```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)

%cd "drive/My Drive/Data Science/Exp 11-Loan Prediction"
```

```
Mounted at /content/drive
/content/drive/My Drive/Data Science/Exp 11-Loan Prediction
```

Ex. No: 11

Aim: Perform following pre-processing techniques on loan prediction dataset

> a. Feature Scaling

> b. Feature Standardization

> c. Label Encoding

> d. One Hot Encoding

Theory:

Pre-processing refers to the transformations applied to your data before feeding it to the algorithm. In python, scikit-learn library has a pre-built functionality under sklearn.preprocessing. There are many more options for pre-processing which we'll explore.

Available Dataset

I have used a subset of the Loan Prediction (missing value observations are dropped) data set

Note : Testing data that you are provided is the subset of the training data from Loan Prediction problem.

Now, lets get started by importing important packages and the data set.

```
# Importing pandas
import pandas as pd
# Importing training data set
X_train=pd.read_csv('Dataset/X_train.csv')
Y_train=pd.read_csv('Dataset/Y_train.csv')
# Importing testing data set
X_test=pd.read_csv('Dataset/X_test.csv')
Y_test=pd.read_csv('Dataset/Y_test.csv')
```

```
print (X_train.head())
```

```
    Loan_ID Gender Married Dependents Education Self_Employed  \
0  LP001032   Male      No          0  Graduate            No
1  LP001824   Male     Yes          1  Graduate            No
2  LP002928   Male     Yes          0  Graduate            No
3  LP001814   Male     Yes          2  Graduate            No
4  LP002244   Male     Yes          0  Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4950                0.0         125               360
1             2882             1843.0         123               480
2             3000             3416.0          56               180
3             9703                0.0         112               360
4             2333             2417.0         136               360

   Credit_History Property_Area
0               1         Urban
1               1     Semiurban
2               1     Semiurban
3               1         Urban
4               1         Urban
```

11 (A):

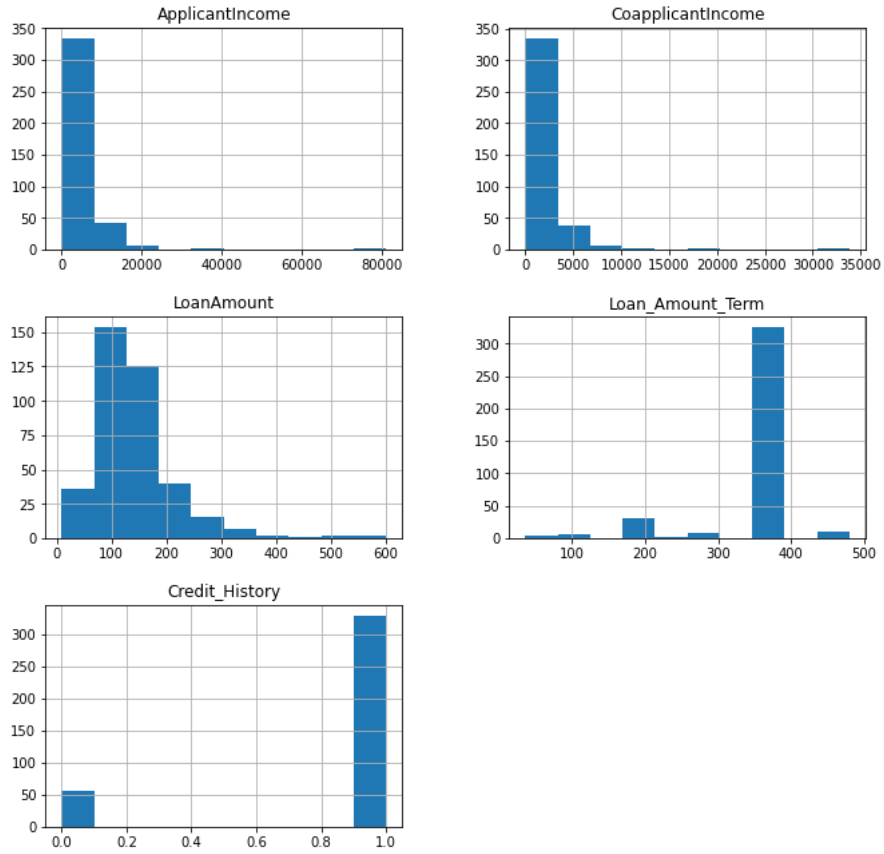
Feature Scaling:

Feature scaling is the method to limit the range of variables so that they can be compared on common grounds. It is performed on continuous variables.

