PROJECT REPORT

Introduction:

Now a day's bank plays a vital role in market economy. The success or failure of organization largely depends on the industry's ability to evaluate credit risk. Before giving the credit loan to borrowers, bank decides whether the borrower is bad (defaulter) or good (non-defaulter).

Loan amount, costumer's history governs his creditability for receiving loan. The problem is to classify borrower as defaulter or non-defaulter. However developing such a model is a very challenging task due to increasing in demands for loans. A prototype of the model is described in the paper which can be used by the organizations for making the correct or right decision for approve or reject the request for loan of the customers. This work includes the construction of different machine learning models.

Working Steps:

STEP 1:

Open jupyter Notebook and connect the database with python

Code:

```
import cx_Oracle

cx_Oracle.init_oracle_client(lib_dir=r"C:\Program Files (x86)\Oracle\instantclient_19_5")

con = cx_Oracle.connect('py/py@192.168.1.42/orcl')

cur = con.cursor()
```

STEP 2:

Import the libraries and load the dataset of customer table with loan wise data

Code:

```
 querys_mast= "select * from PY_customer_mast"

df_mast= pd.read_sql_query(querys_mast,con)

querys_loans= "select * from PY_customer_all_loans"

df_mast= pd.read_sql_query(querys_loans,con)

df1 = pd.merge(df_mast,df_general, left_on='PY_CUSTOMER_ID',
right_on='PY_CUSTOMER_ID', how='inner')

df = pd.merge(df1,dfs, left_on='PY_CUSTOMER_ID', right_on='PY_CUSTOMER_ID',
how='inner')

print(df)
```

STEP 3:

Data cleaning of the dataset with missing value treatment (mean, mode or median)

Code:

```
df.isnull().sum()

# drop columns with more than 80% null values

df = df.dropna(thresh=df.shape[0]*0.2,how='all',axis=1)

df['MOBILE_NO']=df['MOBILE_NO'].apply(lambda x: 'YES' if not pd.isnull(x) else 'NO')

df['AADHAAR_CARD']=df['AADHAAR_CARD'].apply(lambda x: 'YES' if not pd.isnull(x) else
'NO')

df['PAN']=df['PAN'].apply(lambda x: 'YES' if not pd.isnull(x) else 'NO')

df['CUSTOMER_AGE'] = df['CUSTOMER_AGE'].fillna(df['CUSTOMER_AGE'].mean())

df['BLOOD_GROUP'] = df['BLOOD_GROUP'].fillna('None')

df['GUARANTEE_COUNT'] = df['GUARANTEE_COUNT'].fillna('0').astype(int)

df['ADDRESS_PROOF_ID'] = df['ADDRESS_PROOF_ID'].fillna('0').astype(int)

df['IDENTIFICATION_ID'] = df['IDENTIFICATION_ID'].fillna('0').astype(int)

df['GST_VERIFY_FLAG'] = df['GST_VERIFY_FLAG'].fillna('N')

df['CASH_SECURITY_FLAG'] = df['CASH_SECURITY_FLAG'].fillna('N')

df['EMI_AMOUNT'] = df['EMI_AMOUNT'].fillna('0').astype(int)

df['DISBURSEMENT_AMOUNT'] = df['DISBURSEMENT_AMOUNT'].fillna('0').astype(int)

df['NPA_CLASS_ID'] = df['NPA_CLASS_ID'].fillna('ST')

df['AVG_BAL_IN_ACCOUNT_SAVING_ACTIVE'] =

df['AVG_BAL_IN_ACCOUNT_SAVING_ACTIVE'].fillna('0').astype(int)

df['BALANCE_AMOUNT_SAVING_ACTIVE'] =

df['BALANCE_AMOUNT_SAVING_ACTIVE'].fillna('0').astype(int)

df['TOTAL_INTT_PAID_SAVING_ACTIVE'] =

df['TOTAL_INTT_PAID_SAVING_ACTIVE'].fillna('0').astype(int)
```

STEP 4:

Create Target variable of loan status is good or bad by using NPA_STATUS i.e. NPA_STATUS: ST, SS=='good' and others are 'bad'

