Table of Contents

LAB2 tutorial for Machine Learning course

Objectives

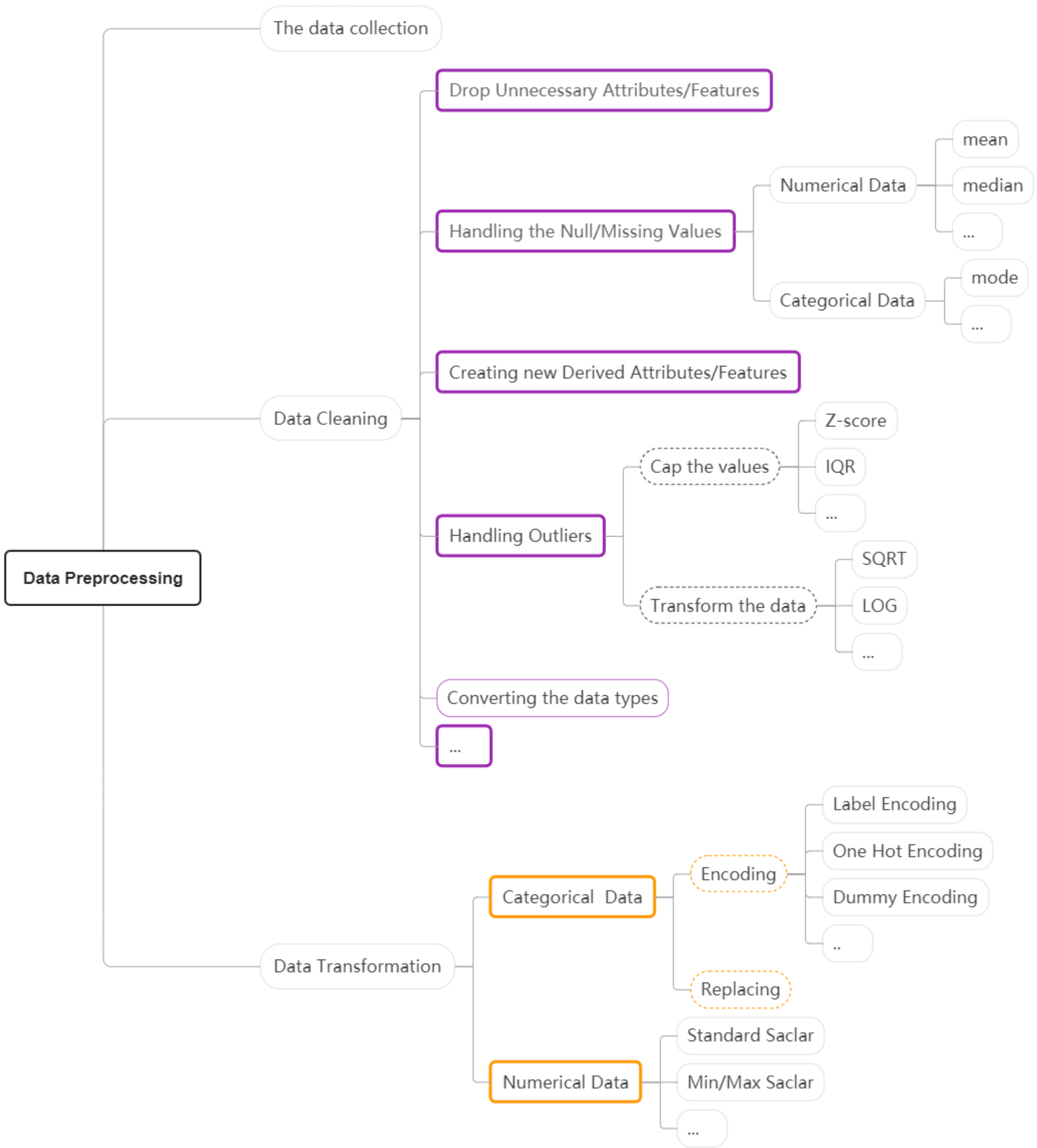
- Learning how to preprocess data
- Learning how to evaluate classification models
- Complete the LAB assignment and submit it (we have provided you with a template for this assignment).

1 Data Preprocessing

In the real world, the data that we work with is raw, it is not clean and needs processing to be ready to be passed to a machine learning model.

You may have heard that 80% of a data scientist's time goes into data preprocessing and 20% of the time for model building.

The data preprocessing techniques in machine learning can be broadly segmented into two parts: Data Cleaning and Data Transformation. The following flow-chart illustrates the above data preprocessing techniques and steps in machine learning:



1.1 Getting\ collecting the dataset

A machine learning model completely works on data. So, to build a machine learning model, the first thing we need is a dataset. The **dataset** is a proper format of collected data for a particular problem.

There are several online sources from where you can download datasets like https://www.kaggle.com/datasets and https://archive.ics.uci.edu/ml/index.php.

You can also create a dataset by collecting data via different Python APIs.

Once the dataset is ready, you must put it in `CSV`, or `HTML`, or `XLSX` file formats.

Next, we will learn the above data preprocessing steps using the `census.csv` dataset.

1.1.1 Importing Libraries

In [ ]:
```
# 'os' module provides functions for interacting with the operating system
import os
```

```
# 'Numpy' is used for mathematical operations on large, multi-dimensional arrays ar
import numpy as np

# 'Pandas' is used for data manipulation and analysis
import pandas as pd

# 'Matplotlib' is a data visualization library for 2D and 3D plots, built on numpy
from matplotlib import pyplot as plt
%matplotlib inline

# 'Seaborn' is based on matplotlib; used for plotting statistical graphics
import seaborn as sns

# to suppress warnings
import warnings
warnings.filterwarnings("ignore")
```

1.1.2 Importing and Exploration of the dataset

In [ ]:
```
# Loading the data and setting the unique client_id as the index::

df = pd.read_csv('load_loans.csv')
# # showing the first 5 rows of the dataset:
df.head(n=10)
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	
7	LP001014	Male	Yes	3+	Graduate	No	3036	
8	LP001018	Male	Yes	2	Graduate	No	4006	
9	LP001020	Male	Yes	1	Graduate	No	12841	

In [ ]:
```
# To check the Dimensions of the dataset:
df.shape
```

Out[ ]: (614, 13)

In [ ]:
```
# Checking the info of the data:
df.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    object 
 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(4), int64(1), object(8)
memory usage: 62.5+ KB
```

Checking the datatypes of the columns:

In [ ]:
```
df.dtypes
```

Out[ ]:
```
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
```

It consists of dataset attributes for a loan with the below-mentioned description.

Sr. No	Column Name	Column Description
1	Loan_ID	Unique Loan Id
2	Gender	Male/Female
3	Married	Applicant married (Y/N)
4	Dependents	Number of dependents
5	Education	Applicant Education (Graduate/ Undergraduate)
6	Loan_Amount_Term	Term of loan in months
7	Self_Employed	Self-employed (Y/N)
8	Applicant_Income	Income of applicant
9	Coapplicant_Income	Income of co-applicant
10	Loan_Amount	Loan amount in thousands
11	Credit_History	Credit history meeting the guidelines
12	Property_Area	Urban/Semi-urban and Rural
13	Loan_Status	Loan approved (Y/N)

The different variables present in the dataset are:

Numerical features: Applicant_Income, Coapplicant_Income, Loan_Amount, Loan_Amount_Term and Dependents.

Categorical features:Gender,Credit_History>, Self_Employed, Married and Loan_Status.

Alphanumeric Features: Loan_Id.

Text Features: Education and Property_Area.

As mentioned above we need to predict our target variable which is "Loan_Status", "Loan_Status" can have two values.

```
- Y (Yes): If the loan is approved.

- N (No): If the loan is not approved.
```

So using the training dataset we'll train our model and predict our target column "Loan_Status".

Summary Statistics of the data:

In [ ]:
```
# Summary Statistics for Numerical data:
df.describe()
```

Out[ ]:

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

In [ ]:
```
# Summary Statistics for Categorical data:
df.describe(exclude=[np.number])
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan
count	614	601	611	599	614	582	614	
unique	614	2	2	4	2	2	3	
top	LP001002	Male	Yes	0	Graduate	No	Semiurban	
freq	1	489	398	345	480	500	233	

Individual variable analysis

- Analyze the target variable status of Loan_Status

In [ ]:
```
#Target variable statistics
df['Loan_Status'].value_counts()
```

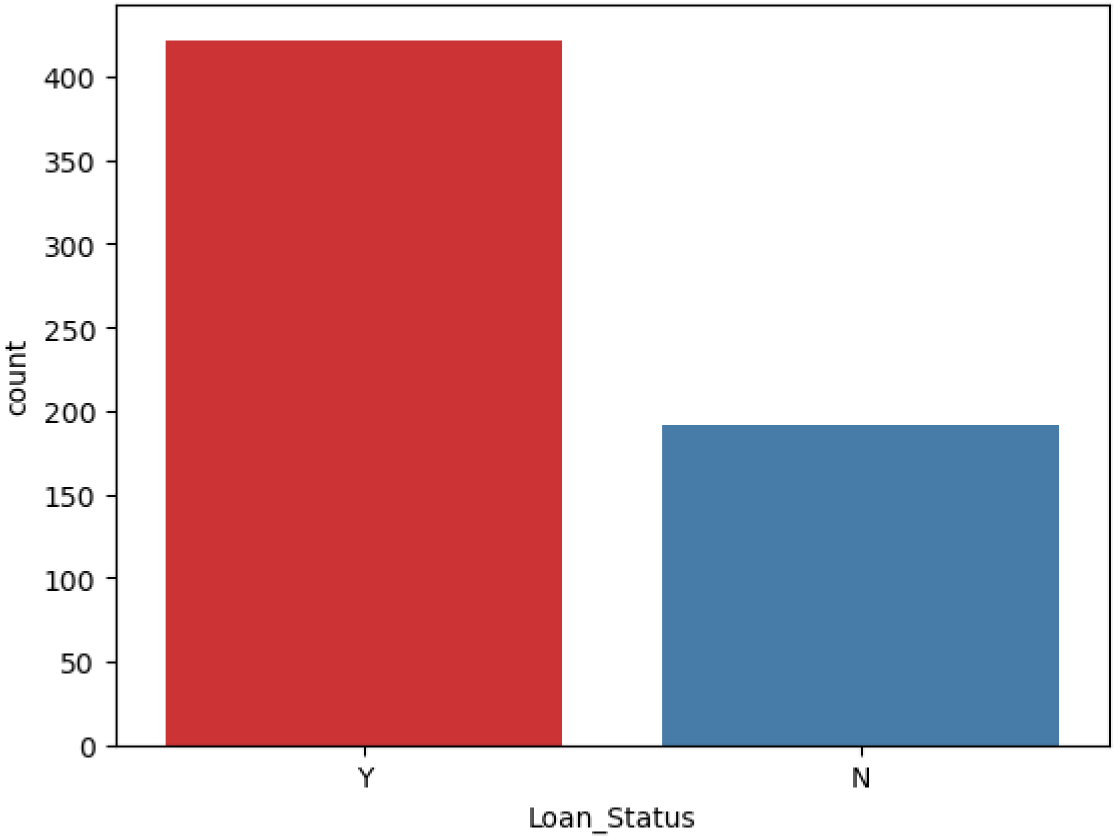
Out[ ]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [ ]:
```
# Percentage statistics
df['Loan_Status'].value_counts(normalize=True)
```

