),X=X_train_new,y=y_train,cv=5)
cross_score
```

```
array([0.94053102, 0.93854333, 0.9399331 , 0.93883421, 0.93906045])
```