Predicting the Loan Status.

Load Application Status Prediction is a task that can be done based on historical information of the customer and bank. By checking the dataset already existed regarding the status of the Load Application and creating a model will help us to Predict the further Loan Application Status.

In this post, we are going to build a model that can predict whether the loan of the applicant will be approved or not on the basis of the details provided in the dataset. Dataset includes details of applicants who have applied for loan. The dataset includes details like credit history, loan amount, their income, dependents etc. Let's start.

Dataset:

We have a dataset with several features. The dataset comprises of 2 things: Independent variables and dependent variables.

- The **independent variable** is the cause. Its value is independent of other variables in your study.
- The **dependent variable** is the effect. Its value depends on changes in the independent variable.

Below is the list of independent and dependent variables in our dataset: -

Independent Variables:

- Loan_ID
- Gender
- Married
- Dependents
- Education
- Self_Employed
- ApplicantIncome
- CoapplicantIncome
- Loan_Amount
- Loan_Amount_Term
- Credit History
- Property_Area

Dependent Variable (Target Variable):

- Loan_Status

Problem Definition: We need to build a model that can predict whether the loan of the applicant will get approved or not on the basis of the details provided.

Importing Required Libraries:

We started the project by importing basic libraries that are required to start a project on python 3 notebook.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

As seen above I have imported below libraries and creates instances for all which helps to perform different tasks in my project:

1. **Pandas**: Helps us with the data analysis.
2. **NumPy**: Helps us in performing several mathematical operations.
3. **Matplotlib**.pyplot: Helps us with visualization for 2D plots array.
4. **Seaborn**: Which is used for data visualization and exploratory data analysis.
5. **Warnings**: Warnings are provided to warn the developer of situations that aren't necessarily exceptions. Usually, a warning occurs when there is some obsolete of certain programming elements, such as keyword, function or class, etc. A warning in a program is distinct from an error.

   Let's start building a machine learning model for loan application prediction status. These are the few steps that we will follow while handling the problem: -

   1. Importing dataset and understanding the problem.
   2. EDA.
   3. Data Cleansing
   4. Feature Engineering
   5. Encoding
   6. Split the data into training and test data sets.
   7. Model Selection
   8. Model Validation
   9. Interpret the result
   10. Saving Model

Importing the dataset and understanding the problem:

After importing necessary libraries, we imported the dataset, on which we need to work using the pandas read function. The dataset is in csv file so we used **pd.read_csv** to fetch the dataset.

```
loan_data2 = pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/loan_prediction.csv')
loan_data2.head(5)
```

ried	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Above is the image showing command to import the variable and saved the dataset to an instance. Also, the above dataset shows first 5 lines of our dataset. The last column of the dataset that is Loan_Status is our target variable which is a categorical column.

```
loan_data.shape
```

```
(614, 13)
```

The above image shows the number of rows, which is ***614*** and number of columns which is ***13*** in our dataset.

Once we checked the dataset, we now checked the target variable which is loan status: a-

```
loan_data['Loan_Status'].unique()
```

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

From the above code we have identified that we are having a categorical output column consists of two values or categories, Yes and No. Hence, we have to build a classification model to predict the loan_satatus. We have 12 independent columns which helps us in predicting the model.

After checking data will **check description** and get to know minimum value, max value, standard deviation etc.:

```
loan_data.describe()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.0	614.000000
mean	0.838762	0.657980	0.842020	0.218241	0.237785	4617.111564	1419.702231	137.365635	360.0	0.855049
std	0.421752	0.484971	1.120531	0.413389	0.534737	2479.851729	1624.605892	55.779749	0.0	0.352339
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	9.000000	360.0	0.000000
25%	1.000000	0.000000	0.000000	0.000000	0.000000	2877.500000	0.000000	100.250000	360.0	1.000000
50%	1.000000	1.000000	0.000000	0.000000	0.000000	3812.500000	1188.500000	128.000000	360.0	1.000000
75%	1.000000	1.000000	2.000000	0.000000	0.000000	5795.000000	2297.250000	164.750000	360.0	1.000000
max	2.000000	2.000000	4.000000	1.000000	2.000000	10171.250000	5743.125000	261.500000	360.0	1.000000

Describe functions gives us the values of mean, standard deviation, 25th percentile, 75th percentile etc. mathematical values for each column of the dataset.

Checking data types our dataset:

Now, we are checking the datatypes of our dataset: -

```
loan_data.dtypes
```

```
Loan_ID              object
Gender               object
Married              object
Dependents           object
Education            object
Self_Employed        object
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
Loan_Status          object
dtype: object
```

Above we can check that we have 8 columns in Object datatype non-numeric and 5 columns as int datatype numeric form. We need to encode the dataset as our machine algorithms doesn't understand the object or string data.

Checking for the Null Values:-

Checking for the null values and handling them accordingly using mean, median and mode.

