Loan Application Status Prediction

1. Problem Definition:

Loan Application Status Predictive model used to predict if the applicant is eligible for the applied loan. It is very important for the bank to predict prior granting the loan to applicant if it is beneficial and loan should be approved or not taking the risk factor in mind.

The dataset for this prediction model consists of credit history, loan amount, their income, dependents etc which used as independent variable / features to predict the loan approval status for the applicant which is our target / label. After comparing and analysing the features the model will provide most accurate results.

2. Data Analysis:

The Data Analysis process follow the flow of loading the dataset from a particular source, checking the information of data set, cleaning, transforming the data, getting the required shape of data for model building. If there are null values present in the dataset we have to find the best approach to handle null values. After doing the analysis on features we make sure the model predict the most accurate target value.

For the Loan Application Status prediction model, Data consist of features and target. On features data all the data analysis, Exploratory Data analysis, Data pre-processing will be done.

Checking the info and null value details from the data frame and filling the null value with the mean values as the null value is in the continuous features. Using the heat map to check the null values in data frame

- Loading Dataset:

```
#Loading the Dataset:
df=pd.read_csv('Loan Application Prediction.csv')
df.head()
```

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

- Checking shape of dataset:

```
df.shape
#there is 614 rows and 13 columns is availble in the data
```

```
(614, 13)
```

- Checking for the null values in dataset:

```
#checking null value from the dataframe
df.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
```

- Handling the null values:

```
#Fillinf the null values:
df['Gender']=df['Gender'].fillna(df['Gender'].mode()[0])
df['Married']=df['Married'].fillna(df['Married'].mode()[0])
df['Dependents']=df['Dependents'].fillna(df['Dependents'].mode()[0])
df['Self_Employed']=df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
df['Credit_History']=df['Credit_History'].fillna(df['Credit_History'].mode()[0])

df['LoanAmount']=df['LoanAmount'].fillna(df['LoanAmount'].median())
df['Loan_Amount_Term']=df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median())
```

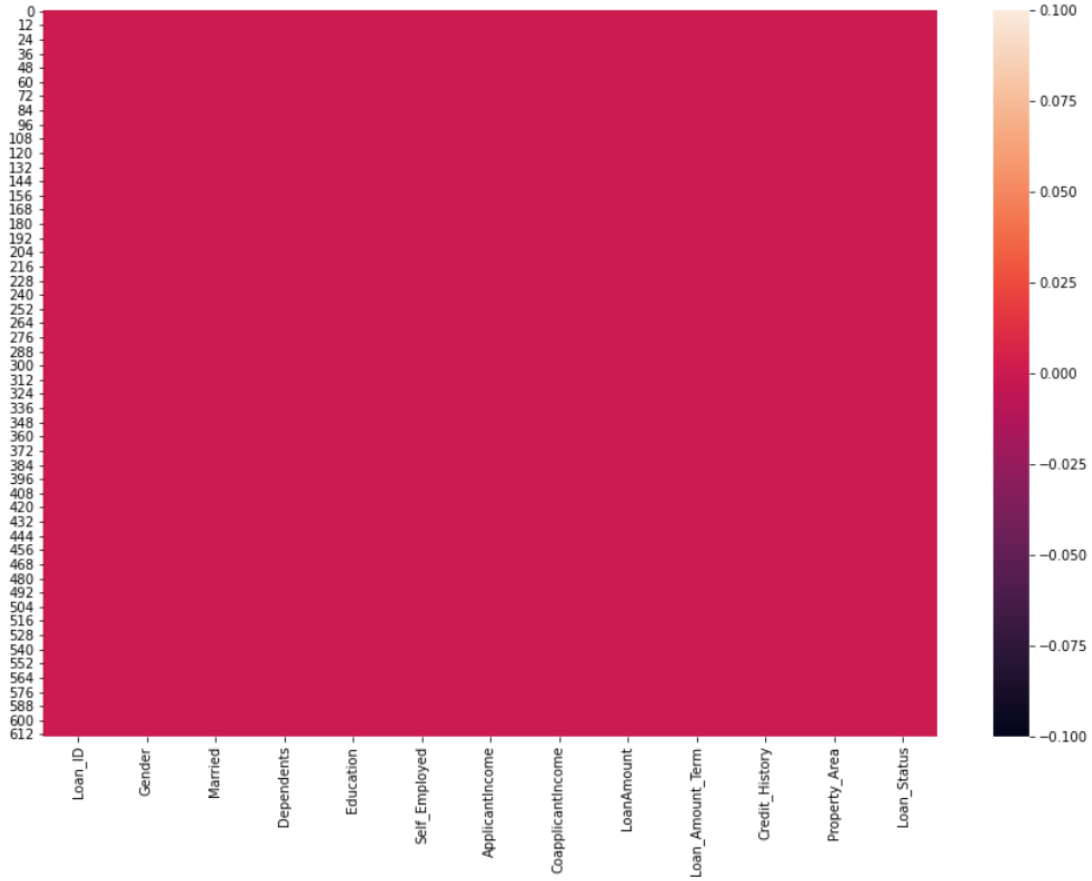
- Checking null value after handling the null value:

```
#checking null value again from the dataframe
df.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
```