Code:

```
df['good_bad'] = np.where(df.loc[:, 'NPA_CLASS_ID'].isin(['ST','SS']), 0, 1)
```

STEP 5:

Split the dataset into train & test split and then divide training data into categorical and numerical subsets

Code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100, stratify = y)

X_train_cat = X_train.select_dtypes(include = 'object').copy()

X_train_num = X_train.select_dtypes(include = 'number').copy()
```

STEP 6:

Find P-value using chi square & F statistics for the goodness of fit of your model and p value is the significance value of your tests.

Code:

```
Calculate Chi-Square for categorical

chi2_check = {}

for column in X_train_cat:

    chi, p, dof, ex = chi2_contingency(pd.crosstab(y_train, X_train_cat[column]))

    chi2_check.setdefault('Feature',[]).append(column)

    chi2_check.setdefault('p-value',[]).append(round(p, 10))

Convert the dictionary to a DataFrame

chi2_result = pd.DataFrame(data = chi2_check)

chi2_result.sort_values(by = ['p-value'], ascending = True, inplace = True)

chi2_result.reset_index(drop=True)
```

Calculate ANOVA F-Statistics For Numerical feature

F_statistic, p_values = f_classif(X_train_num, y_train)

ANOVA_F_table = pd.DataFrame(data = {'Numerical_Feature': X_train_num.columns.values, 'F-Score': F_statistic, 'p values': p_values.round(decimals=10)})

ANOVA_F_table.sort_values(by = ['F-Score'], ascending = False, inplace = True)

ANOVA_F_table.reset_index(drop=True)

For example: $p < 0.05$ is the usual test for dependence.

In this case p is greater than 0.05, so we believe the variables are independent

STEP 7:

Calculate pair-wise correlations between on numerical features in a list where top numerical features i.e. ($p < 0.05$)

Code:

save the top 13 numerical features in a list

top_num_features = ANOVA_F_table.iloc[:13,0].to_list()

 # calculate pair-wise correlations between them

corrmat = X_train_num[top_num_features].corr()

plt.figure(figsize=(10,10))

sns.heatmap(corrmat);

STEP 8:

Function to create dummy variables of categorical variables on x_train & x_test model

Code:

X_train

function to create dummy variables

```
def dummy_creation(df, columns_list):
    df_dummies = []
    for col in columns_list:
        df_dummies.append(pd.get_dummies(df[col], prefix = col, prefix_sep = ':'))
```

```
    df_dummies = pd.concat(df_dummies, axis = 1)
    df = pd.concat([df, df_dummies], axis = 1)
    return df
```

apply to our final categorical variables

```
X_train = dummy_creation(X_train, ['GL_TYPE', 'CUSTOMER_GENDER','MARTIAL_STATUS',
                          'MOBILE_NO','AADHAAR_CARD','BLOOD_GROUP','PAN',
                          'DEBIT_CARD_FLAG','KYC_COMPLETE_FLAG',
                          'GLCODE','ACCOUNT_TYPE',
                          'AC_SECURED_FLAG','CASH_SECURITY_FLAG','LIEN_FLAG','NPA_FLAG',
                          'STATUS_CODE_y'])
```

Code:

X_test

```
X_test = dummy_creation(X_test, ['GL_TYPE', 'CUSTOMER_GENDER','MARTIAL_STATUS',
                          'MOBILE_NO','AADHAAR_CARD','BLOOD_GROUP','PAN',
                          'DEBIT_CARD_FLAG','KYC_COMPLETE_FLAG',
                          'GLCODE','ACCOUNT_TYPE',
                          'AC_SECURED_FLAG','CASH_SECURITY_FLAG','LIEN_FLAG','NPA_FLAG',
                          'STATUS_CODE_y'])
```

reindex the dummied test set variables to make sure all the feature columns in the train set are also available in the test set

```
X_test = X_test.reindex(labels=X_train.columns, axis=1, fill_value=0)
```

Calculate WoE (Weight of Evidence) and IV (Information Value)

STEP 9:

Create copies of the 4 training & testing sets to be pre-processed using WoE

Code:

```
X_train_prepr = X_train.copy()
y_train_prepr = y_train.copy()
X_test_prepr = X_test.copy()
y_test_prepr = y_test.copy()
```

STEP 10:

For categorical variable

The function takes 3 arguments: a dataframe (X_train_prepr), a string (column name), and a dataframe (y_train_prepr) and we define a function for plotting WoE across categories that takes 2 arguments: a dataframe and a number

Code:

```
def woe_discrete(df, cat_variabe_name, y_df):
    df = pd.concat([df[cat_variabe_name], y_df], axis = 1)
    df = pd.concat([df.groupby(df.columns.values[0], as_index =
False)[df.columns.values[1]].count(),
   df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()], axis =
1)
    df = df.iloc[:, [0, 1, 3]]
    df.columns = [df.columns.values[0], 'n_obs', 'prop_good']
    df['prop_n_obs'] = df['n_obs'] / df['n_obs'].sum()
    df['n_good'] = df['prop_good'] * df['n_obs']
    df['n_bad'] = (1 - df['prop_good']) * df['n_obs']
    df['prop_n_good'] = df['n_good'] / df['n_good'].sum()
    df['prop_n_bad'] = df['n_bad'] / df['n_bad'].sum()
    df['WoE'] = np.log(df['prop_n_good'] / df['prop_n_bad'])
    df = df.sort_values(['WoE'])
    df = df.reset_index(drop = True)
    df['diff_prop_good'] = df['prop_good'].diff().abs()
    df['diff_WoE'] = df['WoE'].diff().abs()
    df['IV'] = (df['prop_n_good'] - df['prop_n_bad']) * df['WoE']
    df['IV'] = df['IV'].sum()
    return df
```

STEP 11:

Calculate the one by one WoE and IV value table of categorical variables in a dataframe (X_train_prepr) and plot a graph by woe

Code:

df_gl = woe_discrete(X_train_prepr, 'GL_TYPE', y_train_prepr)

graph :

plot_by_woe(df_gl)

df_cash = woe_discrete(X_train_prepr, 'CASH_SECURITY_FLAG', y_train_prepr)

df_ac = woe_discrete(X_train_prepr, 'AC_SECURED_FLAG', y_train_prepr)

df_actype = woe_discrete(X_train_prepr, 'ACCOUNT_TYPE', y_train_prepr)

df_marrital = woe_discrete(X_train_prepr, 'MARTIAL_STATUS', y_train_prepr)

df_adhar = woe_discrete(X_train_prepr, 'AADHAAR_CARD', y_train_prepr)

and so on upto feature_name p value is less than 0.05 of categorical variable

STEP 12:

For Numerical Variable

We define a function to calculate WoE of continuous variables. This is same as the function we defined earlier for discrete variables.

The only difference are the 2 commented lines of code in the function that results in the df being sorted by continuous variable values

Code:

```
def woe_ordered_continuous(df, continuous_variabe_name, y_df):
    df = pd.concat([df[continuous_variabe_name], y_df], axis = 1)
    df = pd.concat([df.groupby(df.columns.values[0], as_index =
False)[df.columns.values[1]].count(),
    df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()], axis
= 1)
    df = df.iloc[:, [0, 1, 3]]
    df.columns = [df.columns.values[0], 'n_obs', 'prop_good']
    df['prop_n_obs'] = df['n_obs'] / df['n_obs'].sum()
```

```
    df['n_good'] = df['prop_good'] * df['n_obs']
    df['n_bad'] = (1 - df['prop_good']) * df['n_obs']
    df['prop_n_good'] = df['n_good'] / df['n_good'].sum()
    df['prop_n_bad'] = df['n_bad'] / df['n_bad'].sum()
    df['WoE'] = np.log(df['prop_n_good'] / df['prop_n_bad'])
    #df = df.sort_values(['WoE'])
    #df = df.reset_index(drop = True)
    df['diff_prop_good'] = df['prop_good'].diff().abs()
    df['diff_WoE'] = df['WoE'].diff().abs()
    df['IV'] = (df['prop_n_good'] - df['prop_n_bad']) * df['WoE']
    df['IV'] = df['IV'].sum()
    return df
```

STEP 13:

Calculate the one by one WoE and IV value table of numerical variables in a dataframe (X_train_prepr) with their values of different columns name

Code:

Overdue period

a) Check overdue period