Lets plot the distribution of all the continuous variables in the data set.

```
import matplotlib.pyplot as plt
X_train[X_train.dtypes[(X_train.dtypes=="float64")|(X_train.dtypes=="int64")].index.values].hist(figsize=[11,11])
```

```
array([[<AxesSubplot:title={'center':'ApplicantIncome'}>,
        <AxesSubplot:title={'center':'CoapplicantIncome'}>],
       [<AxesSubplot:title={'center':'LoanAmount'}>,
        <AxesSubplot:title={'center':'Loan_Amount_Term'}>],
       [<AxesSubplot:title={'center':'Credit_History'}>, <AxesSubplot:>]],
      dtype=object)
```



After understanding these plots, we infer that ApplicantIncome and CoapplicantIncome are in similar range (0-50000) where as LoanAmount is in thousands and it ranges from 0 to 600$. The story for Loan_Amount_Term is completely different from other variables because its unit is months as opposed to other variables where the unit is dollars.

If we try to apply distance based methods such as kNN on these features, feature with the largest range will dominate the outcome results and we'll obtain less accurate predictions. We can overcome this trouble using feature scaling.

```
# Importing libraries
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Importing training data set
X_train=pd.read_csv('Dataset/X_train.csv')
Y_train=pd.read_csv('Dataset/Y_train.csv')
print("\nShape of train set:", X_train.shape, Y_train.shape)

# Importing testing data set
X_test=pd.read_csv('Dataset/X_test.csv')
Y_test=pd.read_csv('Dataset/Y_test.csv')
print("\nShape of test set:", X_test.shape, Y_test.shape)

# Initializing and Fitting a k-NN model
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']],Y_train.values.ravel())

# Checking the performance of our model on the testing data set
print("\nAccuracy score on test set :", accuracy_score(Y_test,knn.predict(X_test[['ApplicantIncome','CoapplicantIncome','LoanAmount', 'Lc

# We got around 61% of correct prediction which is not bad but in real world practices will this be enough ? Can we deploy this model in

print("\nDistribution of Loan_Status in train set :")
```

```
print(Y_train.Target.value_counts()/Y_train.Target.count())

# There are 70% of approved loans, since there are more number of approved loans we will generate a prediction where all the loans are ap

print("\nDistribution of Loan_Status in predictions on the test set :")
print(Y_test.Target.value_counts()/Y_test.Target.count())

## NOTE: You can run the rest of the code given below in this blog using this live coding window.
```

```
Shape of train set: (384, 12) (384, 1)

Shape of test set: (96, 12) (96, 1)

Accuracy score on test set : 0.6145833333333334

Distribution of Loan_Status in train set :
Y    0.705729
N    0.294271
Name: Target, dtype: float64

Distribution of Loan_Status in predictions on the test set :
Y    0.635417
N    0.364583
Name: Target, dtype: float64
```

Wow !! we got an accuracy of 63% just by guessing, What is the meaning of this, getting better accuracy than our prediction model ?

This might be happening because of some insignificant variable with larger range will be dominating the objective function. We can remove this problem by scaling down all the features to a same range. sklearn provides a tool MinMaxScaler that will scale down all the features between 0 and 1. Mathematical formula for MinMaxScaler is.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
# Importing MinMaxScaler and initializing it
from sklearn.preprocessing import MinMaxScaler
min_max=MinMaxScaler()
# Scaling down both train and test data set
X_train_minmax=min_max.fit_transform(X_train[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
X_test_minmax=min_max.fit_transform(X_test[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])

# Fitting k-NN on our scaled data set
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_minmax,Y_train)
# Checking the model's accuracy
accuracy_score(Y_test,knn.predict(X_test_minmax))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWarning: A column-vector y was passe
  return self._fit(X, y)
0.75
```

Great !! Our accuracy has increased from 61% to 75%. This means that some of the features with larger range were dominating the prediction outcome in the domain of distance based methods(kNN).

11 (B):

Feature Standardization:

Many machine learning algorithms in sklearn requires standardized data which means having zero mean and unit variance.

Standardization (or Z-score normalization) is the process where the features are rescaled so that they'll have the properties of a standard normal distribution with μ=0 and σ=1, where μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called z scores) of the samples are calculated as follows :

$$z = \frac{x - \mu}{\sigma}$$

Features having larger order of variance would dominate on the objective function as it happened in the previous section with the feature having large range. Without any preprocessing on the data the accuracy will be less, lets standardize our data apply logistic regression on that. Sklearn provides scale to standardize the data.