- Using Heatmap to visualize null value:

```
#there is no null value available in the dataframe.cross checking the same with the heatmap.
plt.figure(figsize=(15,10))
sns.heatmap(df.isna())
plt.show()
```



- Describing the dataframe:

```
#Discribing the dataframe
df.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	145.752443	342.410423	0.855049
std	6109.041673	2926.248369	84.107233	64.428629	0.352339
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

- Transforming the categorical feature:

```
#using label transfrom to convert catagorical data in to binary data
lb=LabelEncoder()
cat_var=['Gender','Married','Education','Self_Employed','Property_Area','Loan_Status']

for i in cat_var:
    df[i]=lb.fit_transform(df[i])
```

```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	1	0	0	0	0	5849	0.0	128.0	360.0	1.0
1	LP001003	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0
2	LP001005	1	1	0	0	1	3000	0.0	66.0	360.0	1.0
3	LP001006	1	1	0	1	0	2583	2358.0	120.0	360.0	1.0
4	LP001008	1	0	0	0	0	6000	0.0	141.0	360.0	1.0

```
#using get dummies method to perform the encoding.
df=pd.get_dummies(df,columns=['Property_Area'])
```

```
df.head()
```

ıployed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Property_Area_0	Property_Area_1	Property_Area_2
0	5849	0.0	128.0	360.0	1.0	1	0	0	1
0	4583	1508.0	128.0	360.0	1.0	0	1	0	0
1	3000	0.0	66.0	360.0	1.0	1	0	0	1
0	2583	2358.0	120.0	360.0	1.0	1	0	0	1
0	6000	0.0	141.0	360.0	1.0	1	0	0	1

3. EDA Concluding Remark:

Exploratory data analysis consists of the Data visualizing, summarizing and interpreting the information that is hidden in rows and columns format. Data cleaning and data transformation has performed in data analysis.

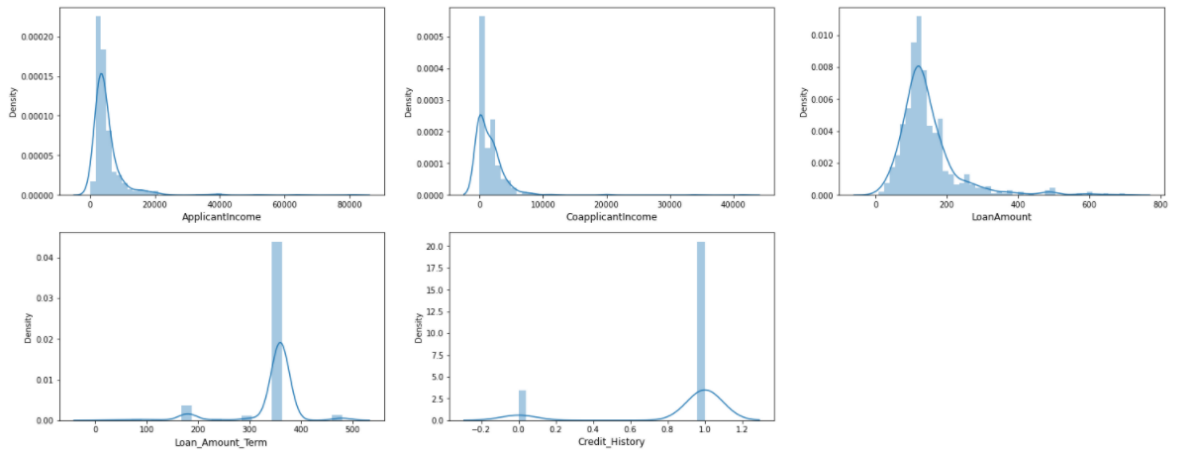
The data reduction part consists of finding the data distribution, outliers finding and deletion, checking the skewness, checking the multicollinearity issue in features and if multicollinearity is there then delete the features having such issue.

