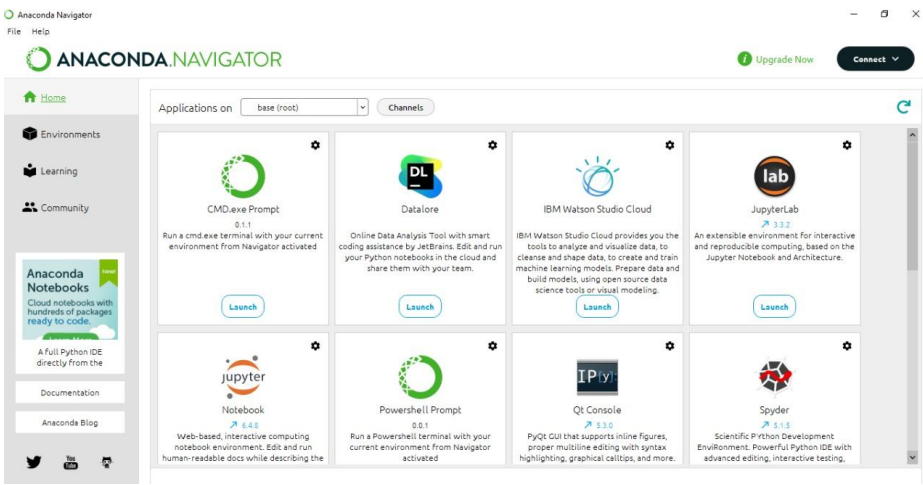
Screenshots

Date	**24 November 2022**
Team ID	**PNT2022TMID30218**
Project Name	**Smart Lender - Applicant Credibility Prediction For Loan Approval**

Pre - Requisites



Visualizing And Analyzing The Data

Importing The Libraries

```
[3] import numpy as np
    import pandas as ps
    import pandas as pd
    import sklearn
    import matplotlib.pyplot as plt
    import seaborn as sns
    import pickle
    from sklearn.preprocessing import StandardScaler
    from imblearn.combine import SMOTETomek
    from sklearn.neighbors import KNeighborsClassifier
```

Reading The Dataset

```
[4] data = ps.read_csv(r"/content/LOAN_PREDICTION.csv")
```

```
[5] data.head()
```

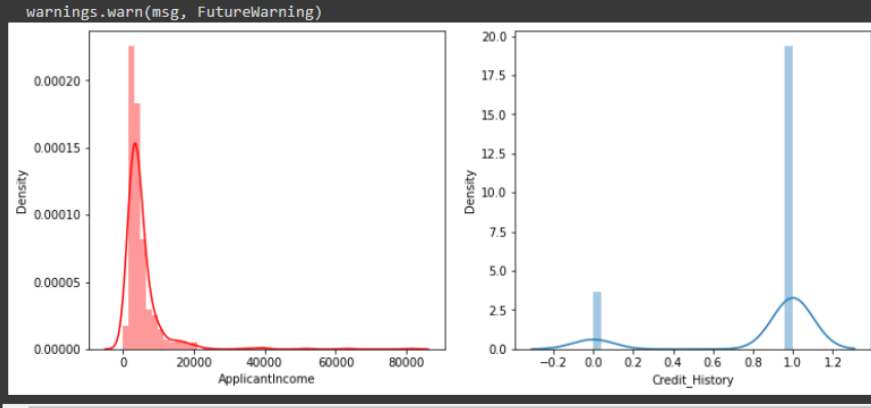
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Uni-Variate Analysis

Uni-Variate Analysis

```
[73] plt.figure(figsize=(12,5))
     plt.subplot(121)
     sns.distplot(data['ApplicantIncome'], color='r')
     plt.subplot(122)
     sns.distplot(data['Credit_History'])
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wil
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wil
  warnings.warn(msg, FutureWarning)
```