```
null_values = 0
columns = []
print("Checking for null values:\n")
for i in col:
    if loan_data[i].isnull().sum() != 0:
        print(i,"Column is having null values:",loan_data[i].isnull().sum())
        null_values = null_values + 1
        columns.append(i)
    else:
        print(i,"Column is not having null values:",loan_data[i].isnull().sum())
print("\n")
print("Total columns with null values are",null_values)
print("Columns are",columns)
```

```
Checking for null values:

Loan_ID Column is not having null values: 0
Gender Column is having null values: 13
Married Column is having null values: 3
Dependents Column is having null values: 15
Education Column is not having null values: 0
Self_Employed Column is having null values: 32
ApplicantIncome Column is not having null values: 0
CoapplicantIncome Column is not having null values: 0
LoanAmount Column is having null values: 22
Loan_Amount_Term Column is having null values: 14
Credit_History Column is having null values: 50
Property_Area Column is not having null values: 0
Loan_Status Column is not having null values: 0


Total columns with null values are 7
Columns are ['Gender', 'Married', 'Dependents', 'Self_Employed', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']
```

After Handling null values,

```
loan_data.isnull().sum()
```

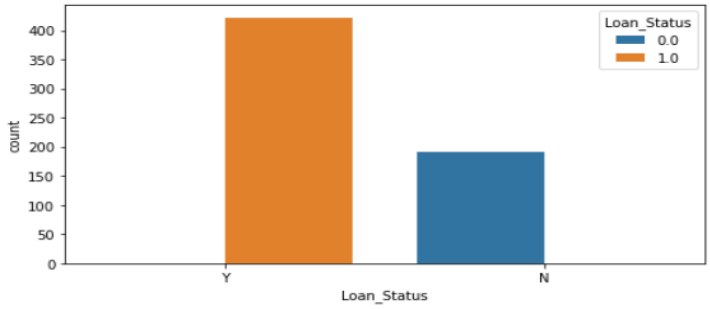
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

We, finally have removed all the null values from the dataset.

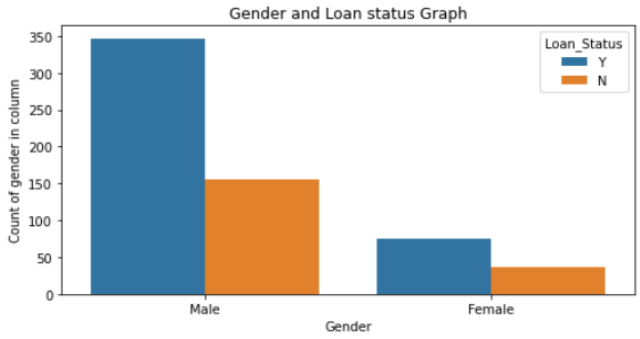
Visualization:

We have plotted the target variable that is Loan_Status, we can check that the column is having categorical data and also number of Yes are greater than No, which states that our data is having class imbalancing.

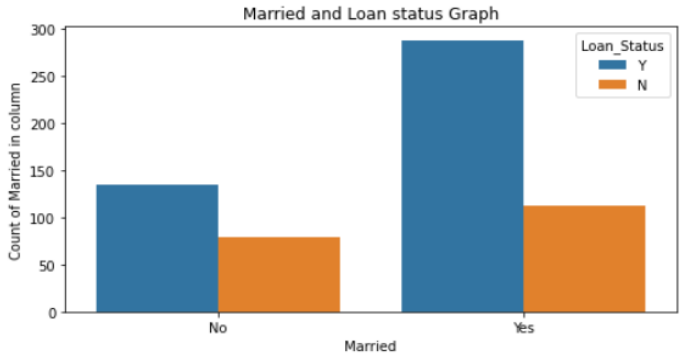
```
plt.figure(figsize=(8,4))
sns.countplot(loan_data2['Loan_Status'], hue = loan_data['Loan_Status'])
plt.show()
```



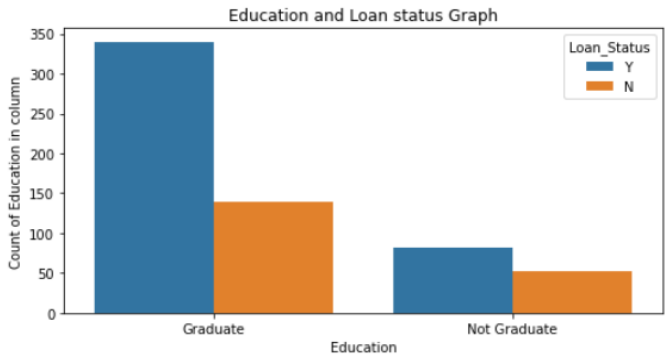

```
plt.figure(figsize=(8,4))
sns.countplot(loan_data['Gender'], hue = loan_data['Loan_Status'])
plt.title("Gender and Loan status Graph")
plt.ylabel("Count of gender in column")
plt.show()
```



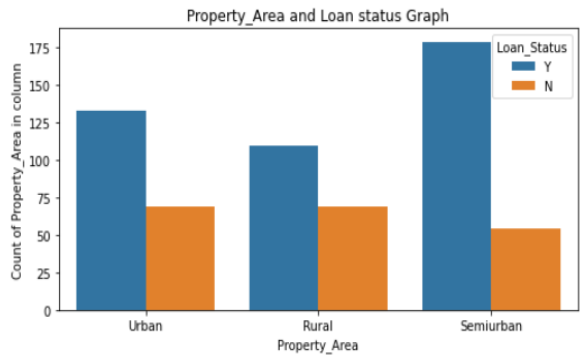