Out[ ]:
```
Y    0.687296
N    0.312704
Name: Loan_Status, dtype: float64
```

In [ ]:
```
sns.countplot(x='Loan_Status', data=df, palette = 'Set1')
```

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



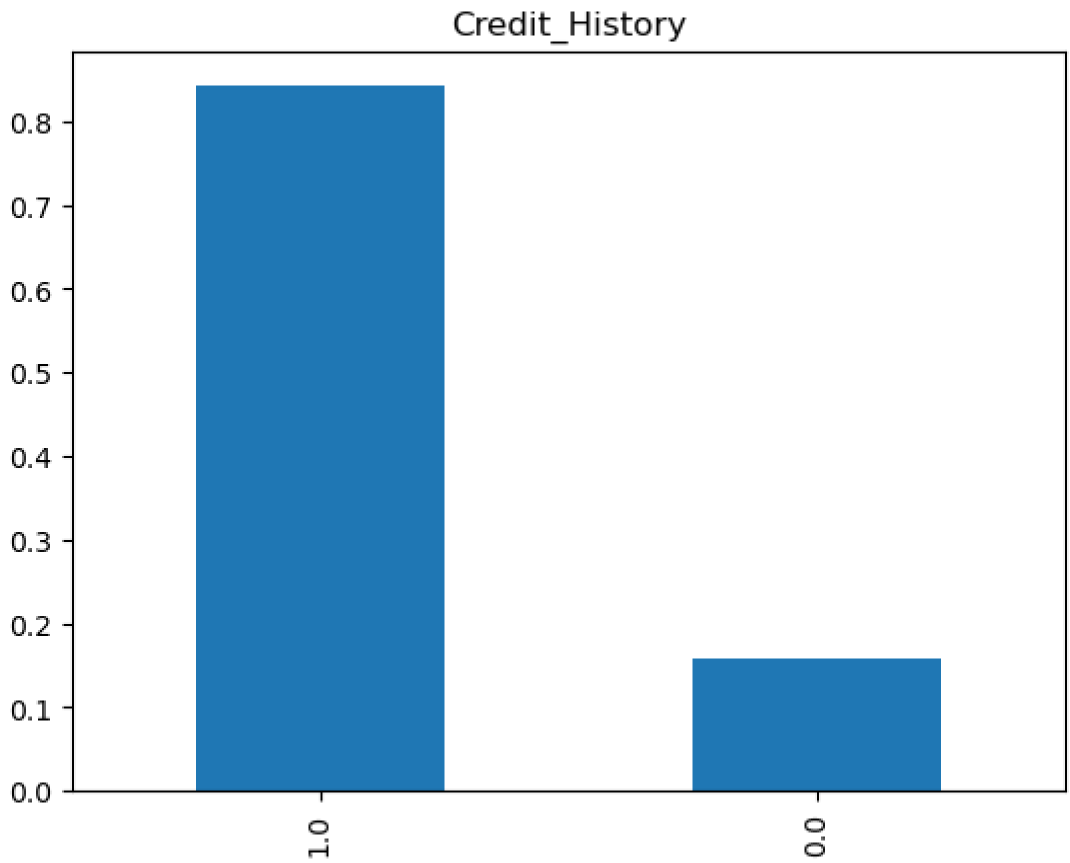
422 of the 614 people received loan approval, about 69% of the loan opportunities

- Analyze Credit_History

In [ ]:
```
Credit_History=df['Credit_History'].value_counts(normalize=True)
print(Credit_History)
Credit_History.plot.bar(title= 'Credit_History')
```

```
1.0    0.842199
0.0    0.157801
Name: Credit_History, dtype: float64
```

Out[ ]: <AxesSubplot:title={'center':'Credit_History'}>



About 84 percent of applicants had a loan history.

Analyze the relationship between each variable and the target variable(loan_status)

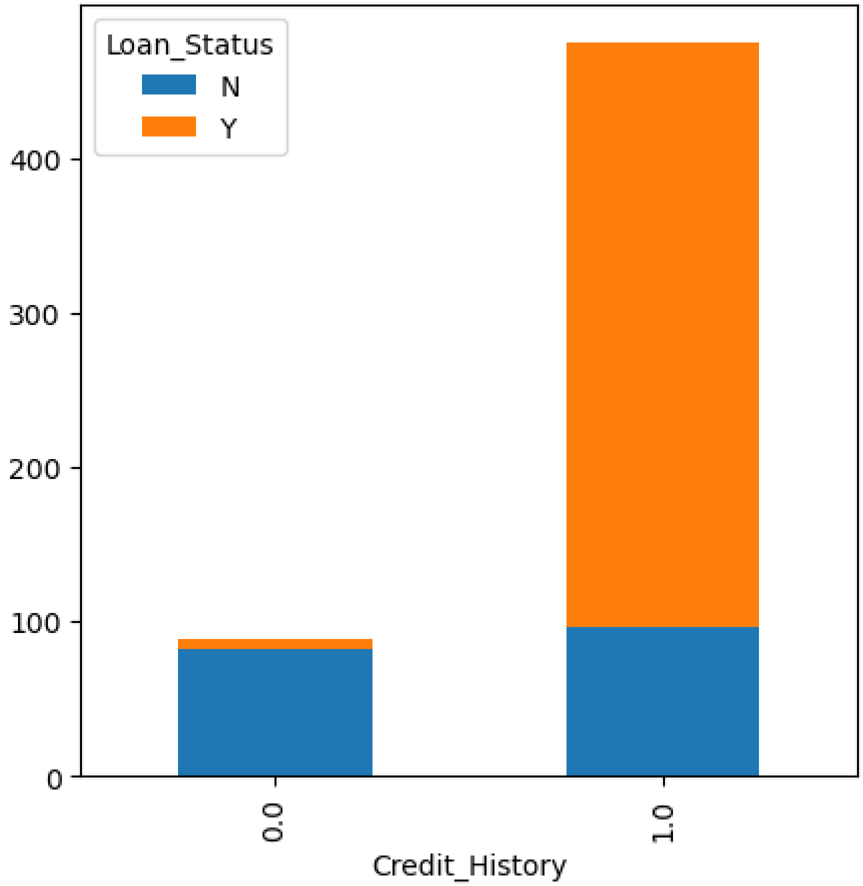
- The relationship between Credit_History and loan_Status

In [ ]:

```
Credit_History=pd.crosstab(df['Credit_History'],df['Loan_Status'])
Credit_History.plot(kind="bar", stacked=True, figsize=(5,5))
```

Out[ ]:

```
<AxesSubplot:xlabel='Credit_History'>
```



Applicants with historical loans are more likely to get loan approval.

1.2 Data Cleaning

1.2.1 Convert the data types

Convert the data types if any mismatch present in the data types of the variables

In [ ]:
```
# CoapplicantIncome:
df['ApplicantIncome'] = df['ApplicantIncome'].astype('float64')
# Checking the datatypes again:
df.dtypes
```

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

1.2.2 Drop unnecessary featuress

In [ ]:
```
#The Loan_ID variable has no effect on the loan status, delete it
```

```
df = df.drop('Loan_ID',axis=1)
df.dtypes
```

Out[ ]:
```
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

1.2.3 Handling the Null/Missing Values

In [ ]:
```
# use isnull().sum() to check for missing values
df.isnull().sum()
```

Out[ ]:
```
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
```

Missing values exist for **Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History**.

The null values in the dataset are imputed using mean/median or mode based on the type of data that is missing:

- **Numerical Data:** If a numerical value is missing, then replace that NaN value with mean or median.
- **Categorical Data:** When categorical data is missing, replace that with the value which is most occurring i.e. by mode.

In [ ]:
```
# fill the missing values for categoriacal terms - mode
df['Gender'].fillna(df['Gender'].value_counts().idxmax(), inplace=True)
df['Married'].fillna(df['Married'].value_counts().idxmax(), inplace=True)
df['Dependents'].fillna(df['Dependents'].value_counts().idxmax(), inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].value_counts().idxmax(), inplace=Tru
# fill the missing values for numerical term s -mean or median
df["LoanAmount"].fillna(df["LoanAmount"].mean(skipna=True), inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].value_counts().idxmax(), inpla
df['Credit_History'].fillna(df['Credit_History'].value_counts().idxmax(), inplace=T
```

```
#Checking missing value again:
df.isnull().sum()
```

Out[ ]:

```
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
```

From the above output, we see that there're no null values and now we can perform the data visualization.

NOTICE: If a column has, let's say, 50% of its values missing, then do we replace all of those missing values with the respective median or mode value? Actually, we don't. We delete that particular column in that case.

Sk-learn library has an in-built function called Iterative Imputer to impute the missing values. Its sklearn domcumentation: https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html

1.2.4 Creating new Derived Features

Create a new attribute named "TotalIncome" which is the sum of "CoapplicantIncome" and "ApplicantIncome" ,Here we assume that "Coapplicant" is the person from the same family

In [ ]:

```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

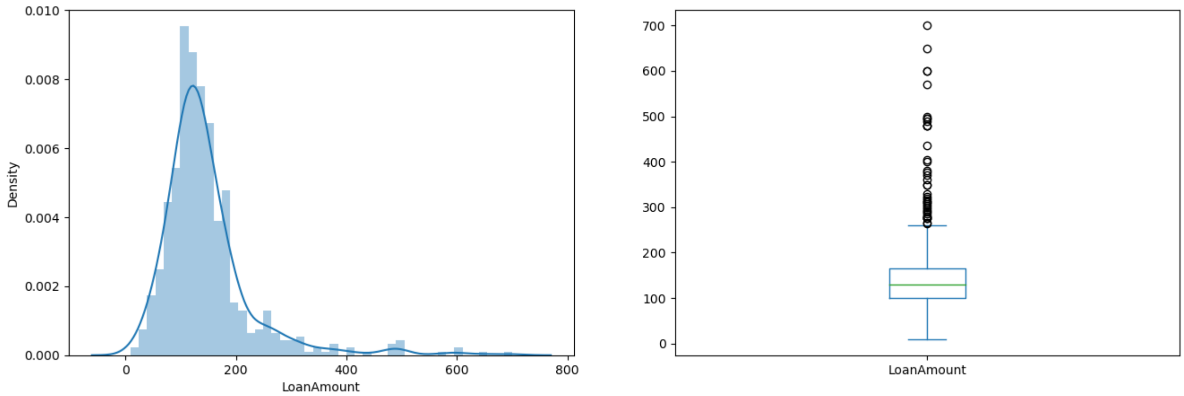
1.2.5 Outliers Treatment

To check for the presence of outliers, we plot distribution and Boxplot.

In [ ]:

```
plt.figure(1)
plt.subplot(121)
sns.distplot(df['LoanAmount']);
```

```
plt.subplot(122)
df['LoanAmount'].plot.box(figsize=(16,5))
plt.show()
```



The loanAmount shows a normal distribution, but from the boxplot, we can see that there are many outliers, which need to be dealt with.

To treat for outliers can either `cap the values` or `transform the data`. Shall demonstrate both the approaches here.