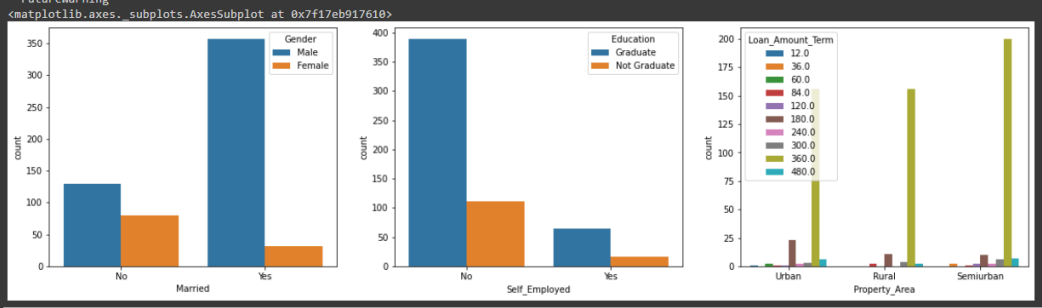


Bivariate Analysis

Bi-Variate Analysis

```
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(data['Married'], hue=data['Gender'])
plt.subplot(132)
sns.countplot(data['Self_Employed'], hue=data['Education'])
plt.subplot(133)
sns.countplot(data['Property_Area'], hue=data['Loan_Amount_Term'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f17eb917610>
```

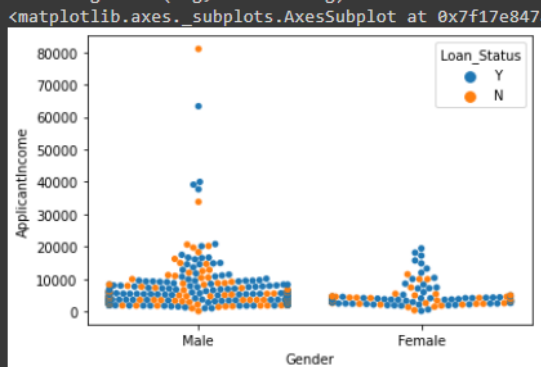


Multivariate Analysis

Multivariate Analysis

```
[75] sns.swarmplot(data['Gender'],data['ApplicantIncome'], hue= data['Loan_Status'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fol
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 67.5% of the
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 33.0% of the
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f17e847aa50>
```



Descriptive Analysis

Descriptive Analysis

```
[76] data.describe()
```

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	599.000000	614.000000	614.000000	592.000000	600.00000	564.000000
mean	0.762938	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	1.015216	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	0.000000	150.000000	0.000000	9.000000	12.00000	0.000000
25%	0.000000	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	0.000000	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	2.000000	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	3.000000	81000.000000	41667.000000	700.000000	480.00000	1.000000

Data Pre-Processing

Checking For Null Values

Checking For Null Values

```
[77] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    float64
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(5), int64(1), object(7)
memory usage: 62.5+ KB
```

```
[78] data.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

✓ 0s completed at

Handling Categorical Values

Handling Categorical Values

```
[83] data.select_dtypes(include='object').columns
```

```
Index(['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area',
       'Loan_Status'],
      dtype='object')
```

```
[84] data['Gender'].unique()
```

```
array(['Male', 'Female'], dtype=object)
```

```
[85] data['Gender'].replace({'Male':1,'Female':0},inplace=True)
```

```
[86] data['Married'].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

```
[87] data['Married'].replace({'Yes':1,'No':0},inplace=True)
```

```
[88] data['Dependents'].unique()
```

```
array([0., 1., 2., 3.])
```

```
[89] data['Dependents'].replace({'0':0,'1':1,'2':2,'3':3},inplace=True)
```

```
[93] data['Property_Area'].replace({'Urban':2,'Rural':0,'Semiurban':1},inplace=True)
```

```
[94] data['Loan_Status'].unique()
```

```
array(['Y', 'N'], dtype=object)
```

```
[95] data['Loan_Status'].replace({'Y':1,'N':0},inplace=True)
```

```
[96] data['Education'].unique()
```

```
array(['Graduate', 'Not Graduate'], dtype=object)
```

```
[97] data['Education'].replace({'Graduate':1,'Not Graduate':0},inplace=True)
```

```
[98] data['CoapplicantIncome']=data['CoapplicantIncome'].astype("int64")
     data['LoanAmount']=data['LoanAmount'].astype("int64")
     data['Loan_Amount_Term']=data['Loan_Amount_Term'].astype("int64")
     data['Credit_History']=data['Credit_History'].astype("int64")
```