```
plt.figure(figsize=(8,4))
sns.countplot(loan_data['Married'], hue = loan_data['Loan_Status'])
plt.title("Married and Loan status Graph")
plt.ylabel("Count of Married in column")
plt.show()
```




```
plt.figure(figsize=(8,4))
sns.countplot(loan_data['Education'], hue = loan_data['Loan_Status'])
plt.title("Education and Loan status Graph")
plt.ylabel("Count of Education in column")
plt.show()
```

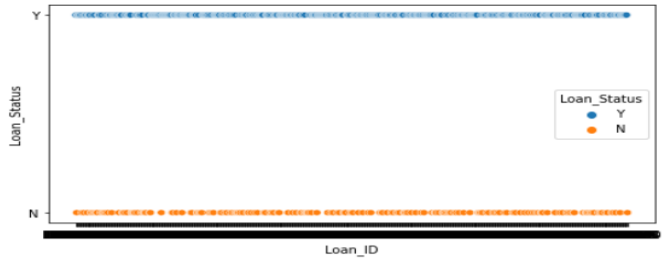



```
plt.figure(figsize=(8,4))
sns.countplot(loan_data['Property_Area'], hue = loan_data['Loan_Status'])
plt.title("Property_Area and Loan status Graph")
plt.ylabel("Count of Property_Area in column")
plt.show()
```

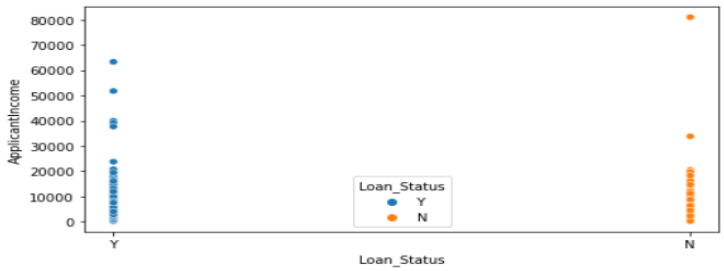


Few count plots used to analyze the nominal columns. These columns represents the count of several independent variables and how many of the loans are approve or not.

```
plt.figure(figsize=(8,4))
sns.scatterplot(x = loan_data['Loan_ID'], y = loan_data['Loan_Status'], hue = loan_data['Loan_Status'], data = loan_data)
plt.show()
```

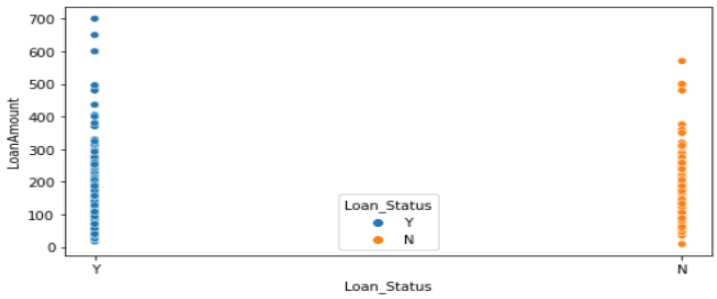



```
plt.figure(figsize=(8,4))
sns.scatterplot(y = loan_data['ApplicantIncome'], x = loan_data['Loan_Status'], hue = loan_data['Loan_Status'], data = loan_data)
plt.show()
```




```
plt.figure(figsize=(8,4))
sns.scatterplot(y = loan_data['CoapplicantIncome'], x = loan_data['Loan_Status'], hue = loan_data['Loan_Status'], data = loan_dat
plt.show()
```


```
plt.figure(figsize=(8,4))
sns.scatterplot(y = loan_data['LoanAmount'], x = loan_data['Loan_Status'], hue = loan_data['Loan_Status'], data = loan_data)
plt.show()
```