```
df_temp = woe_ordered_continuous(X_train_prepr, 'OVERDUE_PRD', y_train_prepr)
```

b) Create overdue period factor

```
X_train_prepr_temp = X_train_prepr[(X_train_prepr['OVERDUE_PRD'] <= 60) & (X_train_prepr['OVERDUE_PRD'] >= -4)].copy()
```

fine-classing again

```
X_train_prepr_temp['OVERDUE_PRD_factor'] = pd.cut(X_train_prepr_temp['OVERDUE_PRD'],8)
```

Make sure to select only the relevant indexes in the target column

```
df_temp = woe_ordered_continuous(X_train_prepr_temp, 'OVERDUE_PRD_factor', y_train_prepr[X_train_prepr_temp.index])
print(df_temp)
```

Interest rate

a) Check interest rate

df_temp = woe_ordered_continuous(X_train_prepr, 'INTEREST_RATE', y_train_prepr)

b) Interest rate factor

X_train_prepr['INTEREST_RATE_FACTOR'] = pd.cut(X_train_prepr['INTEREST_RATE'],5)

df_temp = woe_ordered_continuous(X_train_prepr, 'INTEREST_RATE_FACTOR', y_train_prepr)

print(df_temp)

and so on upto feature_name p value is less than 0.05 of categorical variable

Define Custom Class for WoE Binning/Reengineering

STEP 14:

This custom class will create new categorical dummy features based on the cut-off points that we manually identified based on the WoE plots and IV above.

Given the way it is structured, this class also allows a fit transform method to be implemented on it, thereby allowing us to use it as part of a scikit-learn Pipeline

Code

a)

```
ref_categories = ['BLOOD_GROUP:AB ve+','RISK_TYPE_ID:448','OVERDUE_PRD:<-3.644','OVERDUE_PRD:>59.94','PERIOD_MONTHS:>120.0',
            'TOTAL_NO_OF_LOAN_ACCOUNT:>11.0','AVG_BAL_IN_ACCOUNT:>3700000']
```

b)

```
class WoE_Binning(BaseEstimator, TransformerMixin):

  def __init__(self, X): # no *args or *kargs
    self.X = X

   def fit(self, X, y = None):
      return self #nothing else to do

   def transform(self, X):
```

```
X_new = X.loc[:, 'GL_TYPE:Cash Credit' : 'GL_TYPE:Term Loan']

X_new = X.loc[:, 'CASH_SECURITY_FLAG:N' : 'CASH_SECURITY_FLAG:Y']

X_new = X.loc[:, 'AC_SECURED_FLAG:Y' : 'AC_SECURED_FLAG:N']

X_new['ACCOUNT_TYPE:Senior Citizen'] = X.loc[:,'ACCOUNT_TYPE:Senior Citizen']

X_new['ACCOUNT_TYPE:Women'] = X.loc[:,'ACCOUNT_TYPE:Women']

X_new['ACCOUNT_TYPE:General Customer'] = X.loc[:,'ACCOUNT_TYPE:General Customer']
X_new['ACCOUNT_TYPE:Firm'] = X.loc[:,'ACCOUNT_TYPE:Firm']

X_new['ACCOUNT_TYPE:Handicap_other_society'] = sum([X['ACCOUNT_TYPE:Handicap'], X['ACCOUNT_TYPE:Other'], X['ACCOUNT_TYPE:Society'],
                                X['ACCOUNT_TYPE:Staff'], X['ACCOUNT_TYPE:Student'], X['ACCOUNT_TYPE:Trust']])


X_new = pd.concat([X_new, X.loc[:, 'MARTIAL_STATUS:W': 'MARTIAL_STATUS:D']], axis = 1)
X_new = X.loc[:, 'AADHAAR_CARD:YES': 'AADHAAR_CARD:NO']

X_new = X.loc[:, 'MOBILE_NO:YES': 'MOBILE_NO:NO']

