**1.  Problem Definition :**

A Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers. Here they have provided a data set.

**2. Data Analysis :**

Data analysis is a process of inspecting, cleansing, transforming and modeling data with the goal of discovering useful information, informing conclusions and supporting decision-making.

Data mining is a particular data analysis technique that focuses on statistical modeling and knowledge discovery for predictive rather than purely descriptive purposes, while business intelligence covers data analysis that relies heavily on aggregation, focusing mainly on business information. In statistical applications, data analysis can be divided into descriptive statistics, exploratory data analysis(EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data while CDA focuses on confirming or falsifying existing hypotheses.

2.1.**Used library Descriptions** :

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

2.2.**Variable Descriptions** :

| **Variable** | **Description** |
| --- | --- |
| Loan\_ID | Unique Loan ID |
| Gender | Male/ Female |
| Married | Applicant married (Y/N) |
| Dependents | Number of dependents |
| Education | Applicant Education (Graduate/ Under Graduate) |
| Self\_Employed | Self employed (Y/N) |
| ApplicantIncome | Applicant income |
| CoapplicantIncome | Coapplicant income |
| LoanAmount | Loan amount in thousands |
| Loan\_Amount\_Term | Term of loan in months |
| Credit\_History | credit history meets guidelines |
| Property\_Area | Urban/ Semi Urban/ Rural |
| Loan\_Status | Loan approved (Y/N) |

2.3.**Data Descriptions** :

In Python we can know data type using pandas property **DataFrame.dtypes**

>>**bank.dtypes**

Loan\_ID object

Gender object

Married object

Dependents object

Education object

Self\_Employed object

ApplicantIncome int64

CoapplicantIncome float64

LoanAmount float64

Loan\_Amount\_Term float64

Credit\_History float64

Property\_Area object

Loan\_Status object

dtype: object

2.4.**Data Information Descriptions** :

In Python we can know data information using pandas method DataFrame.info()

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

>>**bank.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

2.5.**Data shape Descriptions** :

In Python we can know data shape using pandas property DataFrame.shape

This return a tuple representing the dimensionality of the DataFrame.

>>**bank.shape**

(614, 13)

The given dataset shape is 614 row and 13 columns value (614 x 13).

2.6.**Data Statistical Descriptions** :

In Python we can know data Descriptive statistics using pandas property DataFrame.describe()

This provide Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.

>> **bank.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 | 146.412162 | 342.000000 | 0.842199 |
| **std** | 6109.041673 | 2926.248369 | 84.037468 | 64.372489 | 0.349681 |
| **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 | 129.000000 | 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 |

2.7.**Data Null Value Descriptions** :

In Python we can know data null values using pandas method DataFrame.isnull()

It's detect missing values for an array-like object.

**>>print(bank.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

In this dataset there are 13 value is null in Gender variable, 3 value is null in Married variable, 15 value is null in Dependents variable, 32 value is null in Self\_Employed variable, 22 value is null in LoanAmount variable, 14 value is null in Loan\_Amount\_Term variable and 50 value is null in Credit\_History variable.

2.8.**Data Imputation Descriptions** :

Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

>>bank['Gender'].fillna(bank['Gender'].mode()[0], inplace=True)

>>bank['Married'].fillna(bank['Married'].mode()[0], inplace=True)

>>bank['Dependents'].fillna(bank['Dependents'].mode()[0], inplace=True)

>>bank['Self\_Employed'].fillna(bank['Self\_Employed'].mode()[0], inplace=True)

>>mean\_LoanAmount=bank['LoanAmount'].mean()

>>LoanAmount=float(mean\_LoanAmount)

>>bank['LoanAmount'].fillna(LoanAmount,inplace=True)

>>mean\_Loan\_Amount\_Term=bank['Loan\_Amount\_Term'].mean()

>>Loan\_Amount\_Term=float(mean\_Loan\_Amount\_Term)

>>bank['Loan\_Amount\_Term'].fillna(Loan\_Amount\_Term,inplace=True)

>>mean\_Credit\_History=bank['Credit\_History'].mean()

>>Credit\_History=float(mean\_Credit\_History)

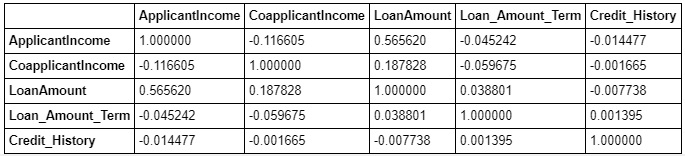
>>bank['Credit\_History'].fillna(Credit\_History,inplace=True)

2.9.**Data Correlation Descriptions** :

Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related.