```
plt.figure(figsize=(8,4))
sns.scatterplot(y = loan_data['Credit_History'], x = loan_data['Loan_Status'], hue = loan_data['Loan_Status'], data = loan_data)
plt.show()
```

Data Encoding:-

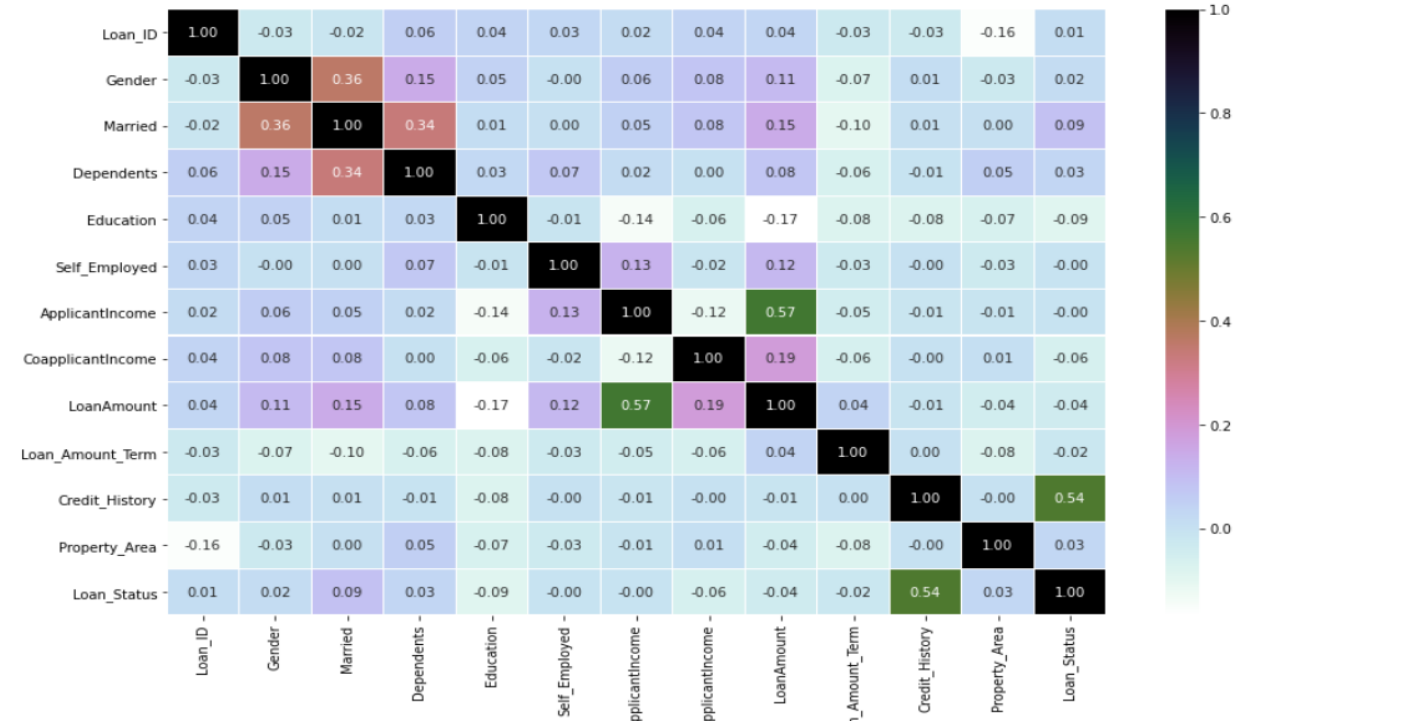
As we can check that we have a dataset where mixed types of data are present, so we need to convert the data into numeric. So, we used ordinal encoder to get the job done. We can call ordinal encoder from the sklearn.preprocessing.

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
for i in col:
    if loan_data[i].dtypes == 'object':
        loan_data[i] = enc.fit_transform(loan_data[i].values.reshape(-1,1))
```

Checking Correlation between Feature variable and Target Variable:

As, we have performed basic action on the dataset we will now check the collinearity of the dataset. Collinearity shows the relation of independent variables with the target variable. Same can be checked using a heatmap (multivariate analysis).

```
plt.figure(figsize=(15,9))
sns.heatmap(loan_data.corr(),annot=True,linewidths=0.1,fmt='0.2f',cmap = 'cubehelix_r')
plt.show()
```



The columns in the above heatmap shows the relation of the variables with each other and with the target variable. Dark color represents that the columns are highly corelated with each other. Light color represents the opposite. We can see that few independent columns are correlated with other independent columns which states that we are having some multicollinearity.

Also, we can check that credit history column is having a very high correlation with the loan_satatus (Target Variable).

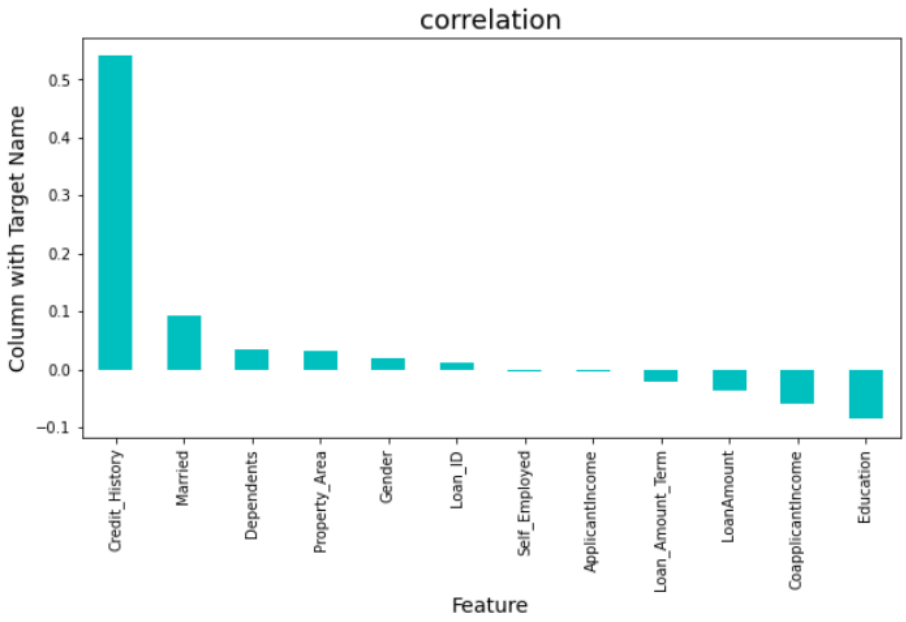
Below, we have arranged all the columns with their correlation with the target variable.

```
loan_data.corr()['Loan_Status'].sort_values(ascending=False)
```

```
Loan_Status           1.000000
Credit_History        0.540483
Married               0.091478
Dependents            0.034111
Property_Area         0.032112
Gender                0.017987
Loan_ID               0.011773
Self_Employed        -0.003700
ApplicantIncome      -0.004710
Loan_Amount_Term     -0.020974
LoanAmount           -0.036416
CoapplicantIncome    -0.059187
Education            -0.085884
Name: Loan_Status, dtype: float64
```