X_new = X.loc[:, 'NPA_FLAG:N': 'NPA_FLAG:Y']

X_new = pd.concat([X_new, X.loc[:, 'STATUS_CODE_y:F':'STATUS_CODE_y:A']], axis = 1)
X_new = X.loc[:, 'LIEN_FLAG:N': 'LIEN_FLAG:Y']

X_new = X.loc[:, 'CUSTOMER_GENDER:F': 'CUSTOMER_GENDER:N']

X_new = X.loc[:, 'PAN:YES': 'PAN:NO']

X_new = pd.concat([X_new, X.loc[:, 'BLOOD_GROUP:AB ve+':'BLOOD_GROUP:None']], axis = 1)

X_new['OVERDUE_PRD:<-3.644'] = np.where((X['OVERDUE_PRD'] <= -3.644), 1, 0)

X_new['OVERDUE_PRD:-3.644-9.124'] = np.where((X['OVERDUE_PRD'] > -3.644) & (X['OVERDUE_PRD'] <= 9.124), 1, 0)

X_new['OVERDUE_PRD:9.124-21.828'] = np.where((X['OVERDUE_PRD'] > 9.124) & (X['OVERDUE_PRD'] <= 21.828), 1, 0)
```

```
    X_new['OVERDUE_PRD:21.828-34.532'] = np.where((X['OVERDUE_PRD'] > 21.828) & (
X['OVERDUE_PRD'] <= 34.532), 1, 0)

    X_new['OVERDUE_PRD:34.532-47.236'] = np.where((X['OVERDUE_PRD'] > 34.532) & (
X['OVERDUE_PRD'] <= 47.236), 1, 0)

    X_new['OVERDUE_PRD:47.236-59.94'] = np.where((X['OVERDUE_PRD'] > 47.236) & (X
['OVERDUE_PRD'] <= 59.94), 1, 0)

    X_new['OVERDUE_PRD:>59.94'] = np.where((X['OVERDUE_PRD'] > 59.94), 1, 0)

    X_new['PERIOD_MONTHS:< 0'] = np.where((X['PERIOD_MONTHS'] <= 0.0), 1, 0)

    X_new['PERIOD_MONTHS:0.0-24.0'] = np.where((X['PERIOD_MONTHS'] > 0.0) & (X['P
ERIOD_MONTHS'] <= 24.0), 1, 0)

    X_new['PERIOD_MONTHS:24.0-48.0'] = np.where((X['PERIOD_MONTHS'] > 24.0) & (X[
'PERIOD_MONTHS'] <= 48.0), 1, 0)

    X_new['PERIOD_MONTHS:48.0-72.0'] = np.where((X['PERIOD_MONTHS'] > 48.0) & (X[
'PERIOD_MONTHS'] <= 72.0), 1, 0)

    X_new['PERIOD_MONTHS:72.0-96.0'] = np.where((X['PERIOD_MONTHS'] > 72.0) & (X[
'PERIOD_MONTHS'] <= 96.0), 1, 0)

    X_new['PERIOD_MONTHS:96.0-120.0'] = np.where((X['PERIOD_MONTHS'] > 96.0) & (
X['PERIOD_MONTHS'] <= 120.0), 1, 0)

    X_new['PERIOD_MONTHS:>120.0'] = np.where((X['PERIOD_MONTHS'] > 120.0), 1, 0)

    X_new['INTEREST_RATE:< 7.239'] = np.where((X['INTEREST_RATE'] <= 7.239), 1, 0)

    X_new['INTEREST_RATE:7.239-9.042'] = np.where((X['INTEREST_RATE'] > 7.239) & (X['
INTEREST_RATE'] <= 9.042), 1, 0)
    X_new['INTEREST_RATE:9.042-10.833'] = np.where((X['INTEREST_RATE'] > 9.042) & (X
['INTEREST_RATE'] <= 10.833), 1, 0)

    X_new['INTEREST_RATE:10.833-12.625'] = np.where((X['INTEREST_RATE'] > 10.833) &
(X['INTEREST_RATE'] <= 12.625), 1, 0)