Data Distribution:

Data distribution is basically checking the features data distribution using the dist plot.Dist plot is also provide the information about the outliers

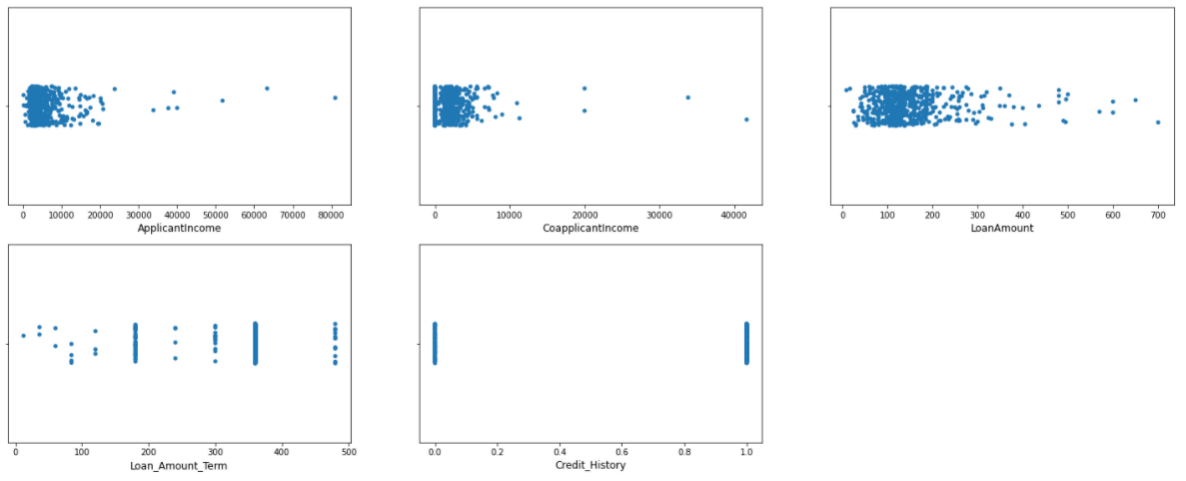
- Data distribution using distplot:

```
#checking the data distrubution using the distribution plot.
plt.figure(figsize=(25,20),facecolor='white')
pltnum=1

for column in df.iloc[:,[5,6,7,8,9]]:
    if pltnum <= 12:
        plt.subplot(4,3,pltnum)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=12)

    pltnum +=1
plt.show()
```



- Data distribution using strip plot:

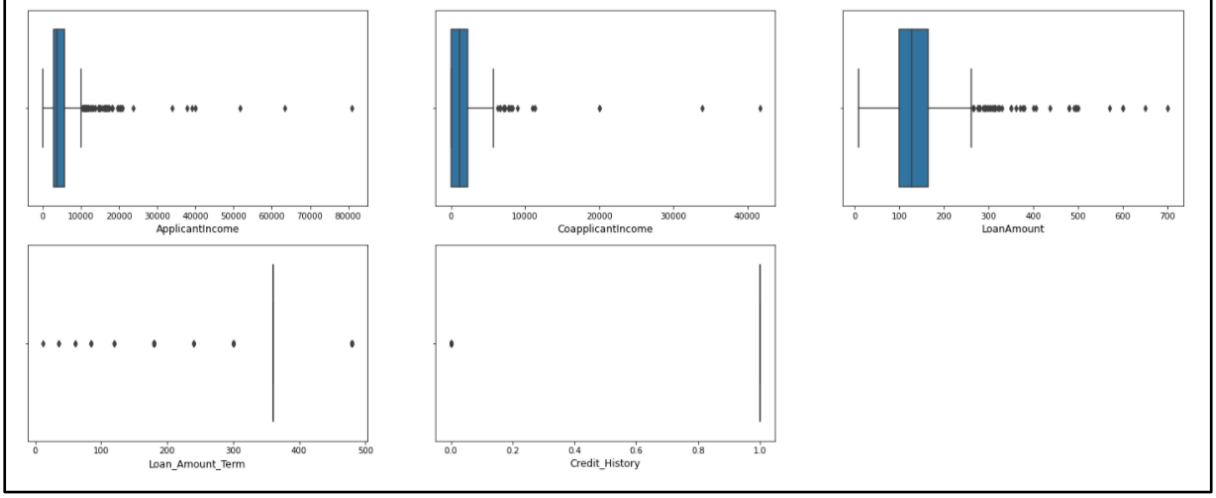
```
#checking the data distrubution using the Strip plot.
plt.figure(figsize=(25,20),facecolor='white')
pltnum=1

for column in df.iloc[:,[5,6,7,8,9]]:
    if pltnum <= 12:
        plt.subplot(4,3,pltnum)
        sns.stripplot(df[column])
        plt.xlabel(column,fontsize=12)

    pltnum +=1
plt.show()
```



After observing above dist and strip plot we can say there are few outliers present in all of the features. Now we will visualize box plot to identify the outlier region.