We can check that credit history is highly related to the target variable and education is the least related.

```
plt.figure(figsize=(10,5))
loan_data.corr()['Loan_Status'].sort_values(ascending=False).drop(['Loan_Status']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=14)
plt.ylabel('Column with Target Name',fontsize=14)
plt.title('correlation',fontsize=18)
plt.show()
```



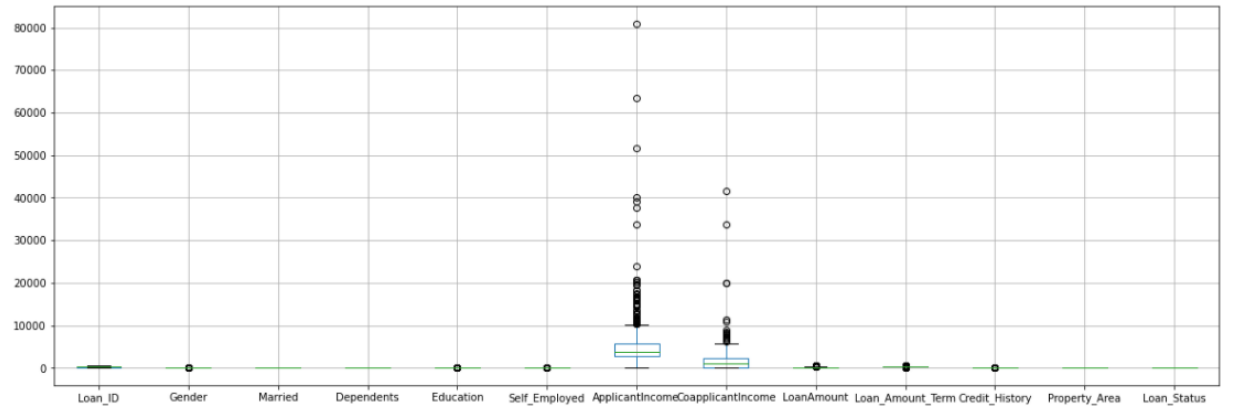
The above is the graphical representation of the correlation of independent variables with the target variable. Here also we can see that credit history is highly corelated and education is the least corelated.

Outlier Detection and Removal:

An **outlier** is a data point that is noticeably different from the rest. They represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data.

Below is the graph showing outliers present in the dataset.

```
loan_data.iloc[:,:].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



We can check that some columns in the dataset is having outliers.

1. ApplicantIncome
2. CoapplicantIncome
3. Loan AMount
4. Loan amount term

We were getting a data loss of 6.5% using z-score, so we removed outliers with the help IQR method.

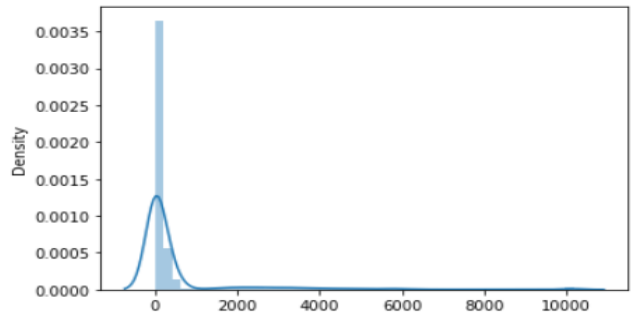
Detecting Skewness and Removal:

Skewed data is common in data science; skew is the degree of distortion from a normal distribution. Skewness refers to a distortion or asymmetry that deviates from the symmetrical bell curve, or normal distribution, in a set of data.

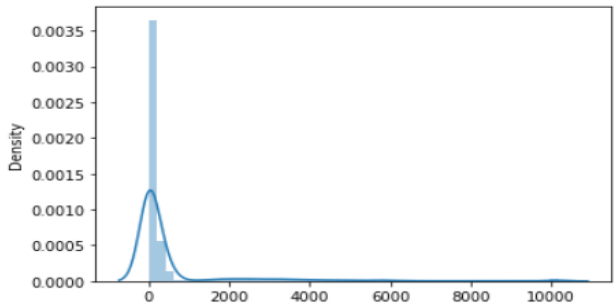
```
skew_col = []
for i in col:
    if (loan_data[i].skew() > 0.55 or loan_data[i].skew() < -0.55):
        print(loan_data[i].skew())
        print(i)
        print("Column is having skewness")
        skew_col.append(i)
    else:
        print(loan_data[i].skew())
        print(i)
        print("No skewness is present")