    X_new['INTEREST_RATE:12.625-14.417'] = np.where((X['INTEREST_RATE'] > 12.625) &
(X['INTEREST_RATE'] <= 14.417), 1, 0)

    X_new['INTEREST_RATE:14.417-16.208'] = np.where((X['INTEREST_RATE'] > 14.417) &
(X['INTEREST_RATE'] <= 16.208), 1, 0)
```

```
X_new['INTEREST_RATE:16.208-18.0'] = np.where((X['INTEREST_RATE'] > 16.208) & (X['INTEREST_RATE'] <= 18.0), 1, 0)

X_new['INTEREST_RATE:>18.0'] = np.where((X['INTEREST_RATE'] > 18.0), 1, 0)

X_new['RISK_TYPE_ID:447'] = np.where((X['RISK_TYPE_ID'] == 447), 1, 0)

X_new['RISK_TYPE_ID:448'] = np.where((X['RISK_TYPE_ID'] == 448), 1, 0)

X_new['RISK_TYPE_ID:681'] = np.where((X['RISK_TYPE_ID'] == 681), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:<1.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 1.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:1.0-2.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 1.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 2.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:2.0-3.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 2.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 3.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:3.0-4.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 3.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 4.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:4.0-5.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 4.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 5.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:5.0-6.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 5.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 6.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:6.0-7.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 6.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 7.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:7.0-8.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 7.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 8.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:8.0-9.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 8.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 9.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:9.0-10.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 9.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 10.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:10.0-11.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 10.0) & (X['TOTAL_NO_OF_LOAN_ACCOUNT'] <= 11.0), 1, 0)

X_new['TOTAL_NO_OF_LOAN_ACCOUNT:>11.0'] = np.where((X['TOTAL_NO_OF_LOAN_ACCOUNT'] > 11.0), 1, 0)
```

```
    X_new['AVG_BAL_IN_ACCOUNT:<-3695.376'] = np.where((X['AVG_BAL_IN_ACCOUNT'] <= -3695.376), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:-3695.376-369537.596'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > -3695.376) & (X['AVG_BAL_IN_ACCOUNT'] <= 369537.596), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:369537.596-739075.192'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 369537.596) & (X['AVG_BAL_IN_ACCOUNT'] <= 739075.192), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:739075.192-1108612.788'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 739075.192) & (X['AVG_BAL_IN_ACCOUNT'] <= 1108612.788), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:1108612.788-1478150.384'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 1108612.788) & (X['AVG_BAL_IN_ACCOUNT'] <= 1478150.384), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:1478150.384-1847687.98'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 1478150.384) & (X['AVG_BAL_IN_ACCOUNT'] <= 1847687.98), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:1847687.98-2217225.576'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 1847687.98) & (X['AVG_BAL_IN_ACCOUNT'] <= 2217225.576), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:2217225.576-2586763.172'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 2217225.576) & (X['AVG_BAL_IN_ACCOUNT'] <= 2586763.172), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:2586763.172-2956300.768'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 2586763.172) & (X['AVG_BAL_IN_ACCOUNT'] <= 2956300.768), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:2956300.768-3325838.364'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 2956300.768) & (X['AVG_BAL_IN_ACCOUNT'] <= 3325838.364), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:3325838.364-3695375.96'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 3325838.364) & (X['AVG_BAL_IN_ACCOUNT'] <= 3695375.96), 1, 0)

    X_new['AVG_BAL_IN_ACCOUNT:>3700000'] = np.where((X['AVG_BAL_IN_ACCOUNT'] > 3695375.96), 1, 0)

    X_new.drop(columns = ref_categories, inplace = True)

    return X_new
```

STEP 15:

Define modelling pipeline using logistics regression and define cross-validation criteria. Repeated Stratified KFold automatially takes care of the class imbalance while splitting and then fit and evaluate the logistic regression pipeline with cross-validation as defined in cv

Code:

```
reg = LogisticRegression(max_iter=1000, class_weight = 'balanced')

woe_transform = WoE_Binning(X)

pipeline = Pipeline(steps=[('woe', woe_transform), ('model', reg)])

define cross-validation criteria. Repeated Stratified KFold automatially takes care of the class imbalance while splitting