1.2.5.1 Transformation

a. SQRT transformation

In [ ]:
```
df['Sqrt_LoanAmount'] = np.sqrt(df['LoanAmount'])
df.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

Checking the skewness, kurtosis:

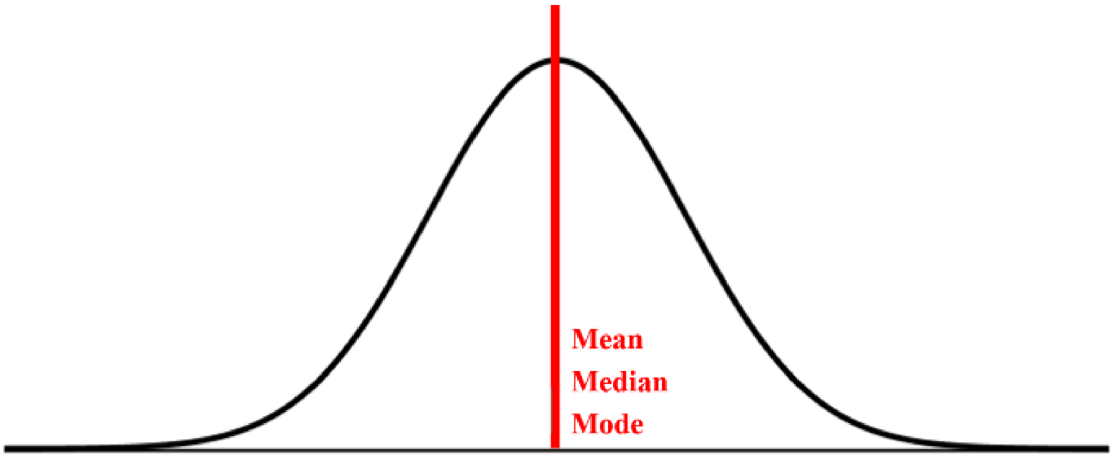
In [ ]:
```
#checking the skewness, kurtosis between the original and transformed data:
print("The skewness of the original data is {}".format(df.LoanAmount.skew()))
print('The skewness of the SQRT transformed data is {}'.format(df.Sqrt_LoanAmount.s

print('')

print("The kurtosis of the original data is {}".format(df.LoanAmount.kurt()))
print("The kurtosis of the SQRT transformed data is {}".format(df.Sqrt_LoanAmount.k
```

```
The skewness of the original data is 2.726601144105299
The skewness of the SQRT transformed data is 1.3141619498030808

The kurtosis of the original data is 10.896456468091559
The kurtosis of the SQRT transformed data is 3.959374942476666
```

Here we explain to you what is skewness and kurtosis.

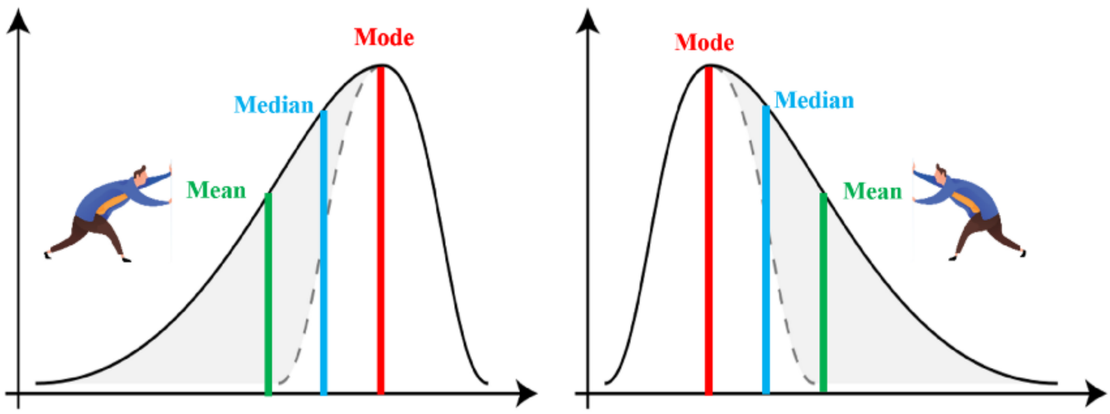
In statistics, `Skewness` is a measure of the asymmetry of a distribution.

The normal distribution helps to know a skewness. When we talk about normal distribution, data symmetrically distributed. The symmetrical distribution has zero skewness as all measures of a central tendency lies in the middle.



Normal distribution

But what if we encounter an asymmetrical distribution, how do we detect the extent of asymmetry? Using skewness!! This value can be positive or negative.

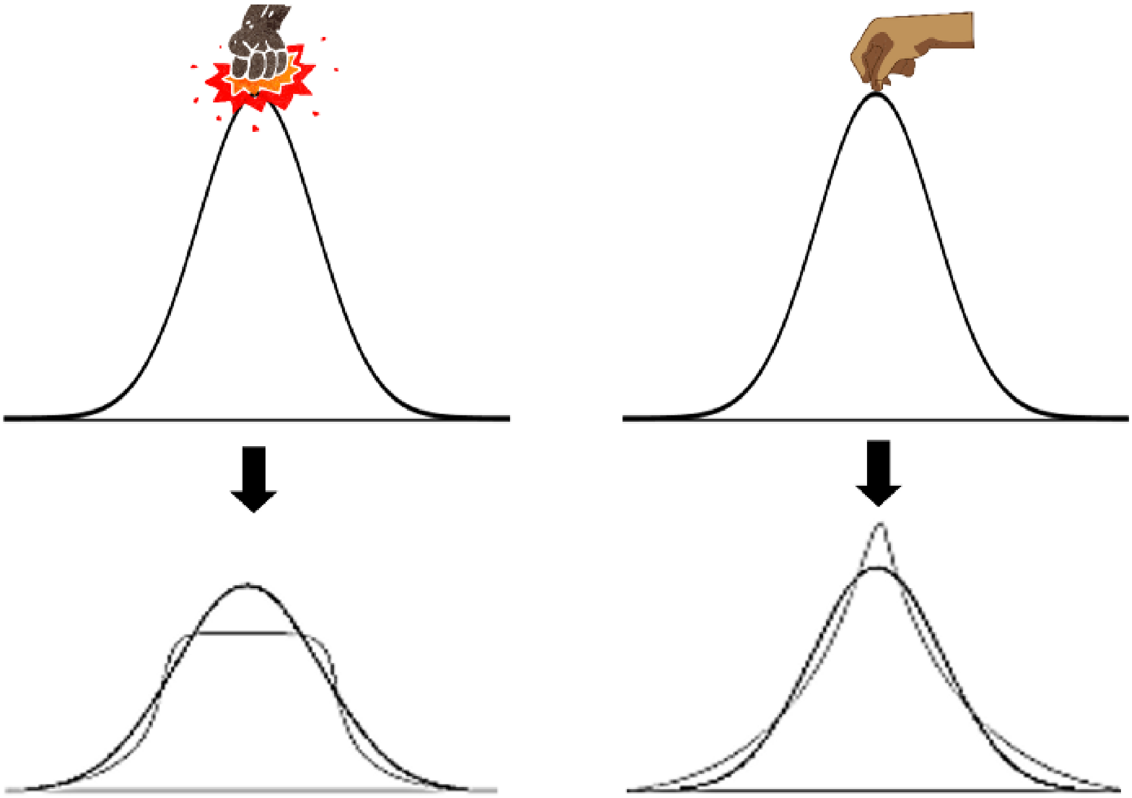


Negative skewed or left-skewed

Positive skewed or right-skewed

Calculate the skewness:

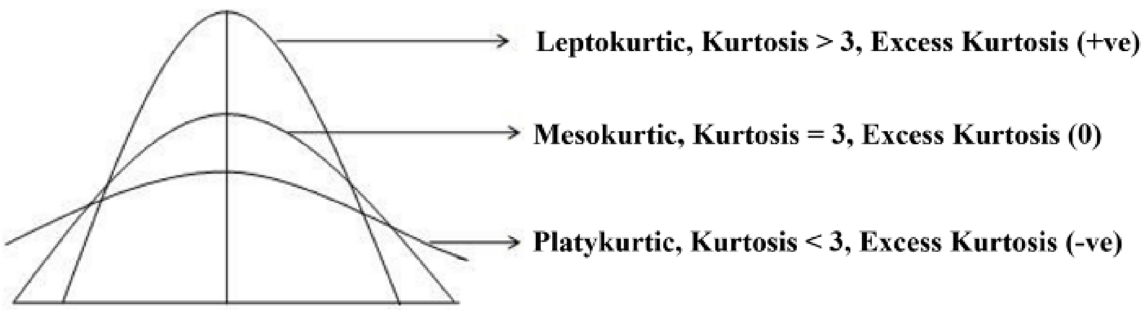
`Kurtosis` is a measure of whether or not a distribution is heavy-tailed or light-tailed relative to a normal distribution. Think of punching or pulling the normal distribution curve from the top, what impact will it have on the shape of the distribution? Let’s visualize:



Punching or Pulling the normal distribution.

So there are two things to notice — The peak of the curve and the tails of the curve, Kurtosis measure is responsible for capturing this phenomenon. Kurtosis ranges from 1 to infinity.

The kurtosis of a normal distribution is 3. Distributions greater than 3 are called leptokurtic(细峰) and less than 3 are called platykurtic（浅峰）.



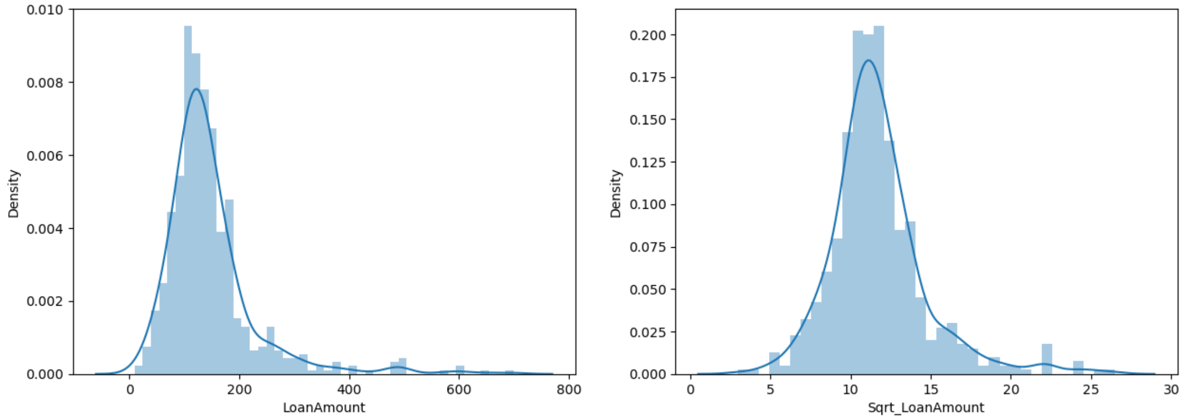
> The topic of Kurtosis has been controversial(争议的) for decades now, the basis of kurtosis all these years has been linked with the peakedness but the ultimate verdict（判决） is that outliers (fatter tails) govern the kurtosis effect far more than the values near the mean (peak).

Note: Some formulas (Fisher's definition) subtract 3 from the kurtosis to make it easier to compare with the normal distribution.

$$\textbf{Excess Kurtosis} = \textbf{Kurtosis} - 3$$

So we can conclude from the above discussions that the horizontal push or pull distortion of a normal distribution curve gets captured by the Skewness measure and the vertical push or pull distortion gets captured by the Kurtosis measure. Also, it is the impact of outliers that dominate the kurtosis effect.

In [ ]:
```
# plotting the distribution

fig, axes = plt.subplots(1,2, figsize=(15,5))
sns.distplot(df['LoanAmount'], ax=axes[0])
sns.distplot(df['Sqrt_LoanAmount'], ax=axes[1])

plt.show()
```



Result:

The LoanAmount column was right skewed earlier. The skewness and kurtosis as reduced significantly. The transformed SQRT rate, on the right graph resembles normal distribution now.

b. Log Transformation

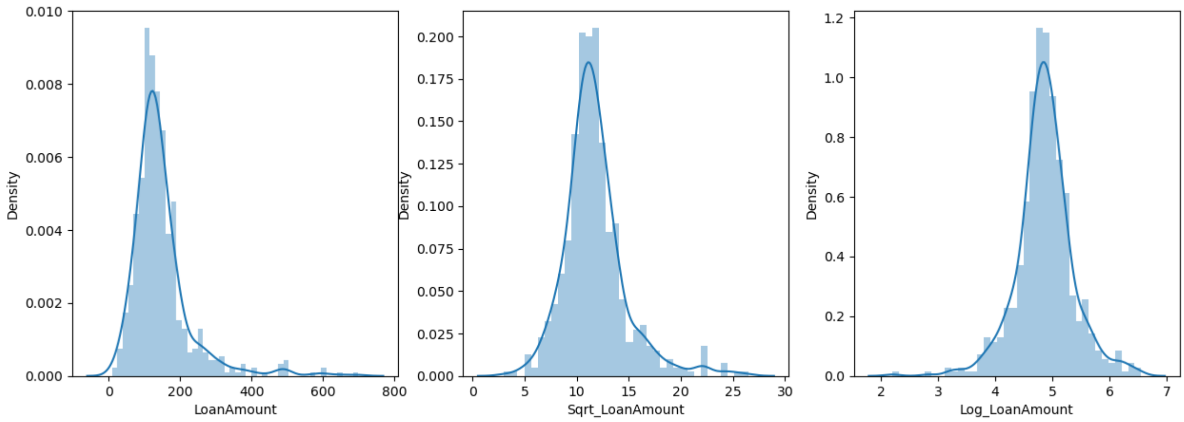
In [ ]:
```
df['Log_LoanAmount'] = np.log(df['LoanAmount'])
```

In [ ]:
```
print("The skewness of the original data is {}".format(df.LoanAmount.skew()))
print('The skewness of the SQRT transformed data is {}'.format(df.Sqrt_LoanAmount.s
print("The skewnss of the LOG transformed data is {}".format(df['Log_LoanAmount'].s

print('')


print("The kurtosis of the original data is {}".format(df.LoanAmount.kurt()))
print("The kurtosis of the SQRT transformed data is {}".format(df['Sqrt_LoanAmount'
print("The kurtosis of the LOG transformed data is {}".format(df['Log_LoanAmount'].
```