- Data distribution using box plot:

```
#using box plot to check the outliers in the feature.
plt.figure(figsize=(25,20),facecolor='white')
pltnum=1

for column in df.iloc[:,[5,6,7,8,9]]:
    if pltnum <= 12:
        plt.subplot(4,3,pltnum)
        sns.boxplot(df[column])
        plt.xlabel(column,fontsize=12)

    pltnum +=1
plt.show()
```



We are using Z-score to identify the outlier indexes for all features. After identifying the outlier indexes we dropped the indexes for all of the features.

- Using Z-score to check outliers:

```
#using the z-score to check the outliers.
z=np.abs(stats.zscore(df.iloc[:,[5,6,7,8,9]]))
index=np.where(z>3)[0]
```

- Dropping outlier indexes:

```
#dropping the outlier index
df=df.drop(df.index[index])
```

- Calculating data lose percentage after dropping outliers:

As the data loss percentage is under 7% i.e 6.02% so we are fine to start data pipeline process with the remaining data.

```
#Data Loss
loss_percentage=(614-577)/614*100
print(loss_percentage)
```

```
6.026058631921824
```

After the outliers deletion skewness check and removal of skewness is also important. The range of the skewness is -0.5 to +0.5. if the score is in between them then there is no skewness in the features. In max case after removing the outliers skewness also get reduced.

Skewness and Multicollinearity Check:

```
#checking the skewness and correlation between the featues and labels
df_corr=df.iloc[:,[5,6,7,8,9,10]].corr()
df_corr['Loan_Status'].sort_values(ascending=False)
```

```
Loan_Status          1.000000
Credit_History       0.560936
CoapplicantIncome    0.045009
ApplicantIncome     -0.005003
Loan_Amount_Term    -0.020291
LoanAmount          -0.030169
Name: Loan_Status, dtype: float64
```

As per above stats we can see the skewness present in Credit_History column as the value is greater then +0.5. In order to remove the skewness we will use power transformation.

- Using Power transformation to remove skewness:

Here we are using yeo-johnson method of Power Transformation to remove skewness. After removing the outliers, we will replace the scaled data of credit_history column with its actual value.

```
#performing the powertransformation to remove skweness from the column
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method = 'yeo-johnson')
```

- Replacing the scaler data for credit_history column:

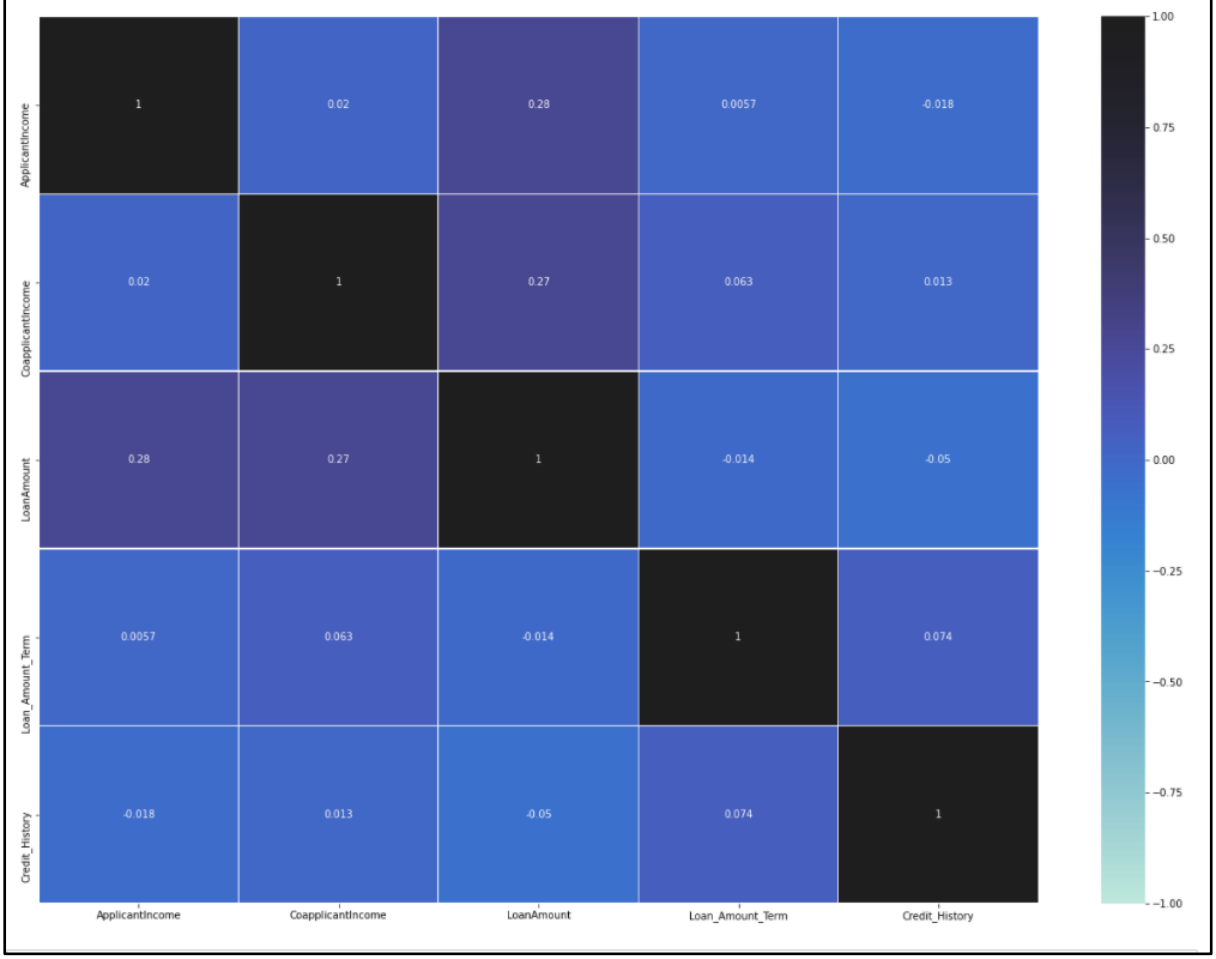
```
#replacing the Credit_History value with the power transformed value
df['Credit_History'] = scaler.fit_transform(feature.values)
df.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Statu
0	1	0	0	0	0	5849	0.0	128.0	360.0	0.418513	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	0.418513	
2	1	1	0	0	1	3000	0.0	66.0	360.0	0.418513	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	0.418513	
4	1	0	0	0	0	6000	0.0	141.0	360.0	0.418513	

The most important part is checking the multicollinearity problem between features. There is multiple method to check the multicollinearity issue between features.

The most common method is heat map and variance inflection factor. This two is commonly used to check multicollinearity. Here we are using heat map first to check if there are any multicollinearity present between features.

```
#using the heatmap to check the correlation between features.
corr_matrix=df1.iloc[:,[5,6,7,8,9]].abs().corr()

plt.figure(figsize=(22,16))

sns.heatmap(corr_matrix,vmin=-1,vmax=1,center=True,annot=True, fmt='.2g',linewidths=0.1)
plt.show()

#as we can see in heatmap there is no correlation between the features and labels
```



From the heatmap we can observe there no collinearity present between any of the features. We use VIF to verify the same for calculating vif score we have to take scalar data of the features.

- Using standard scalar for scaling the data:

```
data_var=['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']
features=df[data_var]
```

```
#using the standerd scalar for scaling the data
scalar=StandardScaler()
x_data=scalar.fit_transform(features.values)
```

```
#assigning the  scaled data to features
df[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']]=x_data
```

- Calculating vif score to check multicollinearity between features:

```
#checking feature colliniarity using VIF
vif=pd.DataFrame()
vif['features']=df1.columns
vif['vif score']=[variance_inflation_factor(x_data,i) for i in range(x_data.shape[1])]
vif
```

	features	vif score
0	ApplicantIncome	1.528906
1	CoapplicantIncome	1.364665
2	LoanAmount	1.608426
3	Loan_Amount_Term	1.022980
4	Credit_History	1.002311

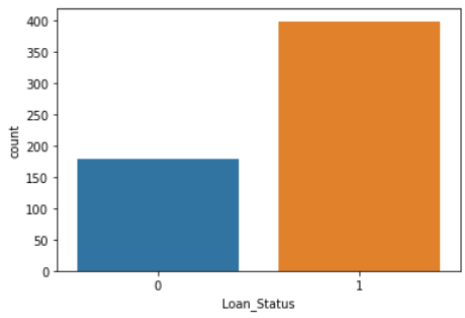
We got the VIF score for the features and none of the VIF score exceed 5 for the multicollinearity between features to be true vif should be >5. So now we can there are no multicollinearity present between any of the feature and we can proceed with the preprocessing and model building.

4. Pre-Processing Pipeline:

Pre-processing consists of the Data Cleaning, Data Transformation and Data Reduction. Data cleaning and data transformation already completed in the data analysis and EDA. Data reduction can be used to reduce the amount of data and de-creases the costs of analysis.

One major issue in the model building for classification is class imbalanced issue of target variable. If the data looks imbalanced for the target variable, then user over sampling and under sampling method to make target data balanced.

```
#plotting the count plot on target data
sns.countplot(x='Loan_Status',data=df)
plt.show()
```



Here issue of class imbalanced problem is present as 0 has 179 count and 1 has only 398 counts. We are using the over sampling to make target data balanced to solve the imbalanced issue.

- Using SMOTE to oversampling the minority class:

```
#handling class imbalanced problem by oversampling the minority class using SMOTE:
from imblearn.over_sampling import SMOTE

#using the SMOTE to solve the imbalanced problem
SM=SMOTE()
x_fit,y_fit=SM.fit_resample(x,y)

#new the issue of class imbalanced problem has been solved.
y_fit.value_counts()
```

```
1    398
0    398
Name: Loan_Status, dtype: int64
```

The Reduction of data is basically removing the unwanted features from the data frame. Like in this model the features have been removed after facing the multicollinearity issue. After all the process separating the features and target variables.

5. Building Machine Learning Models:

Building the machine-learning model is consist of training and testing the data on multiple models. The model is selected on type of target variable. If the target variable is categorical means 1 or 0 or having only categorical values then it is a classification model and if the target variable having continuous data, then it is a regression model. The insurance claim model having the categorical target data so it is a classification model.

There are multiple models to build the classification model. Using all the models one by one to fit and train the data on model.

The classification models are Logistics Classification, K-Neighbour classification, Randomforest classifier, Adaboost classifier, SVM. Using all the models to fit the data and checking the accuracy, confusion matrix and classification report of each model.

- Finding Random state accuracy:

```
#Finding best Random State:
maxAcc=0
maxRs=0

for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x_fit,y_fit,test_size=0.25,random_state=i)
    dt=LogisticRegression()
    dt.fit(x_train,y_train)
    y_pred=dt.predict(x_test)
    acc_s=accuracy_score(y_test,y_pred)
    if acc_s > maxAcc:
        maxAcc=acc_s
        maxRs=i

print(f"Best Accuracy is : {maxAcc* 100:.2f}%")
print("The Best Random State is :",maxRs)
```

```
Best Accuracy is : 83.42%
The Best Random State is : 68
```

- Splitting the data:

```
#seprating the train and test datasets with the best find random state
x_train,x_test,y_train,y_test=train_test_split(x_fit,y_fit,test_size=0.25,random_state=68)
```

- DecisionTree classifier:

```
#Using the DecisionTreeClassifier algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:
lr=LogisticRegression()
lr.fit(x_train,y_train)
y_predlr=lr.predict(x_test)

print('\n==========Outputs of Logistics Reggression==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_predlr)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_predlr,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_predlr))
```

```
==========Outputs of Logistics Reggression==========

==========Accuracy Score==========
Accuracy Score is : 83.42%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.86      0.70      0.78        81
           1       0.82      0.92      0.87       118

    accuracy                           0.83       199
   macro avg       0.84      0.81      0.82       199
weighted avg       0.84      0.83      0.83       199


=========Confusion Matrix============
[[ 57  24]
 [  9 109]]
```

- AdaBoost Classifier:

```
#Using the AdaBoostClassifier algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:
abc=AdaBoostClassifier()
abc.fit(x_train,y_train)
y_predabc=abc.predict(x_test)

print('\n==========Outputs of ADA Boost==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_predabc)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_predabc,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_predabc))
```

```
==========Outputs of ADA Boost==========

==========Accuracy Score==========
Accuracy Score is : 78.89%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.76      0.70      0.73        81
           1       0.81      0.85      0.83       118

    accuracy                           0.79       199
   macro avg       0.78      0.78      0.78       199
weighted avg       0.79      0.79      0.79       199