    plt.figure()
    sns.distplot(loan_data)
    plt.show()
```

We use a for loop to check the skewness, below is the distplot showing distribution of the data. We considered that is a column is having skewness greater than 0.55 and less than -0.55, the data is skewed in that column.

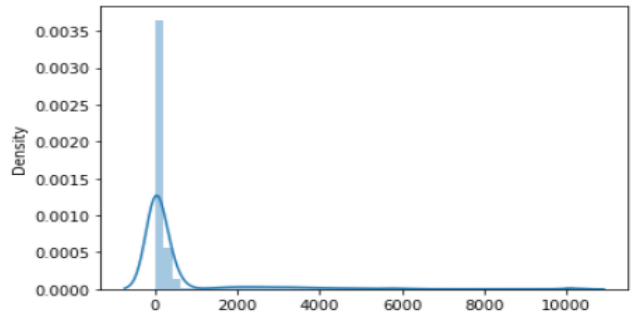
```
0.0
Loan_ID
No skewness is present
```



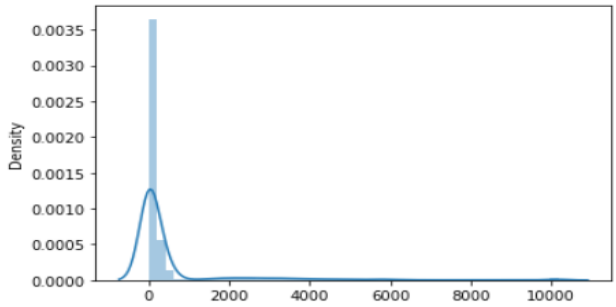

```
-0.6448502342244192
Married
Column is having skewness
```



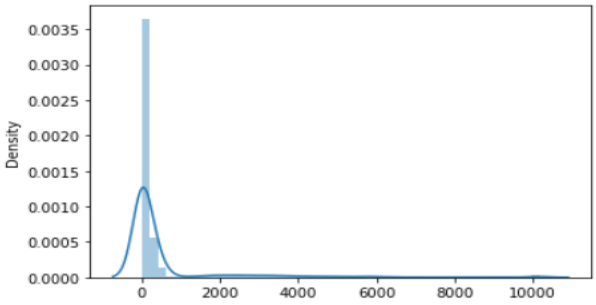

```
-1.6487952886687591
Gender
Column is having skewness
```



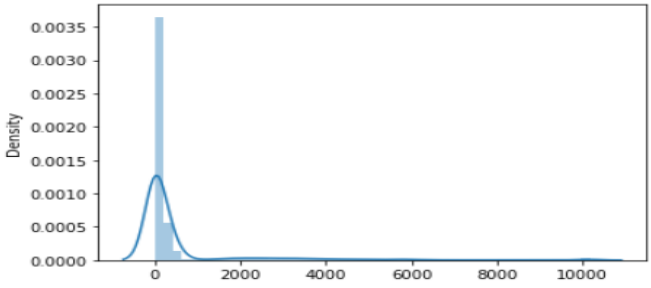

```
1.0222553436853667
Dependents
Column is having skewness
```



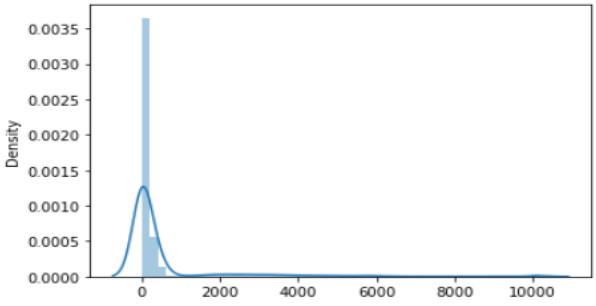

```
1.367622010164177
Education
Column is having skewness
```



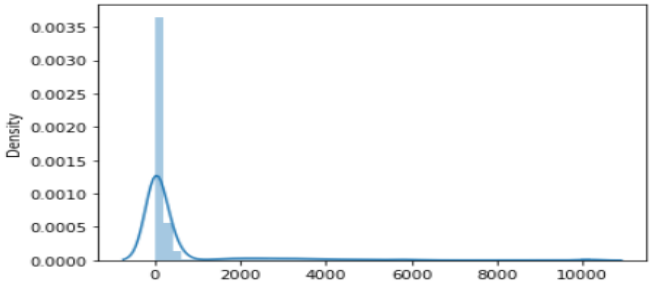