```
The skewness of the original data is 2.726601144105299
The skewness of the SQRT transformed data is 1.3141619498030808
The skewnss of the LOG transformed data is -0.22322704759640444

The kurtosis of the original data is 10.896456468091559
The kurtosis of the SQRT transformed data is 3.959374942476666
The kurtosis of the LOG transformed data is 2.7999727252250457
```

In [ ]:
```
# plot the graph:
```

```
fig, axes = plt.subplots(1,3,figsize=(15,5))

sns.distplot(df['LoanAmount'], ax=axes[0])
sns.distplot(df['Sqrt_LoanAmount'], ax=axes[1])
sns.distplot(df['Log_LoanAmount'], ax=axes[2])

plt.show()
```



Inference:

Log transformation is more closer to 0 and hence is more normal. Though it heavily maniupulates the data. In our case, Log transformation is more suitable.

In [ ]:
```
df1 = df.copy()
df.drop(columns = ['Log_LoanAmount' ,'Sqrt_LoanAmount'], inplace=True)

df1.drop(columns = ['Sqrt_LoanAmount','LoanAmount'], inplace=True)
df1.dtypes
```

Out[ ]:
```
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome      float64
CoapplicantIncome    float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
TotalIncome          float64
Log_LoanAmount       float64
dtype: object
```

There are other transformations available also called BoxCox. There is an inbuilt function in Sci-kit Learn library called PowerTransformer for this which can also be called to transform the data. We'll see how it works below. Its sklearn domcumentation: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.power_transform.html

---

1.2.5.2 Using Capping Approach

1) Z-Score approach to treat Outliers:

All the values above 3 standard deviation and below -3 standard deviation are outliers and can be removed.

Using SciPy Library to calculate the Z-Score:

In [ ]:
```
# 'SciPy' is used to perform scientific computations
import scipy.stats as stats
# Creating new variable with Z-score of each record:
df2 = df.copy()
df2['ZR'] = stats.zscore(df2['LoanAmount'])
df2.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

In [ ]:
```
# Combined Lower limit and Upper limit:
df2[(df2['ZR']<-3) | (df2['ZR']>3)]
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncor
130	Male	No	0	Graduate	Yes	20166.0	(
155	Male	Yes	3+	Graduate	No	39999.0	(
171	Male	Yes	3+	Graduate	No	51763.0	(
177	Male	Yes	3+	Graduate	No	5516.0	1130(
278	Male	Yes	0	Graduate	No	14583.0	(
308	Male	No	0	Graduate	No	20233.0	(
333	Male	Yes	0	Graduate	No	63337.0	(
369	Male	Yes	0	Graduate	No	19730.0	526(
432	Male	No	0	Graduate	No	12876.0	(
487	Male	Yes	1	Graduate	No	18333.0	(
506	Male	Yes	0	Graduate	No	20833.0	666
523	Male	Yes	2	Graduate	Yes	7948.0	716(
525	Male	Yes	2	Graduate	Yes	17500.0	(
561	Female	Yes	1	Graduate	Yes	19484.0	(
604	Female	Yes	1	Graduate	No	12000.0	(

In [ ]:
```
df2[(df2['ZR']<-3) | (df2['ZR']>3)].shape[0]
```

Out[ ]: 15

In [ ]:
```
### Cleaned Data: without outliers so z>-3 and z< +3

df2= df2[(df2['ZR']>-3) & (df2['ZR']<3)].reset_index()
df2.head()
```

Out[ ]:

	index	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	0	Male	No	0	Graduate	No	5849.0	
1	1	Male	Yes	1	Graduate	No	4583.0	
2	2	Male	Yes	0	Graduate	Yes	3000.0	
3	3	Male	Yes	0	Not Graduate	No	2583.0	
4	4	Male	No	0	Graduate	No	6000.0	

In [ ]:
```
# A crude way to know whether the outliers have been removed or not is to check the
df2.shape,df.shape
```

Out[ ]: ((599, 15), (614, 13))

Interpretation:

From the above output, we can see that the dimensions are reduced that implies outliers are removed.

In [ ]:
```
df3 = df.copy()
df3.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

2) IQR Method to treat Outliers:

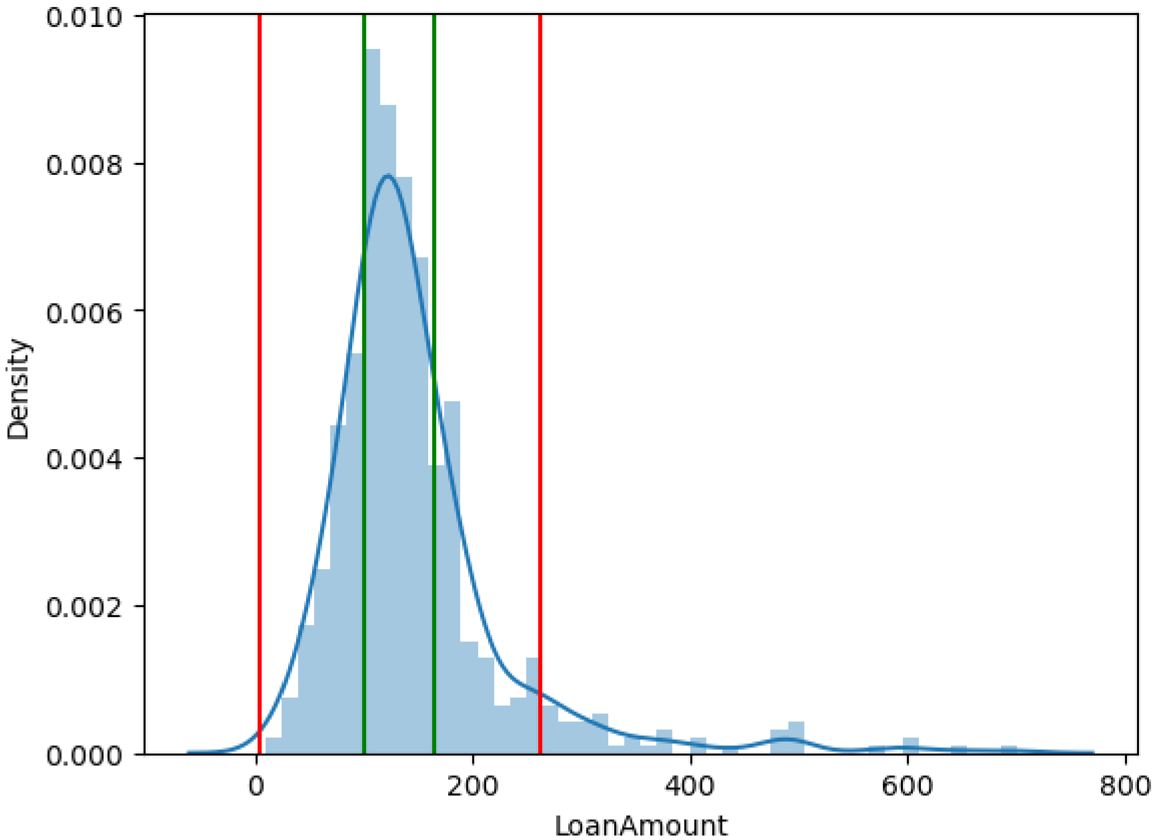
All the values below Q1 - 1.5*IQR and values above Q3 + 1.5*IQR are outliers and can be removed.

In [ ]:
```
# finding the Quantiles:

Q1 = df3.LoanAmount.quantile(0.25)
Q2 = df3.LoanAmount.quantile(0.50)
Q3 = df3.LoanAmount.quantile(0.75)
```

```
# IQR : Inter-Quartile Range

IQR = Q3 - Q1

# Lower Limit:
LC = Q1 - (1.5*IQR)

# Upper Limit:
UC = Q3 + (1.5*IQR)

display(LC)
display(UC)
```

```
3.5
261.5
```

In [ ]:
```
## Plot

sns.distplot(df3.LoanAmount)
plt.axvline(UC, color='r')
plt.axvline(LC, color ='r')
plt.axvline(Q1, color='g')
plt.axvline(Q3, color='g')
plt.show()
```



In [ ]:
```
# Find count of Outliers wrt IQR

df3[(df3.LoanAmount<LC) | (df3.LoanAmount>UC)].reset_index(drop=True)
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncom
0	Male	Yes	2	Graduate	Yes	5417.0	4196.
1	Male	Yes	1	Graduate	No	12841.0	10968.
2	Male	Yes	1	Graduate	No	5955.0	5625.
3	Male	No	3+	Graduate	No	12500.0	3000.
4	Female	Yes	1	Graduate	Yes	11500.0	0.
5	Male	Yes	1	Graduate	No	10750.0	0.
6	Male	Yes	0	Graduate	No	6000.0	2250.
7	Male	Yes	3+	Graduate	No	23803.0	0.
8	Male	No	0	Graduate	Yes	20166.0	0.
9	Male	Yes	3+	Graduate	No	4000.0	7750.
10	Male	Yes	3+	Graduate	No	39999.0	0.
11	Male	Yes	0	Graduate	No	7933.0	0.
12	Male	Yes	3+	Graduate	No	51763.0	0.
13	Male	Yes	3+	Graduate	No	5516.0	11300.
14	Female	No	0	Graduate	No	8333.0	0.
15	Male	Yes	1	Not Graduate	No	2661.0	7101.
16	Male	Yes	0	Graduate	No	14683.0	2100.
17	Male	Yes	1	Graduate	No	6083.0	4250.
18	Male	Yes	0	Graduate	No	14583.0	0.
19	Male	No	0	Graduate	No	20233.0	0.
20	Male	Yes	3+	Graduate	No	15000.0	0.
21	Male	Yes	1	Graduate	Yes	8666.0	4983.
22	Male	Yes	0	Graduate	No	63337.0	0.
23	Male	No	0	Graduate	No	8750.0	4167.
24	Male	Yes	0	Graduate	No	19730.0	5266.
25	Male	Yes	2	Graduate	Yes	9323.0	7873.
26	Male	No	0	Graduate	No	5941.0	4232.
27	Male	Yes	3+	Graduate	No	9504.0	0.
28	Male	Yes	3+	Graduate	No	81000.0	0.
29	Male	No	0	Graduate	No	12876.0	0.
30	Male	Yes	1	Graduate	No	18333.0	0.
31	Male	Yes	0	Graduate	No	20833.0	6667.
32	Male	No	0	Graduate	No	5815.0	3666.
33	Male	Yes	2	Graduate	Yes	7948.0	7166.
34	Male	Yes	2	Graduate	Yes	17500.0	0.

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncom
35	Male	Yes	0	Graduate	No	6133.0	3906.
36	Female	Yes	1	Graduate	Yes	19484.0	0.
37	Male	Yes	2	Graduate	No	16666.0	0.
38	Male	No	3+	Graduate	Yes	9357.0	0.
39	Female	No	3+	Graduate	No	416.0	41667.
40	Female	Yes	1	Graduate	No	12000.0	0.

In [ ]:
```
df3[(df3.LoanAmount<LC) | (df3.LoanAmount>UC)].shape[0]
```

Out[ ]: 41

In [ ]:
```
## Store the clean data wrt IQR:

df3 = df3[(df3.LoanAmount>LC) & (df3.LoanAmount<UC)]
df3.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