```
# Standardizing the train and test data
from sklearn.preprocessing import scale
X_train_scale=scale(X_train[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
X_test_scale=scale(X_test[['ApplicantIncome', 'CoapplicantIncome',
               'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])

# Fitting logistic regression on our standardized data set
from sklearn.linear_model import LogisticRegression
log=LogisticRegression(penalty='l2',C=.01)
log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
accuracy_score(Y_test,log.predict(X_test_scale))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
  y = column_or_1d(y, warn=True)
0.75
```

We again reached to our maximum score that was attained using kNN after scaling. This means standardizing the data when using a estimator having l1 or l2 regularization helps us to increase the accuracy of the prediction model. Other learners like kNN with euclidean distance measure, k-means, SVM, perceptron, neural networks, linear discriminant analysis, principal component analysis may perform better with standardized data.

11 (C):

Label Encoding:

In previous experiment, we did the pre-processing for continuous numeric features. But, our data set has other features too such as **Gender, Married, Dependents, Self_Employed** and **Education**. All these categorical features have string values. For example, **Gender** has two levels either **Male** or **Female**. Lets feed the features in our logistic regression model.

```
# Fitting a logistic regression model on whole data
log=LogisticRegression(penalty='l2',C=.01)
log.fit(X_train,Y_train)
# Checking the model's accuracy
accuracy_score(Y_test,log.predict(X_test))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-0b8941f8cace> in <module>
      1 # Fitting a logistic regression model on whole data
      2 log=LogisticRegression(penalty='l2',C=.01)
----> 3 log.fit(X_train,Y_train)
      4 # Checking the model's accuracy
      5 accuracy_score(Y_test,log.predict(X_test))

                                 5 frames
/usr/local/lib/python3.9/dist-packages/pandas/core/generic.py in __array__(self, dtype)
   2062
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'LP001032'
```

SEARCH STACK OVERFLOW

We got an error saying that it cannot convert string to float. So, what's actually happening here is learners like logistic regression, distance based methods such as kNN, support vector machines, tree based methods etc. in sklearn needs numeric arrays. Features having string values cannot be handled by these learners.

Sklearn provides a very efficient tool for encoding the levels of a categorical features into numeric values. LabelEncoder encode labels with value between 0 and n_classes-1.

```
# Importing LabelEncoder and initializing it
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
# Iterating over all the common columns in train and test
for col in X_test.columns.values:
  # Encoding only categorical variables
  if X_test[col].dtypes=='object':
    # Using whole data to form an exhaustive list of levels
    data=X_train[col].append(X_test[col])
    le.fit(data.values)
    X_train[col]=le.transform(X_train[col])
    X_test[col]=le.transform(X_test[col])
```

All our categorical features are encoded. You can look at your updated data set using X_train.head(). We are going to take a look at Gender frequency distribution before and after the encoding.

```
print (X_train.head())
```

```
   Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
0       13       1        0           0          0              0
1      193       1        1           1          0              0
2      461       1        1           0          0              0
3      191       1        1           2          0              0
4      300       1        1           0          0              0

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4950                0.0         125               360
1             2882             1843.0         123               480
2             3000             3416.0          56               180
3             9703                0.0         112               360
4             2333             2417.0         136               360

   Credit_History  Property_Area
0               1              2
1               1              1
2               1              1
3               1              2
4               1              2
```

Now that we are done with label encoding, lets now run a logistic regression model on the data set with both categorical and continuous features.