```
1.0398459567416636
ApplicantIncome
Column is having skewness
```




```
2.159796196971883
Self_Employed
Column is having skewness
```




```
1.012762761452279
CoapplicantIncome
Column is having skewness
```



Above is the graphical representation of the distribution plot, which represents that distribution of data over the column. We can see that many of the columns are skewed means that their data is not normally distributed or it is out of the range. Most of the data is skewed to right side. We will use power transform (Yeo-Johnson) to remove the skewness from the dataset.

Splitting the dataset:-

We will now split the dataset into X and Y, where X contains all the independent variable and Y contains the target variable. Please see the picture below: -

```
x = loan_data.iloc[:,0:12]
x.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	P
0	0.0	1.0	0.0	0.0	0.0	0.0	5849.0	0.0	146.412162	360.0	1.0	
1	1.0	1.0	1.0	1.0	0.0	0.0	4583.0	1508.0	128.000000	360.0	1.0	
2	2.0	1.0	1.0	0.0	0.0	1.0	3000.0	0.0	66.000000	360.0	1.0	
3	3.0	1.0	1.0	0.0	1.0	0.0	2583.0	2358.0	120.000000	360.0	1.0	
4	4.0	1.0	0.0	0.0	0.0	0.0	6000.0	0.0	141.000000	360.0	1.0	

```
y = loan_data['Loan_Status']
y.head()
```

```
0    1.0
1    0.0
2    1.0
3    1.0
4    1.0
Name: Loan_Status, dtype: float64
```

Scaling of Values:

As the range of data is varying we can use Min-Max scaler to limit the range of data between 0 to 1. Transform **features by scaling each feature to a given range**. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

```
from sklearn.preprocessing import MinMaxScaler
mc = MinMaxScaler()
x = mc.fit_transform(x)
```

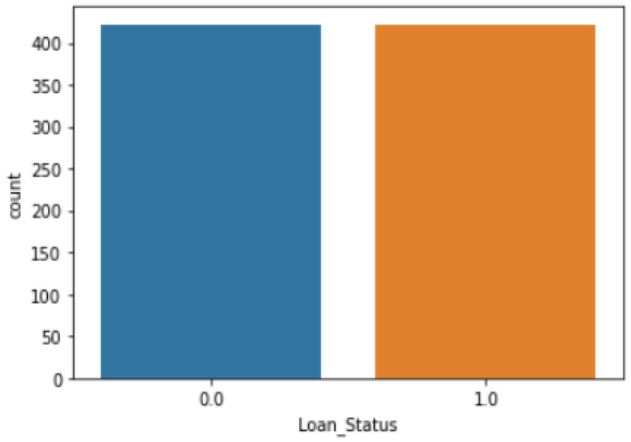
Min-max normalization method **guarantees all features will have the exact same scale**

Balancing the Classes:

As, we have checked earlier that our dataset is having an issue of classes unbalancing. We will now use SMOTE technique to balance the classes.

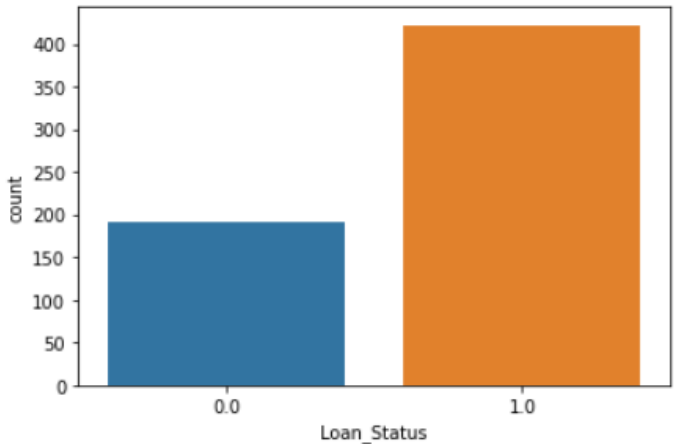
```
sns.countplot(y)
```

```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```




```
sns.countplot(y)
```

```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



MODEL TRAINING:

Here the data was fetched into the machine for making predictions. We first finding best random state for the algorithms.

```
def predict_best_state(X):
    max_acc = 0
    maxRS = 0
    for i in range(1,200):
        x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25, random_state = i)
        X.fit(x_train,y_train)
        pred = X.predict(x_test)
        acc = accuracy_score(y_test,pred)
        if acc > max_acc:
            max_acc = acc
            maxRS = i
    print("Best accuracy for",X,"is",max_acc,"at random state",maxRS)
```

```
model = [log,knear,dec_tree,nv,support]
```

```
for i in model:
    predict_best_state(i)
```

```
Best accuracy for LogisticRegression() is 0.7725118483412322 at random state 18
Best accuracy for KNeighborsClassifier() is 0.7962085308056872 at random state 130
Best accuracy for DecisionTreeClassifier() is 0.8246445497630331 at random state 96
Best accuracy for GaussianNB() is 0.7867298578199052 at random state 18
Best accuracy for SVC() is 0.8009478672985783 at random state 18
```

We can check that we are getting 82% accuracy with decision tree at random state 96. So, will proceed with decision tree. Below we can check the results for decision Tree.

```
dec_tree.fit(x_train,y_train)
pred_dec = dec_tree.predict(x_test)
print("The accuracy score for is:-",accuracy_score(y_test,pred_dec)*100,'\n')
```

```
The accuracy score for is:- 82.46445497630332
```

Here we applied the random state that we got above and try to run the model and we can check that we are getting 82% for decision tree algorithm.