=========Confusion Matrix============
[[ 57  24]
 [ 18 100]]
```

- RandomForest Classifier:

```
#Using the RandomForestClassifier algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:

rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
y_predrfc=rfc.predict(x_test)

print('\n==========Outputs of Random Forest Classifier==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_predrfc)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_predrfc,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_predrfc))
```

```
==========Outputs of Random Forest Classifier==========

==========Accuracy Score==========
Accuracy Score is : 88.44%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.87      0.84      0.86        81
           1       0.89      0.92      0.90       118

    accuracy                           0.88       199
   macro avg       0.88      0.88      0.88       199
weighted avg       0.88      0.88      0.88       199


=========Confusion Matrix============
[[ 68  13]
 [ 10 108]]
```

- K-Neighbors Classifiers:

```
#Using the KNeighborsClassifier algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:

knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
y_predknn=knn.predict(x_test)

print('\n==========Outputs of KNeighborsClassifier==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_predknn)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_predknn,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_predknn))
```

```
==========Outputs of KNeighborsClassifier==========

==========Accuracy Score==========
Accuracy Score is : 81.91%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.76      0.81      0.79        81
           1       0.87      0.82      0.84       118

    accuracy                           0.82       199
   macro avg       0.81      0.82      0.81       199
weighted avg       0.82      0.82      0.82       199


=========Confusion Matrix============
[[66 15]
 [21 97]]
```

- SVC algorithm and Confusion Matrix:

```
#Using the SVC algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:

#svc=SVC(probability=True)
svc=SVC()
svc.fit(x_train,y_train)
y_predsvc=svc.predict(x_test)

print('\n==========Outputs of SVC==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_predsvc)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_predsvc,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_predsvc))
```

```
==========Outputs of SVC==========

==========Accuracy Score==========
Accuracy Score is : 83.42%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.84      0.73      0.78        81
           1       0.83      0.91      0.87       118

    accuracy                           0.83       199
   macro avg       0.84      0.82      0.82       199
weighted avg       0.83      0.83      0.83       199


=========Confusion Matrix============
[[ 59  22]
 [ 11 107]]
```

After performing the model building, checking the cross-validation score of the model to check the model is overfitting or under fitting. This is very import to perform in process of selecting the best model form all the models. The model selection is based on the difference between the accuracy and cross validation score.

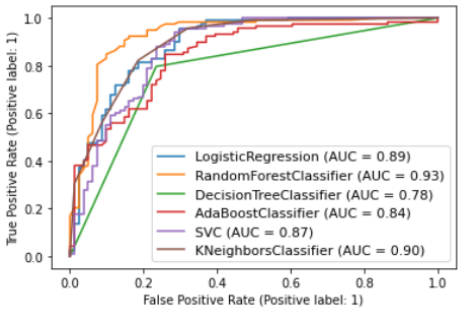
Model having least difference between accuracy and cross validation score is best model to select for the hyper parameter tuning.

```
print('\n==========Cross Validation Score of Logistices Classifier==========')
cvs_lr=cross_val_score(lr,x_fit,y_fit,cv=5)
print(f"Cross validation for the Logistices Classifier is: {cvs_lr.mean()*100:.2f}%\n")

print('\n==========Cross Validation Score of Random Forest Classifier==========')
cvs_rfc=cross_val_score(rfc,x_fit,y_fit,cv=5)
print(f"Cross validation for the Random Forest Classifier is: {cvs_rfc.mean()*100:.2f}%\n")

print('\n=============Cross Validation Score of Decision Tree====================')
cvs_dt=cross_val_score(dt,x_fit,y_fit,cv=5)
print(f"Cross validation for the Decision Tree is: {cvs_dt.mean()*100:.2f}%\n")

print('\n=============Cross Validation Score of AdaBoost Classifier=============')
cvs_abc=cross_val_score(abc,x_fit,y_fit,cv=5)
print(f"Cross validation for the AdaBoost Classifier is: {cvs_abc.mean()*100:.2f}%\n")

print('\n==================Cross Validation Score of KNN=======================')
cvs_knn=cross_val_score(knn,x_fit,y_fit,cv=5)
print(f"Cross validation for the KNN Classifier is: {cvs_knn.mean()*100:.2f}%\n")

print('\n==================Cross Validation Score of SVC=======================')
cvs_svc=cross_val_score(svc,x_fit,y_fit,cv=5)
print(f"Cross validation for the SVC Classifier is: {cvs_svc.mean()*100:.2f}%\n")
```