In [ ]:
```
df3.shape,df.shape
```

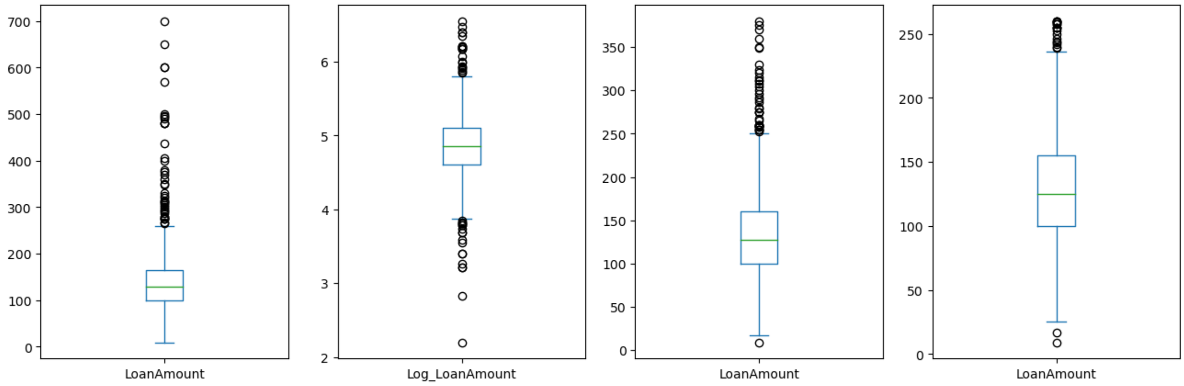
Out[ ]: ((573, 13), (614, 13))

Interpretation:

A crude way to know whether the outliers have been removed or not is to check the dimensions of the data. From the above output, we can see that the dimensions are reduced that implies outliers are removed.

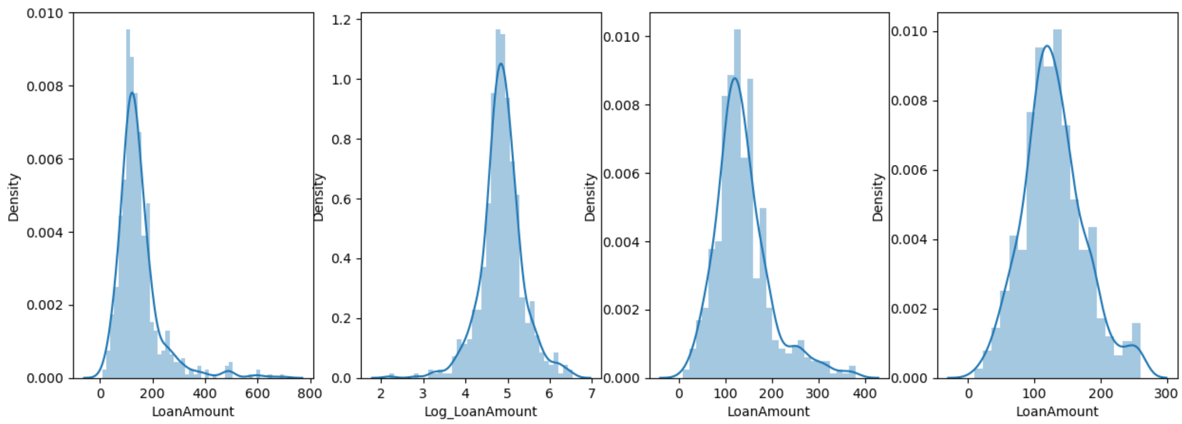
In [ ]:
```
# fig, axes = plt.subplots(1,4,figsize=(15,5))

# sns.boxplot(df['LoanAmount'])
# sns.boxplot(df1['Log_LoanAmount'])
# sns.boxplot(df2['LoanAmount'])
# sns.boxplot(df3['LoanAmount'])
# df1['LoanAmount'].plot.box(figsize=(16,5))
# plt.show()


plt.figure(1)
plt.subplot(141)
```

```
# sns.distplot(df['LoanAmount']);
df['LoanAmount'].plot.box(figsize=(16,5))
plt.subplot(142)
df1['Log_LoanAmount'].plot.box(figsize=(16,5))

plt.subplot(143)
df2['LoanAmount'].plot.box(figsize=(16,5))

plt.subplot(144)
df3['LoanAmount'].plot.box(figsize=(16,5))
plt.show()
```



In [ ]:

```
fig, axes = plt.subplots(1,4,figsize=(15,5))
sns.distplot(df['LoanAmount'], ax=axes[0])
sns.distplot(df1['Log_LoanAmount'], ax=axes[1])
sns.distplot(df2['LoanAmount'], ax=axes[2])
sns.distplot(df3['LoanAmount'], ax=axes[3])

plt.show()
```



1.3 Data Transformation

1.3.1 Scaling the Numerical Features

There are two ways to scale the data:

1. Standardization (Z-Score)
2. Normalization: Min Max Scalar

Both can by done manually as well as have in-built functions in sklearn. Will demonstrate both.

a. Standardization (Z-Score)

Scales the data using the formula (x-mean)/standard deviation.

In [ ]:
```
df4 = df3.copy()
numeral = ['LoanAmount','ApplicantIncome','CoapplicantIncome']
Z_numeral = ['Z_LoanAmount','Z_ApplicantIncome','Z_CoapplicantIncome']
df4[numeral].head()
```

Out[ ]:

	LoanAmount	ApplicantIncome	CoapplicantIncome
0	146.412162	5849.0	0.0
1	128.000000	4583.0	1508.0
2	66.000000	3000.0	0.0
3	120.000000	2583.0	2358.0
4	141.000000	6000.0	0.0

In [ ]:
```
from sklearn.preprocessing import StandardScaler

df4[Z_numeral] = StandardScaler().fit_transform(df4[numeral])

df4.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

In [ ]:
```
# checking if the skewness and kurtosis post scaling or not:

print("The skewness for the original data is {}.".format(df4.LoanAmount.skew()))
print("The kurtosis for the original data is {}.".format(df4.LoanAmount.kurt()))

print('')

print("The skewness for the Zscore Scaled column is {}.".format(df4.Z_LoanAmount.sl
print("The kurtosis for the Zscore Scaled columns is {}.".format(df4.Z_LoanAmount.l
```

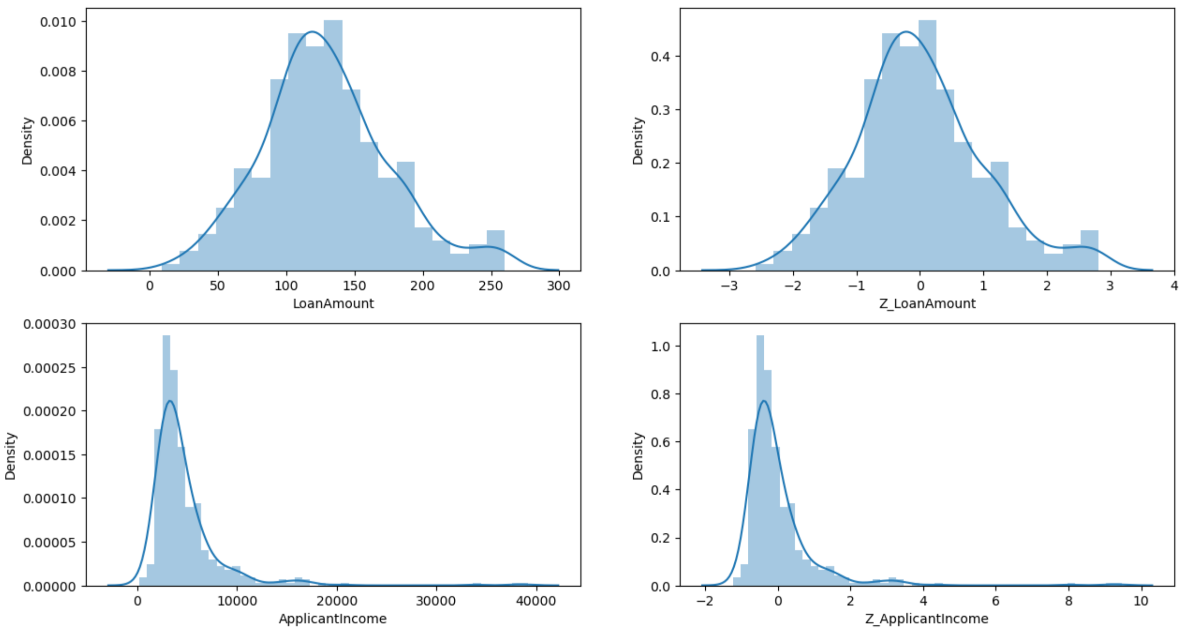
```
The skewness for the original data is 0.43191097222093977.
The kurtosis for the original data is 0.3657918163665843.

The skewness for the Zscore Scaled column is 0.4319109722209412.
The kurtosis for the Zscore Scaled columns is 0.36579181636658564.
```

In [ ]:
```
# checking if the skewness and kurtosis post scaling or not:

print("The skewness for the original data is {}.".format(df4.ApplicantIncome.skew(
print("The kurtosis for the original data is {}.".format(df4.ApplicantIncome.kurt(

print('')
```

```
print("The skewness for the Zscore Scaled column is {}.".format(df4.Z_ApplicantInco
print("The kurtosis for the Zscore Scaled columns is {}.".format(df4.Z_ApplicantInc
```

```
The skewness for the original data is 4.657677116718219.
The kurtosis for the original data is 33.56075734515245.

The skewness for the Zscore Scaled column is 4.657677116718219.
The kurtosis for the Zscore Scaled columns is 33.56075734515245.
```

In [ ]:
```
# Distribution of the columns

fig, axes = plt.subplots(2,2, figsize=(15,8))

sns.distplot(df4['LoanAmount'], ax=axes[0,0])
sns.distplot(df4['Z_LoanAmount'], ax=axes[0,1])
sns.distplot(df4['ApplicantIncome'], ax=axes[1,0])
sns.distplot(df4['Z_ApplicantIncome'], ax=axes[1,1])


plt.show()
```



The only difference between the two curves is of the Range on the x-axis. The impact of scaling on data is: Skewness, Kurtosis and Distribution all remain same.

The need for Scaling is :

1. Comparison between variables is easier\
2. Computation power is more efficient and less time consuming.

Documentation for Standard Scaler: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

b. Normalization: Min Max Scalar

Scales the data using the formula (x - min)/(max - min)

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
model = MinMaxScaler()
Min_Max_numeral = ['Min_Max_LoanAmount','Min_Max_ApplicantIncome','Min_Max_Coapplic
```

```
df4[Min_Max_numeral] = model.fit_transform(df4[numeral])
df4.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849.0	0.0
1	Male	Yes	1	Graduate	No	4583.0	1508.0
2	Male	Yes	0	Graduate	Yes	3000.0	0.0
3	Male	Yes	0	Not Graduate	No	2583.0	2358.0
4	Male	No	0	Graduate	No	6000.0	0.0

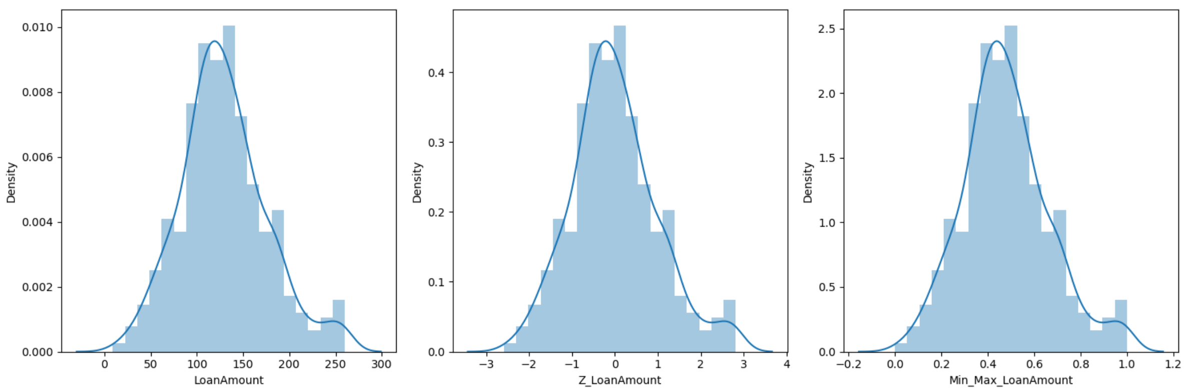
In [ ]:

```
# checking if the skewness and kurtosis post scaling or not:

print("The skewness for the original data is {}.".format(df4.LoanAmount.skew()))
print("The skewness for the original data is {}.".format(df4.Z_LoanAmount.skew()))
print("The skewness for the Min Max Scaled Data is {}.".format(df4.Min_Max_LoanAmou

print('')

print("The kurtosis for the Zscore Scaled column is {}.".format(df4.LoanAmount.kurt
print("The kurtosis for the Zscore Scaled columns is {}.".format(df4.Z_LoanAmount.k
print("The kurtosis for the Min Max Scaled Data is {}.".format(df4.Min_Max_LoanAmou
```

```
The skewness for the original data is 0.43191097222093977.
The skewness for the original data is 0.4319109722209412.
The skewness for the Min Max Scaled Data is 0.43191097222093994.

The kurtosis for the Zscore Scaled column is 0.3657918163665843.
The kurtosis for the Zscore Scaled columns is 0.36579181636658564.
The kurtosis for the Min Max Scaled Data is 0.3657918163665834.
```

In [ ]:

```
# Distribution of the columns

# For Rate

fig, axes = plt.subplots(1,3, figsize=(15,5))

sns.distplot(df4['LoanAmount'], ax=axes[0])
sns.distplot(df4['Z_LoanAmount'], ax=axes[1])
sns.distplot(df4['Min_Max_LoanAmount'], ax=axes[2])

plt.tight_layout()
plt.show()
```



Documentation for Min Max Scaler: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Few things to keep in mind:

With Scaling all three - Skewness, Kurtosis and distribution remain same so there is no impact on outliers as well.

1.3.2 Encoding the Categorical Features

There are two ways to encode the categorical data into dummy variables. Using:

1. pd.get_dummies\
2. sklearn's in-built function of OneHotEncoder and LabelEncoder

In [ ]:
```
# Loans data:
df_loans = df3.copy()
df_loans[numeral] = StandardScaler().fit_transform(df_loans[numeral])
df_loans.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	0.328911	-0.630154
1	Male	Yes	1	Graduate	No	-0.018351	0.012474
2	Male	Yes	0	Graduate	Yes	-0.452565	-0.630154
3	Male	Yes	0	Not Graduate	No	-0.566948	0.374698
4	Male	No	0	Graduate	No	0.370330	-0.630154

In [ ]:
```
df_loans.dtypes
```

Out[ ]:
```
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
TotalIncome          float64
dtype: object
```

1) pd.get_dummies approach:

In [ ]:
```
df_loans = pd.get_dummies(df_loans, columns=['Gender','Married','Property_Area'],dr

df_loans.head()
```

```
# drop_first = True drops the first column for each feature
```

Out[ ]:

	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	0	Graduate	No	0.328911	-0.630154	0.370646	
1	1	Graduate	No	-0.018351	0.012474	-0.025618	
2	0	Graduate	Yes	-0.452565	-0.630154	-1.359976	
3	0	Not Graduate	No	-0.566948	0.374698	-0.197794	
4	0	Graduate	No	0.370330	-0.630154	0.254166	

2) OneHot Encoding

Documentation for this: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

1.3.3 Label Encoding

Documentation for this: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
# Label Encoding

df_loans['Loan_Status'] = LabelEncoder().fit_transform(df_loans['Loan_Status'])
# Replacing
df_loans['Dependents'].replace(('0', '1', '2', '3+'),(0, 1, 2, 3),inplace=True)
df_loans['Education'].replace(('Not Graduate', 'Graduate'),(0, 1),inplace=True)
df_loans['Self_Employed'].replace(('No','Yes'),(0,1),inplace=True)


df_loans.head()
```

Out[ ]:

	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	0	1	0	0.328911	-0.630154	0.370646	
1	1	1	0	-0.018351	0.012474	-0.025618	
2	0	1	1	-0.452565	-0.630154	-1.359976	
3	0	0	0	-0.566948	0.374698	-0.197794	
4	0	1	0	0.370330	-0.630154	0.254166	

In [ ]:

```
df_loans.dtypes
```

Out[ ]:
```
Dependents                  int64
Education                   int64
Self_Employed               int64
ApplicantIncome           float64
CoapplicantIncome         float64
LoanAmount                float64
Loan_Amount_Term          float64
Credit_History            float64
Loan_Status                 int32
TotalIncome               float64
Gender_Male                 uint8
Married_Yes                 uint8
Property_Area_Semiurban     uint8
Property_Area_Urban         uint8
dtype: object
```

1.3.4 Replacing

In [ ]:
```
# Replacing
df_loans['Dependents'].replace(('0', '1', '2', '3+'),(0, 1, 2, 3),inplace=True)
df_loans['Education'].replace(('Not Graduate', 'Graduate'),(0, 1),inplace=True)
df_loans['Self_Employed'].replace(('No','Yes'),(0,1),inplace=True)
df_loans.head()
```

Out[ ]:

	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	0	1	0	0.328911	-0.630154	0.370646	
1	1	1	0	-0.018351	0.012474	-0.025618	
2	0	1	1	-0.452565	-0.630154	-1.359976	
3	0	0	0	-0.566948	0.374698	-0.197794	
4	0	1	0	0.370330	-0.630154	0.254166	

In [ ]:
```
df_loans.dtypes
```

Out[ ]:
```
Dependents                  int64
Education                   int64
Self_Employed               int64
ApplicantIncome           float64
CoapplicantIncome         float64
LoanAmount                float64
Loan_Amount_Term          float64
Credit_History            float64
Loan_Status                 int32
TotalIncome               float64
Gender_Male                 uint8
Married_Yes                 uint8
Property_Area_Semiurban     uint8
Property_Area_Urban         uint8
dtype: object
```

1.4 Training and Testing data

Documentation for this: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Extracting dependent and independent variables

In machine learning, it is important to differentiate the matrix of features (independent variables) and dependent variables from dataset.

In the simple, Loan_Status is the dependent variable and the others are the independent variables

In [ ]:
```
## Splitting for X and Y variables:
from sklearn.model_selection import train_test_split
Y = df_loans['Loan_Status']
X = df_loans.drop('Loan_Status', axis=1)
```

In [ ]:
```
# Independent Variable

X.head()
```

Out[ ]:

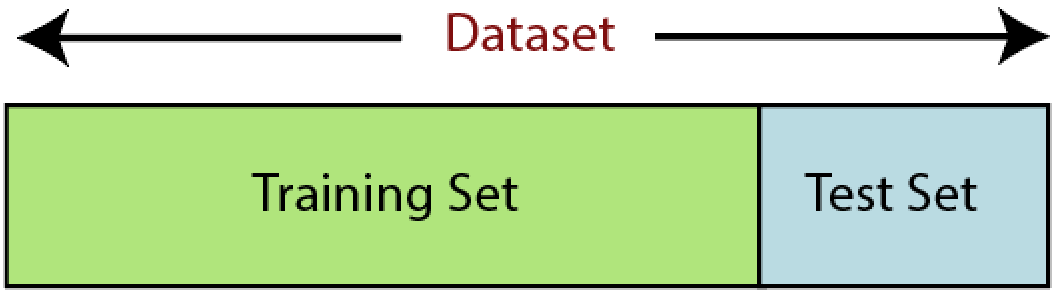
	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	0	1	0	0.328911	-0.630154	0.370646	
1	1	1	0	-0.018351	0.012474	-0.025618	
2	0	1	1	-0.452565	-0.630154	-1.359976	
3	0	0	0	-0.566948	0.374698	-0.197794	
4	0	1	0	0.370330	-0.630154	0.254166	

In [ ]:
```
# Dependent or Target Variable

Y.head()
```

Out[ ]:
```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int32
```

Splitting dataset

Splitting the dataset is the next step in data preprocessing in machine learning. Every dataset for Machine Learning model must be split into two separate sets – training set and test set.



Trainning and Testing data

In [ ]:
```
## Splitting dataset into 80% Training and 20% Testing Data:

X_train, X_test, y_train, y_test = train_test_split(X,Y,train_size=0.8, random_stat

# random_state ---> is seed -- fixing the sample selection for Training & Testing d

# check the dimensions of the train & test subset for

print("The shape of X_train is:", X_train.shape)
print("The shape of X_test is:", X_test.shape)

print('')
print("The shape of y_train is:", y_train.shape)
print("The shape of y_test is:", y_test.shape)
```

```
The shape of X_train is: (458, 13)
The shape of X_test is: (115, 13)

The shape of y_train is: (458,)
The shape of y_test is: (115,)
```

Conclusion:

Based on the above result, we can conclude statistically that the train and test representative of the overall data as the median for both y_train and y_test are similar. We have successfully divided the dataset into training and testing dataset. Now, we will be using the classification Models for predicting the Loan aprovals.

---

2 Training classification model

Logistic regression classification

Importing the required libraries for model preparation.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

We will be utilizing the Logistic regression classification model for our dataset and predict the loan approvals.

In [ ]:
```
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
y_prediction = model.predict(X_test)
print('Logistic Regression accuracy = ', metrics.accuracy_score(y_prediction,y_test
```

```
Logistic Regression accuracy =  0.8608695652173913
```

The accuracy for the logistic regression model turns out to be 0.8852760 (i.e., Approximately 88%)

Decision tree classifier

Importing the required libraries for model preparation.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
```

We will be utilizing the Logistic regression classification model for our dataset and predict the loan approvals.