Confusion Matrix:

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

```
print('Confusion Matrix:\n', confusion_matrix(y_test,pred_dec))
print('\n')
```

```
Confusion Matrix:
 [[71 11]
 [15 72]]
```

We can check that: -

1. We have 71 True positive which means, 71 predicted value matches the actual value.
2. 11 predicated values were falsely predicted. The actual value was negative but the model predicted a positive value
3. 15 predicted values were falsely predicted. The actual value was positive but the model predicted a negative value
4. 72 predicted values matches the actual value. The actual value was negative and the model predicted a negative value

Classification Report:

A classification report is **a performance evaluation metric in machine learning**. It is used to show the precision, recall, F1 Score, and support of your trained classification model.

```
print('Classification Report:\n', classification_report(y_test,pred_dec))
```

```
Classification Report:
                precision    recall  f1-score   support

           0.0       0.83      0.87      0.85        82
           1.0       0.87      0.83      0.85        87

    accuracy                           0.85       169
   macro avg       0.85      0.85      0.85       169
weighted avg       0.85      0.85      0.85       169
```

From the classification report we can say that:-

1. We are having 83% precision for predicting 0 and 87% for predicting 1.
2. Weighted harmonic mean score is 85%.

Cross Validation for Decision Tree Classifier:

Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

```
from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(dec_tree,x,y,cv=5)
print('\n')
print("Cross Validation Score for is:- \n",cv_score)
print("Mean of Cross Validation Score for is:- \n",cv_score.mean())
print("Standard Deviation of Cross Validation Score for is:- \n",cv_score.std())
print('\n')

print('Model Score',accuracy_score(y_test,pred_dec)*100)
print('Mean Score After CV',)
print("Model Score is", accuracy_score(y_test,pred_dec)-cv_score.mean())
print('\n')
```

```
Cross Validation Score for is:-
 [0.73964497 0.72781065 0.79289941 0.80473373 0.76785714]
Mean of Cross Validation Score for is:-
 0.7665891800507185
Standard Deviation of Cross Validation Score for is:-
 0.029592590694869178


Model Score 82.46445497630332
Mean Score After CV
Model Score is 0.05805536971231462
```

Here, we calculated the CV score for the decision tree and we can check that the difference between cv score mean is 0.05, which makes decision tree suitable for this problem.

Observation of the CV Score:

Minimum difference between Accuracy and Cross validation is of **Decision Tree Classifier** which, will be proven the Best Algorithm for this model.

Hyper Parameter Tuning:

Hyperparameter tuning is **choosing a set of optimal hyperparameters for a learning algorithm**. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

We used randomized search CV for improving the performance of our model.

```
from sklearn.model_selection import RandomizedSearchCV
```

```
criterion = ['gini','entropy']
splitter = ['random','best']
max_depth = [int(x) for x in np.linspace(5,30,num=6)]
min_samples_split = [5,10,15,20,100]
min_samples_leaf = [2,5,10,15]
ccp_alpha = [0.5,1.5,2.5,3.5,4.5,5.5]
max_features = [2,4,6,8,10]
max_features = ['auto','sqrt','log2']

random_grid = {
    'criterion':criterion,
    'max_features':max_features,
    'max_features':max_features,
    'splitter':splitter,
    'max_depth':max_depth,
    'min_samples_split':min_samples_split,
    'min_samples_leaf':min_samples_leaf
}
```

```
zedSearchCV(estimator=dec_tree,param_distributions=random_grid,scoring='neg_mean_squared_error',n_iter=10,cv=5,verbose=2,n_jobs=1
```

SAVING THE MODEL:-

Hence we found that Decision tree I best suitable algorithm for this model. Hence, we are finally saving this model

```
import pickle
filename = 'loan_predict_last.pkl'
pickle.dump(dec_tree, open(filename, 'wb'))
```

Pickle libraries is used to save the model.

Loading Model:

```
loaded_model = pickle.load(open('loan_predict_last_2.pkl','rb'))
result = loaded_model.score(x_test,y_test)
print(result)
```

```
0.8402366863905325
```

Conclusion:

We have successfully created a machine learning model to predict the loan status with an accuracy of 84%.

Below we can see that predicted and original values for the model.

```
conclusion = pd.DataFrame([loaded_model.predict(x_test)[:],pred_decision[:]],index=['Predicted','Original'])
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	201	202	203	204	205	206	207	208	209	210
Predicted	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0
Original	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0

2 rows × 211 columns

Thanks!