```
==========Cross Validation Score of Logistices Classifier==========
Cross validation for the Logistices Classifier is: 75.26%


==========Cross Validation Score of Random Forest Classifier==========
Cross validation for the Random Forest Classifier is: 82.05%


=============Cross Validation Score of Decision Tree====================
Cross validation for the Decision Tree is: 74.50%


=============Cross Validation Score of AdaBoost Classifier=============
Cross validation for the AdaBoost Classifier is: 74.25%


==================Cross Validation Score of KNN=======================
Cross validation for the KNN Classifier is: 77.39%


==================Cross Validation Score of SVC=======================
Cross validation for the SVC Classifier is: 74.25%
```

- Checking the Roc AUC Curve to analyze the models:

```
#Using the roc_auc_plot to the same.
dist=plot_roc_curve(lr,x_test,y_test)
plot_roc_curve(rfc,x_test,y_test,ax=dist.ax_)#abc svc,
plot_roc_curve(dt,x_test,y_test,ax=dist.ax_)
plot_roc_curve(abc,x_test,y_test,ax=dist.ax_)
plot_roc_curve(svc,x_test,y_test,ax=dist.ax_)
plot_roc_curve(knn,x_test,y_test,ax=dist.ax_)

plt.legend(prop={'size':11},loc='lower right')
plt.show()
```



The DecisonTree classifier is the best model for the Loan application status model as it has highest accuracy and least difference between the cross validation and accuracy.

The hyper parameter is most important part of modal building as its tune the model for better performance. The Hyper parameter tuning is done on the selected models, here DecisonTree classifier has been selected for the hyper parameter tuning. Passing the multiple parameters to hyper parameter to tune the model and find the best parameters for it to increases the accuracy of the model.

- Using gridsearchcv to calculate best parameter:

```
#Now Appling the gridsearchcv beofore building the model to check the best parameter.
best_param={'criterion':['gini','entropy'],
            'max_depth':[10,12,15,20,25,30],
            'min_samples_split':[2,3,5,7,9,11],
            'min_samples_leaf':[2,4,6,8,10,12]}
```

```
gridcv=GridSearchCV(dt,param_grid=best_param)
```

```
gridcv.fit(x_train,y_train)
```

```
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [10, 12, 15, 20, 25, 30],
                         'min_samples_leaf': [2, 4, 6, 8, 10, 12],
                         'min_samples_split': [2, 3, 5, 7, 9, 11]})
```

The best parameter to tune the default parameter for RandomForest classifier:

```
gridcv.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 12,
 'min_samples_leaf': 6,
 'min_samples_split': 2}
```

```
#Using the DecisionTreeClassifier algorithm to check the accuray score,DecisionTreeClassifier and Confusion Matrix:
dt=DecisionTreeClassifier(criterion='gini',max_depth=12,min_samples_leaf=6,min_samples_split=2)
dt.fit(x_train,y_train)
y_preddt=dt.predict(x_test)

print('\n==========Outputs of DT==========')

print('\n==========Accuracy Score==========')
print(f"Accuracy Score is : {accuracy_score(y_test,y_preddt)* 100:.2f}%\n")

print('======Classification Report=============')
print(classification_report(y_test,y_preddt,digits=2),'\n')

print('=========Confusion Matrix============')
print(confusion_matrix(y_test,y_preddt))
```

```
==========Outputs of DT==========

==========Accuracy Score==========
Accuracy Score is : 82.41%

======Classification Report=============
              precision    recall  f1-score   support

           0       0.76      0.84      0.80        81
           1       0.88      0.81      0.85       118

    accuracy                           0.82       199
   macro avg       0.82      0.83      0.82       199
weighted avg       0.83      0.82      0.83       199


=========Confusion Matrix============
[[68 13]
 [22 96]]
```

The accuracy of Loan application status model is 82.41% after the hyper parameter tuning. We are selecting DecisonTree as our final model as it give highest accuracy.

5. Concluding Remarks:

The predictive models based on Logistic Regression, Decision Tree and Random Forest, give the accuracy as 83.42%,88.44% and 78.39% whereas the cross-validation is found to be 75.26%, 82.05% and 74.05% respectively. This shows that for the given dataset, the accuracy of model based on random forest is highest but decision tree is better at generalization even though its cross validation is not much higher than logistic regression.

Building of Loan application status model consist of multiple parameters like, data analysing, Data cleaning, Visualization, EDA, Model building. There is multiple factors, which is used to help building the model. Lots of learning and research is required to build a model. The final accuracy of the Loan application status model is 82.41%.