Balancing The Dataset

Balancing The Dataset

```
[100] smote = SMOTETomek(0.90)
```

```
/usr/local/lib/python3.7/dist-packages/imblearn/utils/_validation.py:591: FutureWarning: Pass sampling_strategy=0.9 as keyword args. From version 0
  FutureWarning,
```

```
[101] y = data['Loan_Status']
      x = data.drop(columns=['Loan_Status'],axis=1)
```

```
[102] x_bal,y_bal = smote.fit_resample(x,y)
```

```
[103] print(y.value_counts())
      print(y_bal.value_counts())
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
1    355
0    312
Name: Loan_Status, dtype: int64
```

Scaling The Data

Scaling The Data

```
[104] sc = StandardScaler()
      x_bal = sc.fit_transform(x_bal)
```

```
[105] x_bal = pd.DataFrame(x_bal)
```

```
[106] x_bal.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	0.534007	-1.179271	-0.749348	0.621848	-0.320124	0.081803	-0.528345	-0.273023	0.278859	0.584852	1.348301
1	0.534007	0.847981	0.278126	0.621848	-0.320124	-0.125702	-0.004684	-0.170189	0.278859	0.584852	-1.232235
2	0.534007	0.847981	-0.749348	0.621848	3.123790	-0.385166	-0.528345	-0.967153	0.278859	0.584852	1.348301
3	0.534007	0.847981	-0.749348	-1.608111	-0.320124	-0.453515	0.290483	-0.273023	0.278859	0.584852	1.348301
4	0.534007	-1.179271	-0.749348	0.621848	-0.320124	0.106553	-0.528345	-0.003083	0.278859	0.584852	1.348301

```
[107] data.drop("Property_Area",axis=1,inplace=True)
      data.drop("Loan_Amount_Term",axis=1,inplace=True)
```

```
[108] x = data.iloc[:, :-1]
      y = data['Loan_Status']
```

```
[109] x
```

Splitting Data Into Train And Test

```
[115] from sklearn.model_selection import train_test_split
      x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=42)
```

Model Building

Decision Tree Model

Decision Tree Model

```
[116] from sklearn.tree import DecisionTreeClassifier
      a = DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
      a.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
[117] Y= a.predict(x_test)
      Y
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1])
```

```
[118] y_predict_train =a.predict(x_train)
```

```
[119] from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

```
[120] print('Testing accuracy :  ',accuracy_score(y_test,Y))
      print("Training accuracy : ",accuracy_score(y_train,y_predict_train))
```

```
Testing accuracy :   0.7389162561576355
Training accuracy :  0.8394160583941606
```

Random Forest Model

Random Forest Model

```
[121] from sklearn.ensemble import RandomForestClassifier
      m = RandomForestClassifier(n_estimators=10,criterion='entropy')
      m.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
[122] ypredict= m.predict(x_test)
      ypredict
```

```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1])
```

```
[123] ypredict_train = m.predict(x_train)
```

```
[124] from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
      print('Testing accuracy : ', accuracy_score(y_test,ypredict))
      print('Training accuracy : ',accuracy_score(y_train,ypredict_train))
```

```
Testing accuracy :  0.7192118226600985
Training accuracy :  0.9902676399026764
```

KNN Model

KNN Model

```
[125] S = KNeighborsClassifier()
      S.fit(x_train,y_train)

      KNeighborsClassifier()

[126] y1=S.predict(x_test)
      y2=S.predict(x_train)

[127] kNN = KNeighborsClassifier(n_neighbors=7)
      kNN.fit(x_train, y_train)
      print(kNN.predict(x_test))

      [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
       1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0
       1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
       0 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1
       1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
       1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1]

[128] print('Testing accuracy : ',accuracy_score(y_test,y1))
      print('Training accuracy : ',accuracy_score(y_train,y2))

      Testing accuracy :  0.5862068965517241
      Training accuracy :  0.7445255474452555
```