In [ ]:
```
model = GaussianNB()

model.fit(X_train,y_train)

y_prediction = model.predict(X_test)

print('Logistic Regression accuracy = ', metrics.accuracy_score(y_prediction,y_test
```

```
Logistic Regression accuracy =  0.8782608695652174
```

The accuracy for the logistic regression model turns out to be 0.87826 (i.e., Approximately 88%)

3 Evaluation classification model

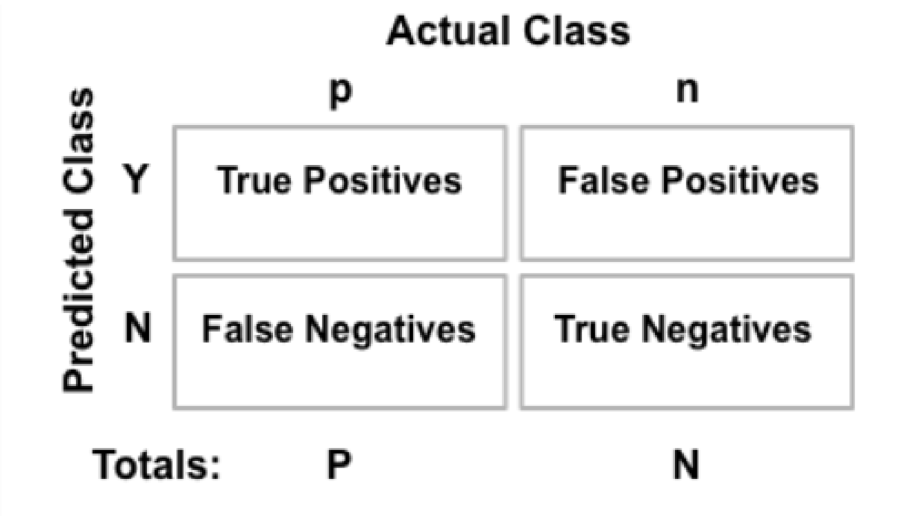


Figure 2. The confusion matrix

1) Accuracy

Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

In Python you can calculate it in the following way:

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = model.predict_proba(X_test)#Return probability estimates for the test vect
threshold = 0.8
y_pred_class = y_pred[:, 1]   > threshold
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_class).ravel()
accuracy = (tp+ tn) / (tp + fp + fn + tn)

# Or simply
accuracy_score(y_test, y_pred_class)
```

Out[ ]:
```
0.8521739130434782
```

2) Precision:
Precision tells us what proportion of messages we classified as positive. It is a ratio of true positives to all positive predictions. In other words,

$$Precision = TP/(TP + FP)$$

3) Recall:
Recall(sensitivity) tells us what proportion of messages that actually were positive were classified by us as positive.

$$Recall = TP/(TP + FN)$$

4) F1 score:

We can use **F-beta** score as a metric that considers both precision and recall:

$$F_\beta = (1 + \beta^2) \bullet \frac{precision \bullet recall}{(\beta^2 \bullet precision) + recall}$$

When choosing beta in your F-beta score **the more you care about recall** over precision **the higher beta** you should choose. For example, with **F1 score** **we care equally about recall and precision** with F2 score, recall is twice as important to us.

5) TPR & FPR & ROC & AUC:

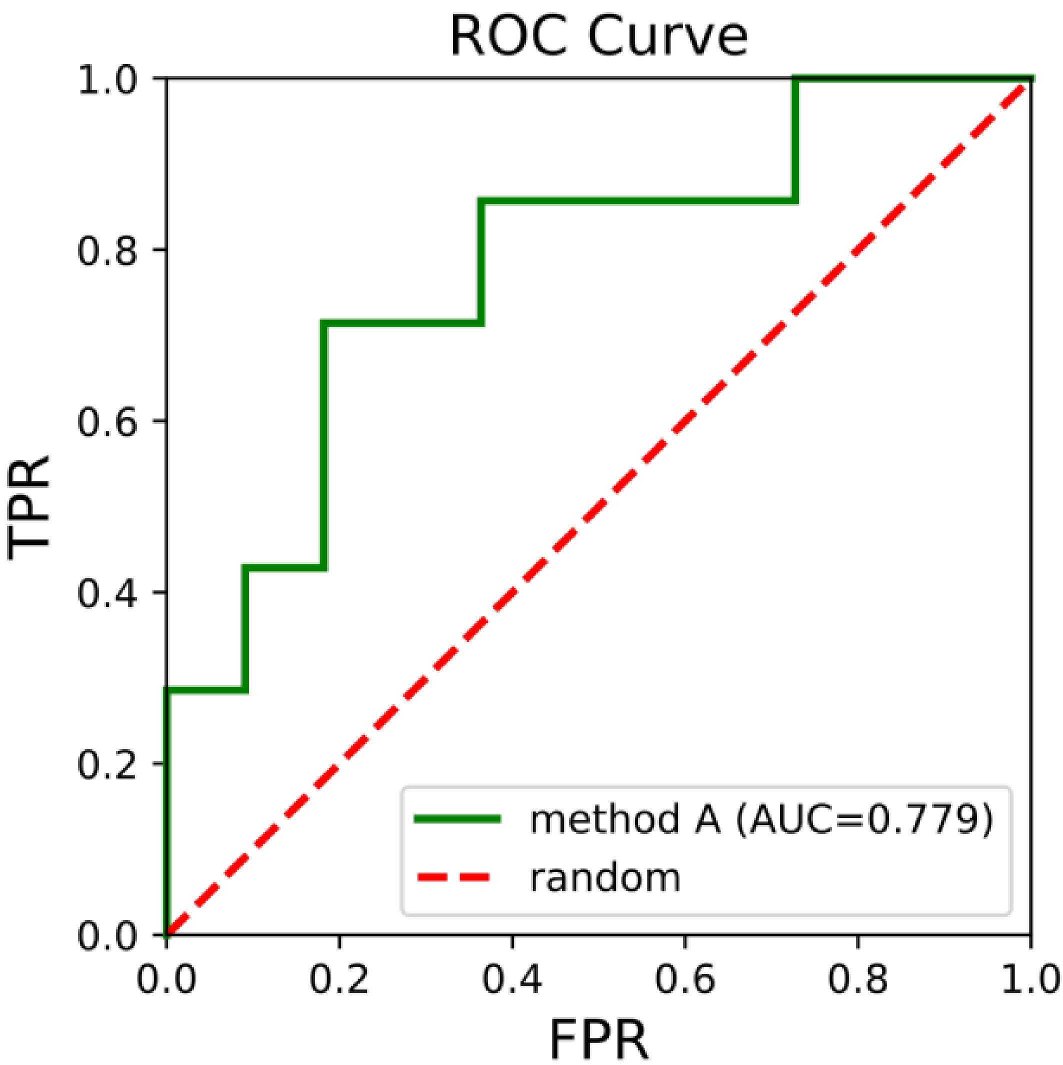
$$TPR = \frac{positives\ correctly\ classified}{total\ positives} = \frac{TP}{TP + FN} = \frac{TP}{P}$$

$$FPR = \frac{negatives\ incorrectly\ classified}{total\ negatives} = \frac{FP}{TN + FP} = \frac{FP}{N}$$

ROC (Receiver Operating Characteristic is used to measure the output quality of the evaluation classifier. ROC curves are two-dimensional graphs in which true positive rate (TPR) is ploted on the Y axis and false positive rate (FPR) is plotted on the X axis. An ROC graph depicts relative tradeoffs between true positive rate (TPR) and false positive rate (FPR). Basically, for every threshold, we calculate TPR and FPR and plot it on one chart.

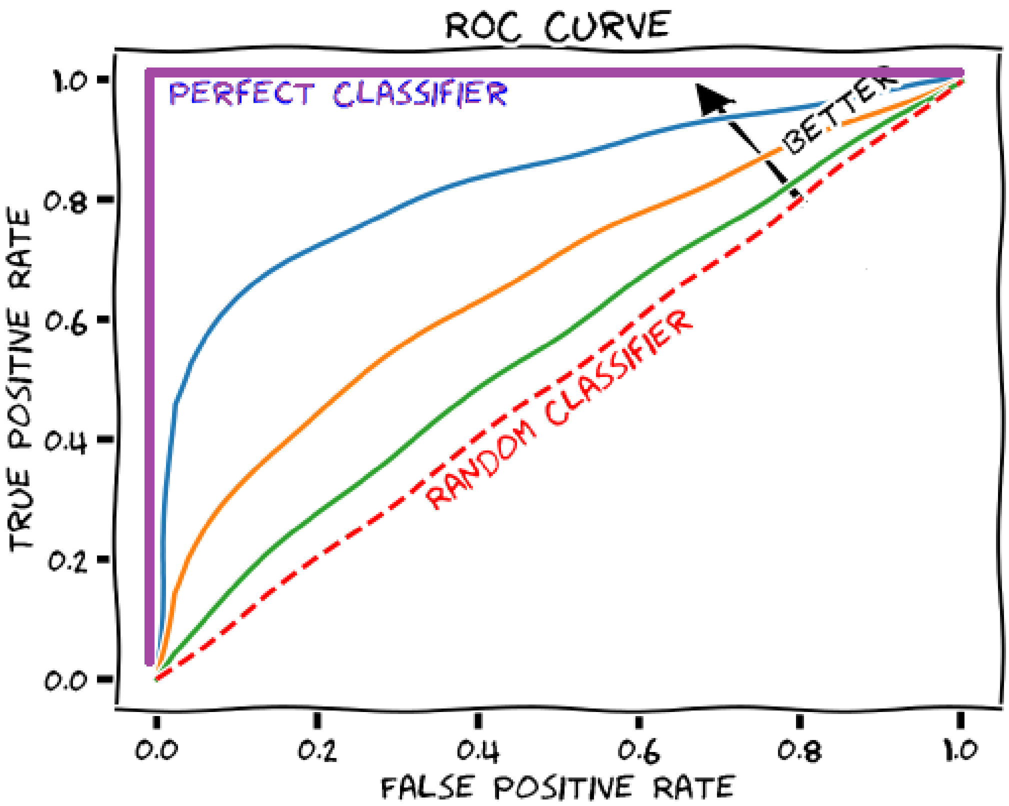
ID	Actual	Prediction Probability	>0.6	>0.7	>0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR

Example data and curve for ROC



Example of ROC curve

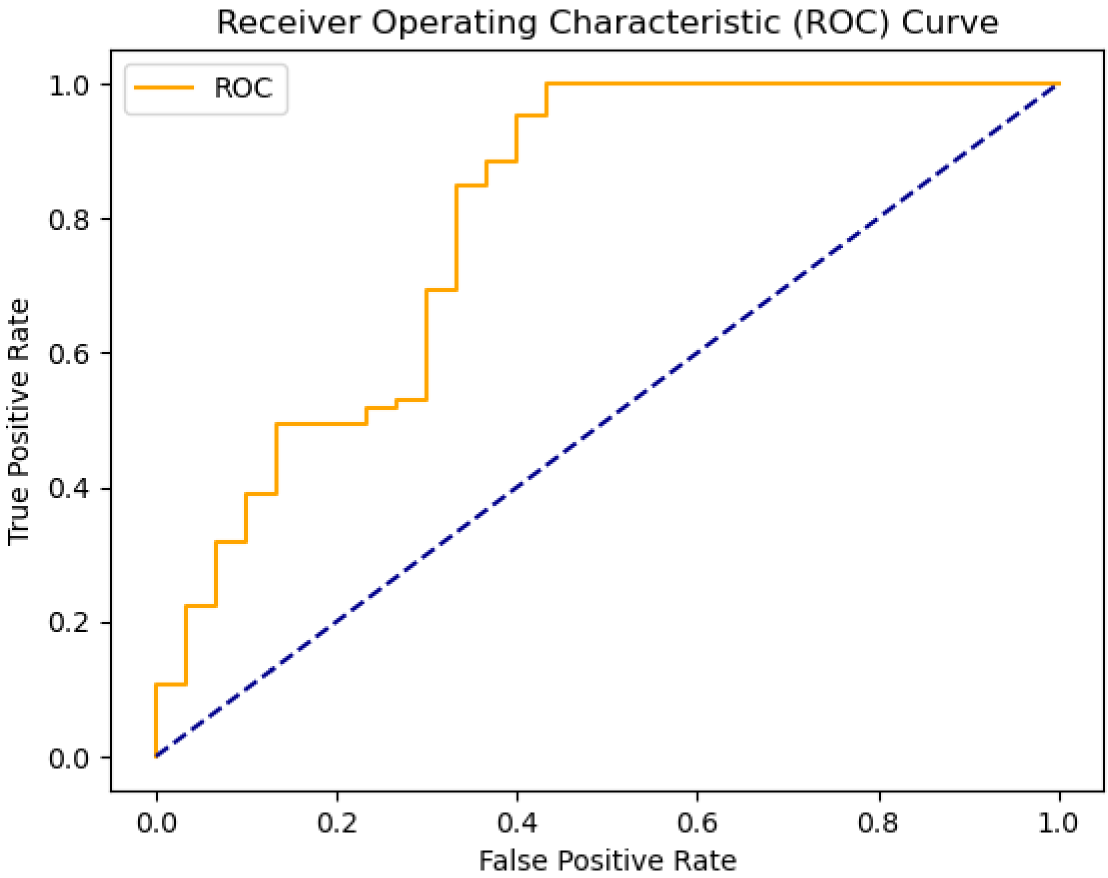
The higher TPR and the lower FPR is for each threshold the better and so classifiers that have curves that are more top-left-side are better.



Example of ROC curve

AUC (Area Under Curve) means area under the curve, it is a performance metric that you can use to evaluate classification models. There are functions for calculating AUC available in many programming languages.
In python, you can refer to [document from sklearn](#).

In [ ]:
```
fper, tper, thresholds = metrics.roc_curve(y_test, y_pred[:, 1])
plt.plot(fper, tper, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



4 LAB Assignment

This part requires you to complete and submit by yourself according to the template.