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

# fit and evaluate the logistic regression pipeline with cross-validation as defined in cv

scores = cross_val_score(pipeline, X_train, y_train, scoring = 'roc_auc', cv = cv)

AUROC = np.mean(scores)

GINI = AUROC * 2 - 1

print the mean AUROC score and Gini

print('Mean AUROC: %.4f' % (AUROC))

print('Gini: %.4f' % (GINI))
```

STEP 16:

a) First create a transformed training set through our WoE (Weight of Evidance) Binning custom class

Code:

```
pipeline.fit(X_train, y_train)

X_train_woe_transformed = woe_transform.fit_transform(X_train)
```

b) Store the column names in X_train as a list

feature_name = X_train_woe_transformed.columns.values

c) Create a summary table of our logistic regression model

summary_table = pd.DataFrame(columns = ['Feature name'], data = feature_name)

d) Create a new column in the dataframe, called 'Coefficients', with row values the transpo sed coefficients from the 'LogisticRegression'
Model

summary_table['Coefficients'] = np.transpose(pipeline['model'].coef_)

summary_table.index = summary_table.index + 1

e) Assign our model intercept to this new row

summary_table.loc[0] = ['Intercept', pipeline['model'].intercept_[0]]

summary_table.sort_index(inplace = True)

summary_table

STEP 17:

Prediction time

make preditions on our test set

y_hat_test = pipeline.predict(X_test)

get the predicted probabilities

y_hat_test_proba = pipeline.predict_proba(X_test)

select the probabilities of only the positive class (class 1 - default)

y_hat_test_proba = y_hat_test_proba[:][: , 1]

we will now create a new DF with actual classes and the predicted probabilities # create a te mp y_test DF to reset its index to allow proper concaternation with y_hat_test_proba

y_test_temp = y_test.copy() y_test_temp.reset_index(drop = True, inplace = True) y_test_proba = pd.concat([y_test_temp, pd.DataFrame(y_hat_test_proba)], axis = 1)

check the shape to make sure the number of rows is same as that in

y_test y_test_proba.shape

Rename the columns

y_test_proba.columns = ['y_test_class_actual', 'y_hat_test_proba']

Makes the index of one dataframe equal to the index of another dataframe.

y_test_proba.index = X_test.index

y_test_proba.head()

Confusion Matrix and AUROC on Test Set

assign a threshold value to differentiate good with bad

tr = 0.5

create a new column for the predicted class based on predicted probabilities and threshold We will determine this optimat threshold later in this project

y_test_proba['y_test_class_predicted'] = np.where(y_test_proba['y_hat_test_proba'] > tr, 1, 0)

create the confusion matrix

confusion_matrix(y_test_proba['y_test_class_actual'], y_test_proba['y_test_class_predicted'])

get the values required to plot a ROC curve

fpr, tpr, thresholds = roc_curve(y_test_proba['y_test_class_actual'], y_test_proba['y_hat_test_proba'])

plot the ROC curve

plt.plot(fpr, tpr)

plot a secondary diagonal line, with dashed line style and black color to represent a no-skill classifier

plt.plot(fpr, fpr, linestyle = '--', color = 'k')

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve');

Calculate the Area Under the Receiver Operating Characteristic Curve (AUROC) on our test set

AUROC = roc_auc_score(y_test_proba['y_test_class_actual'], y_test_proba['y_hat_test_proba'])

print(AUROC)

calculate Gini from AUROC

Gini = AUROC * 2 - 1

print(Gini)

draw a PR curve

calculate the no skill line as the proportion of the positive class

no_skill = len(y_test[y_test == 1]) / len(y)

plot the no skill precision-recall curve

plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')

calculate inputs for the PR curve

precision, recall, thresholds = precision_recall_curve(y_test_proba['y_test_class_actual'], y_test_proba['y_hat_test_proba'])

plot PR curve

plt.plot(recall, precision, marker='.', label='Logistic')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.legend()

plt.title('PR curve');

Calculate PR AUC

```
auc_pr = auc(recall, precision)

print(auc_pr)
```

Applying the Model - Scorecard Creation

STEP 18:

Using summary table create the scorecard table and reset the index of dataframe and then create a new column, called 'Original feature name', which contains the value of the 'Feature name' column, up to the column symbol

Code:

We create a new dataframe with one column. Its values are the values from the 'reference_categories' list. We name it 'Feature name'.