Double-click (or enter) to edit

Xgboost Model

Xgboost Model

```
[129] from sklearn.ensemble import  GradientBoostingClassifier
      from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
      V = GradientBoostingClassifier()
      V.fit(x_train,y_train)

      GradientBoostingClassifier()

[130] yp=V.predict(x_test)
      ypr=V.predict(x_train)

[131] print('Testing accuracy : ',accuracy_score(y_test,yp))
      print('Training accuracy : ',accuracy_score(y_train,ypr))

      Testing accuracy :  0.7438423645320197
      Training accuracy :  0.8978102189781022
```

Compare The Model

Decision Tree Model

```
decisionTreeClassifier(x_train, x_test, y_train, y_test)

DecisionTreeClassifier
Confusion matrix
[[ 35  37]
 [ 26 105]]
Classification report
              precision    recall  f1-score   support

           0       0.57      0.49      0.53        72
           1       0.74      0.80      0.77       131

    accuracy                           0.69       203
   macro avg       0.66      0.64      0.65       203
weighted avg       0.68      0.69      0.68       203

Testing accuracy :  0.6896551724137931
Training accuracy :  1.0
```

Random Forest Model

```
randomForestClassifier(x_train, x_test, y_train, y_test)

RandomForestClassifier
Confusion matrix
[[ 36  36]
 [ 10 121]]
Classification report
              precision    recall  f1-score   support

           0       0.78      0.50      0.61        72
           1       0.77      0.92      0.84       131

    accuracy                           0.77       203
   macro avg       0.78      0.71      0.73       203
weighted avg       0.77      0.77      0.76       203

Testing accuracy :  0.7733990147783252
Training accuracy :  1.0
```

KNN Model

```
kneighborsClassifier(x_train, x_test, y_train, y_test)

KNeighborsClassifier
Confusion matrix
[[ 30  42]
 [ 10 121]]
Classification report
              precision    recall  f1-score   support

           0       0.75      0.42      0.54        72
           1       0.74      0.92      0.82       131

    accuracy                           0.74       203
   macro avg       0.75      0.67      0.68       203
weighted avg       0.75      0.74      0.72       203

Testing accuracy :  0.5862068965517241
Training accuracy :  0.7445255474452555
```

Xgboost Model

```
xgboost(x_train, x_test, y_train, y_test)

Gradient BoostingClassifier
Confusion matrix
[[ 29  43]
 [  9 122]]
Classification report
              precision    recall  f1-score   support

           0       0.76      0.40      0.53        72
           1       0.74      0.93      0.82       131

    accuracy                           0.74       203
   macro avg       0.75      0.67      0.68       203
weighted avg       0.75      0.74      0.72       203

Testing accuracy :  0.7438423645320197
Training accuracy :  0.8978102189781022
```

Save The Model

```python
[136] #saving the model
      import pickle
      filename ='prediction_loan.sav'
      pickle.dump(m,open(filename,"wb"))
```

Application Building

```python
if request.method == "POST":
    gender=request.form['gender']
    married=request.form['married']
    dependents=request.form['Dependents']
    education=request.form['education']
    self_employment=request.form['employment']
    applicant_income=request.form['income']
    co_applicant=request.form['coincome']
    Loan_Amount=request.form['loan']
    credit_history=request.form['history']
    if(gender=="Male"):
        gender=1
    else:
        gender=0
    if(married=="Yes"):
        married=1
    else:
        married=0
    if(education=="Graduate"):
        education=0
    else:
        education=1
    if(self_employment=="Yes"):
        self_employment=1
    else:
        self_employment=0
    S=[[gender,married,dependents,education,self_employment,applicant_income,co_applicant,Loan_Amount,credit_history]]
    ans=model.predict(S)
    print(ans)
    if (ans==1):
        ans="yes"
        print("Congratulations your eligble for this Loan")
        return render_template('result.html', result="Congratulations!", result1="you are eligble for this Loan")
    else:
```