```
# Standardizing the features
X_train_scale=scale(X_train)
X_test_scale=scale(X_test)
# Fitting the logistic regression model
log=LogisticRegression(penalty='l2',C=.01)
log.fit(X_train_scale,Y_train)
# Checking the models accuracy
accuracy_score(Y_test,log.predict(X_test_scale))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
  y = column_or_1d(y, warn=True)
0.7395833333333334
```

11 (D):

One Hot Encoding:

One-Hot Encoding transforms each categorical feature with n possible values into n binary features, with only one active.

Most of the ML algorithms either learn a single weight for each feature or it computes distance between the samples. Algorithms like linear models (such as logistic regression) belongs to the first category.

Lets take a look at an example from loan_prediction data set. Feature Dependents have 4 possible values 0,1,2 and 3+ which are then encoded without loss of generality to 0,1,2 and 3.

We, then have a weight “W” assigned for this feature in a linear classifier,which will make a decision based on the constraints **W*Dependents + K > 0** or eqivalently **W*Dependents < K**.

Let **f(w)= W*Dependents**

Possible values that can be attained by the equation are 0, W, 2W and 3W. A problem with this equation is that the weight “W” cannot make decision based on four choices. It can reach to a decision in following ways:

All leads to the same decision (all of them <K or vice versa) 3:1 division of the levels (Decision boundary at f(w)>2W) 2:2 division of the levels (Decision boundary at f(w)>W) Here we can see that we are loosing many different possible decisions such as the case where “0” and “2W” should be given same label and “3W” and “W” are odd one out.

This problem can be solved by One-Hot-Encoding as it effectively changes the dimensionality of the feature “Dependents” from one to four, thus every value in the feature “Dependents” will have their own weights. Updated equation for the decison would be f'(w) < K.

where, **f'(w) = W1*D_0 + W2*D_1 + W3*D_2 + W4*D_3**

All four new variable has boolean values (0 or 1).

The same thing happens with distance based methods such as kNN. Without encoding, distance between “0” and “1” values of Dependents is 1 whereas distance between “0” and “3+” will be 3, which is not desirable as both the distances should be similar. After encoding, the values will be new features (sequence of columns is 0,1,2,3+) : [1,0,0,0] and [0,0,0,1] (initially we were finding distance between “0” and “3+”), now the distance would be √2.

For tree based methods, same situation (more than two values in a feature) might effect the outcome to extent but if methods like random forests are deep enough, it can handle the categorical variables without one-hot encoding.

Now, lets take look at the implementation of one-hot encoding with various algorithms.

Lets create a logistic regression model for classification without one-hot encoding.

```
# We are using scaled variable as we saw in previous section that
# scaling will effect the algo with l1 or l2 reguralizer
X_train_scale=scale(X_train)
X_test_scale=scale(X_test)
# Fitting a logistic regression model
log=LogisticRegression(penalty='l2',C=1)
log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
accuracy_score(Y_test,log.predict(X_test_scale))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
  y = column_or_1d(y, warn=True)
0.7395833333333334
```

Now we are going to encode the data.

```
from sklearn.preprocessing import OneHotEncoder
enc=OneHotEncoder(sparse=False)
X_train_1=X_train
X_test_1=X_test
columns=['Gender', 'Married', 'Dependents', 'Education','Self_Employed',
         'Credit_History', 'Property_Area']
for col in columns:
  # creating an exhaustive list of all possible categorical values
  data=X_train[[col]].append(X_test[[col]])
  enc.fit(data)
  # Fitting One Hot Encoding on train data
  temp = enc.transform(X_train[[col]])
  # Changing the encoded features into a data frame with new column names
  temp=pd.DataFrame(temp,columns=[(col+"_"+str(i)) for i in data[col].value_counts().index])
  # In side by side concatenation index values should be same
  # Setting the index values similar to the X_train data frame
  temp=temp.set_index(X_train.index.values)
  # adding the new One Hot Encoded varibales to the train data frame
  X_train_1=pd.concat([X_train_1,temp],axis=1)
  # fitting One Hot Encoding on test data
  temp = enc.transform(X_test[[col]])
  # changing it into data frame and adding column names
  temp=pd.DataFrame(temp,columns=[(col+"_"+str(i)) for i in data[col].value_counts().index])
  # Setting the index for proper concatenation
  temp=temp.set_index(X_test.index.values)
  # adding the new One Hot Encoded varibales to test data frame
  X_test_1=pd.concat([X_test_1,temp],axis=1)

#Now, lets apply logistic regression model on one-hot encoded data.

# Standardizing the data set
X_train_scale=scale(X_train_1)
X_test_scale=scale(X_test_1)
# Fitting a logistic regression model
log=LogisticRegression(penalty='l2',C=1)
log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
accuracy_score(Y_test,log.predict(X_test_scale))
```

```
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
<ipython-input-20-fc78ffeaa477>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  data=X_train[[col]].append(X_test[[col]])
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
  y = column_or_1d(y, warn=True)
0.75
```

Here, again we got the maximum accuracy as 0.75 that we have gotten so far. In this case, logistic regression regularization(C) parameter 1 where as earlier we used C=0.01.

✓ 0s completed at 14:12