```
df_ref_categories = pd.DataFrame(ref_categories, columns = ['Feature name'])
```

We create a second column, called 'Coefficients', which contains only 0 values.

```
df_ref_categories['Coefficients'] = 0

print(df_ref_categories)
```

Concatenates two dataframes.

```
df_scorecard = pd.concat([summary_table, df_ref_categories])
```

We reset the index of a dataframe.

```
df_scorecard.reset_index(inplace = True)

print(df_scorecard)

df_scorecard['Original feature name'] = df_scorecard['Feature name'].str.split(':').str[0]

print(df_scorecard)
```

STEP 19:

Define the min and max thresholds for our scorecard min_score = 300 & max_score = 850

a) Calculate the sum of the minimum coefficients of each category within the original featu re name

b) Calculate the sum of the maximum coefficients of each category within the original featu re name

c) Create a new columns that has the imputed calculated score based on the multiplication of the coefficient by the ratio of the difference between maximum & minimum score and m aximum & minimum sum of coefficients

d) Update the calculated score of the Intercept (i.e. the default score for each loan)

e) Round the values of the 'Score - Calculation' column and store them in a new column

f) We'll evaluate based on the rounding differences of the minimum category within each Or iginal Feature Name

Code:

```
min_sum_coef = df_scorecard.groupby('Original feature name')['Coefficients'].min().sum()

max_sum_coef = df_scorecard.groupby('Original feature name')['Coefficients'].max().sum(
)

df_scorecard['Score - Calculation'] = df_scorecard['Coefficients'] * (max_score - min_score
) / (max_sum_coef - min_sum_coef)

df_scorecard.loc[0, 'Score - Calculation'] = ((df_scorecard.loc[0,'Coefficients'] - min_sum_c
oef) / (max_sum_coef - min_sum_coef)) * (max_score - min_score) + min_score

df_scorecard['Score - Preliminary'] = df_scorecard['Score - Calculation'].round()

print(df_scorecard)
```

STEP 20:

First create a transformed test set through our WoE_Binning custom class and insert an Inte rcept column in its beginning to align with the of rows in scorecard

Code:

```
df_scorecard['Score - Final'] = df_scorecard['Score - Preliminary']

df_scorecard.loc[0, 'Score - Final'] = 588

print(df_scorecard)

X_test_woe_transformed = woe_transform.fit_transform(X_test)

X_test_woe_transformed.insert(0, 'Intercept', 1)

X_test_woe_transformed.head()
```

STEP 21:

We can see that the test set has 7 less columns than the rows in scorecard due to the reference categories since the reference categories will always be scored as 0 based on the s corecard, it is safe to add these categories to the end of test set with 0 values

Code:

get the list of our final scorecard scores

```
scorecard_scores = df_scorecard['Score - Final']
```

check the shapes of test set and scorecard before doing matrix dot multiplication

```
print(X_test_woe_transformed.shape)

print(scorecard_scores.shape)
```

Need to reshape scorecard scores so that it is (55,1) to allow for matrix dot multiplication

```
X_test_woe_transformed = pd.concat([X_test_woe_transformed, pd.DataFrame(dict.fromkeys(ref_categories, [0] * len(X_test_woe_transformed)),  index = X_test_woe_transformed.index)], axis = 1)

scorecard_scores = scorecard_scores.values.reshape(55, 1)
```

```
print(X_test_woe_transformed.shape)

print(scorecard_scores.shape)
```

STEP 22:

Matrix dot multiplication of test set with scorecard scores

Code:

```
y_scores = X_test_woe_transformed.dot(scorecard_scores)

print(y_scores)
```

Final Output:

ID	credit_score
11584	*552*
31809	*576*
44571	*514*
5595	*552*
18657	*552*
3662	*465*