You will employ several supervised algorithms of your choice to accurately model individuals' income using data collected from the 1994 U.S census (census.csv). Your goal with this lab is to construct a model that accurately predicts whether an individual makes more than $50000.

Exercise 0 Importing the census

Note that the last column from this dataset "income", will be our target label (whether an individual makes more than, or at most, $50,000 annually). All other columns are features about each individual in the census database.

Exercise 1 Exploration

A cursory investigation of the dataset will determine how many individuals fit into either group, and will tell us about the percentage of these individuals making more than $50,000 annually. In the code cell below, you will need to compute the following:

- The total number of records, `n_records` ;
- The number of individuals making more than $50000 annually, `n_greater_50k` .
- The number of individuals making at most $50000 annually, `n_at_most_50K` .
- The percentage of individuals making at more than $50000 annually, `greater_percent`

- Feature values for each column

Tips :As the data is stored as pandas, this tutorial will help you finish.

Exercise 2 Preprocessing

- Before the data can be used as the input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as preprocessing. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

- Transforming Skewed Continuous Features. A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this description: `capital-gain` and `capital-loss` .

- Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so we must translate the values by a small amount above 0 to apply the the logarithm successfully.

- Normalizing Numerical Features. In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as `capital-gain` or `capital-loss` above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning.

- Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called 'categorical variables') be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a 'dummy' variable for each possible category of each non-numeric feature. For example, assume some features has three possible entries: A, B and C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

	someFeature		someFeature_A	someFeature_B	someFeature_C
0	B		0	1	0
1	C	----> one-hot encode ---->	0	0	1
2	A		1	0	0

Additionally, as with the non-numeric features, we need to convert the non-numeric target label, 'income' to numerical values for the learning algorithm to work. Since there are only two possible categories for this label ("<=50K" and ">50K"), we can avoid using one-hot encoding and simply encode these two categories as 0 and 1, respectively.

	age	education-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	...	native-country_Portugal	native-country_Puerto-Rico	native-country_Scotland	native-country_South	native-country_Taiwan	native-country_Thailan
	0.301370	0.800000	0.667492	0.0	0.397959	0	0	0	0	0	...	0	0	0	0	0	
	0.452055	0.800000	0.000000	0.0	0.122449	0	0	0	0	1	...	0	0	0	0	0	
	0.287671	0.533333	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	
	0.493151	0.400000	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	
	0.150685	0.800000	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	

Exercise 3 Shuffle and Split Data

When all categorical variables have been converted into numerical features, and all numerical features have been normalized. As always, we will now split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

Exercise 4 :

Now we chose a model that always predicted an individual made more than $50,000, what would that model's accuracy and F-score be on this dataset? You must use the code cell below and assign your results to 'accuracy' and 'f-score' to be used later.

Exercise 5 Evaluating Model

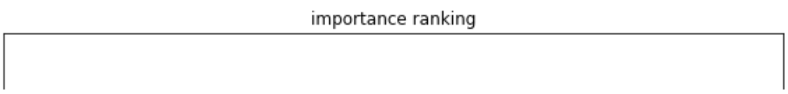
Now if we assume a model that predicts any individual's income more than $50,000, then what would be that model's accuracy and F-score on this dataset? You can use the code provided in the previous section. The following are some of the supervised learning models that are currently available in `scikit-learn`:

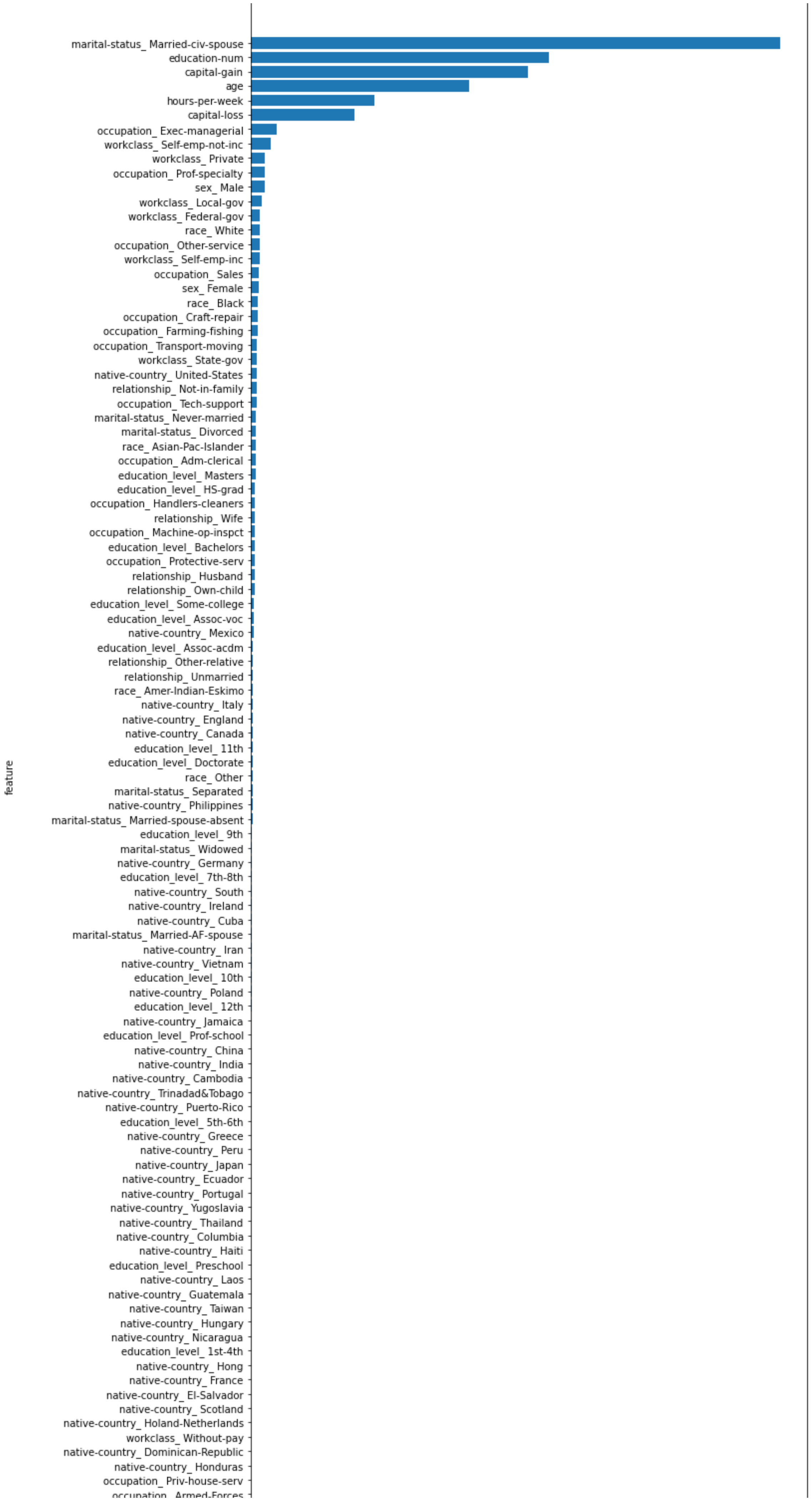
- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, RandomForest)
- K-Nearest Neighbors
- Support Vector Machines (SVM)
- Logistic Regression

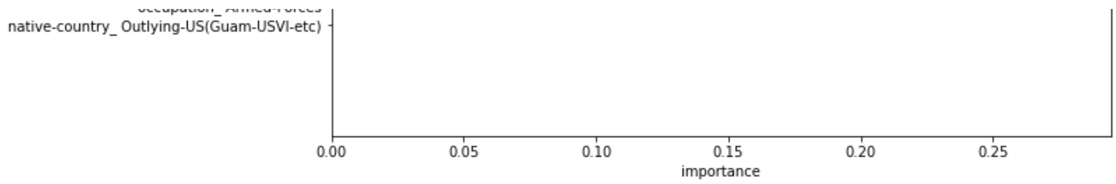
You need choose three of them, draw three ROC curves on the census data, and analyze and compare the them.

Exercise 6 Questions

(1) An important task when performing supervised learning on a dataset like the census data we study here is determining which features provides the most predictive power. Choose a scikit-learn classifier (e.g adaboost, random forests) that has a feature_importance_ attribute, which is a function that ranks the importance of features according to the chosen classifier. List two supervised learning models that apply to this problem, and you will test them on census data and plot the following graph.




marital-status_ Married-civ-spouse
education-num
capital-gain
age
hours-per-week
capital-loss
occupation_ Exec-managerial
workclass_ Self-emp-not-inc
workclass_ Private
occupation_ Prof-specialty
sex_ Male
workclass_ Local-gov
workclass_ Federal-gov
race_ White
occupation_ Other-service
workclass_ Self-emp-inc
occupation_ Sales
sex_ Female
race_ Black
occupation_ Craft-repair
occupation_ Farming-fishing
occupation_ Transport-moving
workclass_ State-gov
native-country_ United-States
relationship_ Not-in-family
occupation_ Tech-support
marital-status_ Never-married
marital-status_ Divorced
race_ Asian-Pac-Islander
occupation_ Adm-clerical
education_level_ Masters
education_level_ HS-grad
occupation_ Handlers-cleaners
relationship_ Wife
occupation_ Machine-op-inspct
education_level_ Bachelors
occupation_ Protective-serv
relationship_ Husband
relationship_ Own-child
education_level_ Some-college
education_level_ Assoc-voc
native-country_ Mexico
education_level_ Assoc-acdm
relationship_ Other-relative
relationship_ Unmarried
race_ Amer-Indian-Eskimo
native-country_ Italy
native-country_ England
native-country_ Canada
education_level_ 11th
education_level_ Doctorate
race_ Other
marital-status_ Separated
native-country_ Philippines
marital-status_ Married-spouse-absent
education_level_ 9th
marital-status_ Widowed
native-country_ Germany
education_level_ 7th-8th
native-country_ South
native-country_ Ireland
native-country_ Cuba
marital-status_ Married-AF-spouse
native-country_ Iran
native-country_ Vietnam
education_level_ 10th
native-country_ Poland
education_level_ 12th
native-country_ Jamaica
education_level_ Prof-school
native-country_ China
native-country_ India
native-country_ Cambodia
native-country_ Trinadad&Tobago
native-country_ Puerto-Rico
education_level_ 5th-6th
native-country_ Greece
native-country_ Peru
native-country_ Japan
native-country_ Ecuador
native-country_ Portugal
native-country_ Yugoslavia
native-country_ Thailand
native-country_ Columbia
native-country_ Haiti
education_level_ Preschool
native-country_ Laos
native-country_ Guatemala
native-country_ Taiwan
native-country_ Hungary
native-country_ Nicaragua
education_level_ 1st-4th
native-country_ Hong
native-country_ France
native-country_ El-Salvador
native-country_ Scotland
native-country_ Holand-Netherlands
workclass_ Without-pay
native-country_ Dominican-Republic
native-country_ Honduras
occupation_ Priv-house-serv
feature



(2) Describe one real-world application in industry where a model can be applied

(3) What are the strengths of the model; when does it perform well?

(4) What are the weaknesses of the model; when does it perform poorly?

(5) What makes this model a good candidate for the problem, given what you know about the data?