>>corrmat = bank.corr()

>>corrmat



Alternate way to represent correlation using heatmap

# Correlation Plot

>>corrmat = bank.corr()

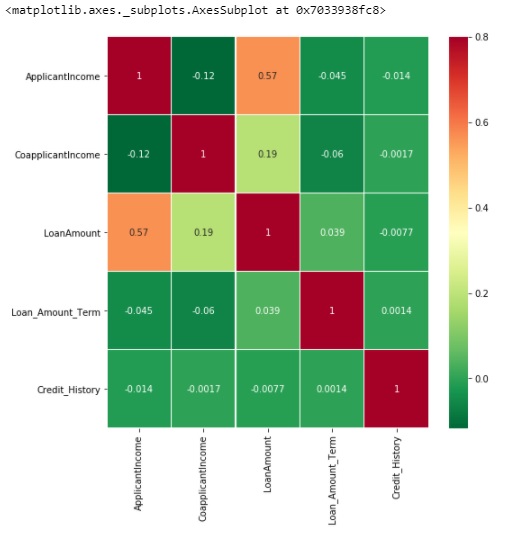
#Set uo the matplolib figure

>>f, ax = plt.subplots(figsize=(8,8))

#Draw the heatmap using seaborn

>>colormap = plt.cm.RdYlGn\_r

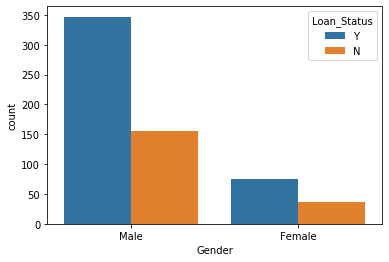
>>sns.heatmap(corrmat, linewidths=0.1, cmap = colormap, linecolor = 'White', vmax=0.8, annot=True)



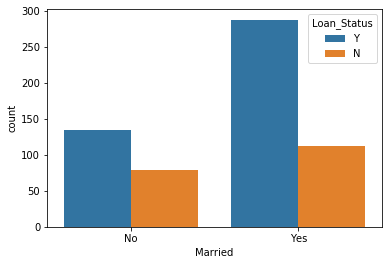
2.10 .**Data visualization Descriptions** :

**Python** allows us to create **visualizations** easily and quickly using Matplotlib and Seaborn.

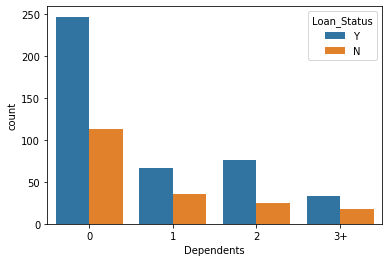
>>sns.countplot(x='Gender',data=bank,hue='Loan\_Status')



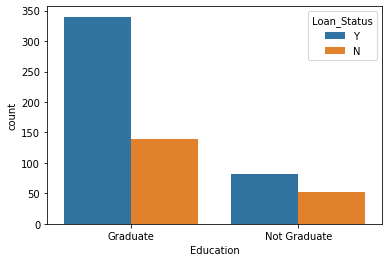
>>sns.countplot(x='Married',data=bank,hue='Loan\_Status')

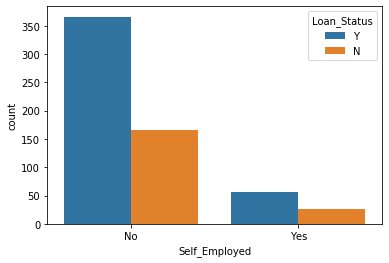


>>sns.countplot(x='Dependents',data=bank,hue='Loan\_Status')

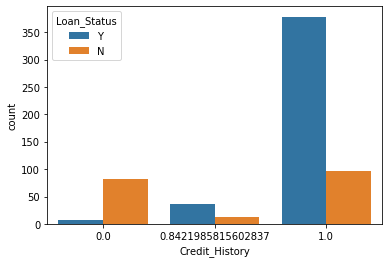


>>sns.countplot(x='Education',data=bank,hue='Loan\_Status')

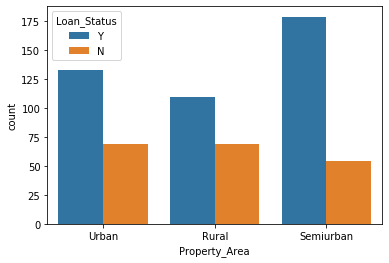


>>sns.countplot(x='Self\_Employed',data=bank,hue='Loan\_Status')

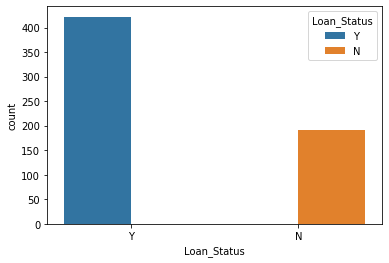
>>sns.countplot(x='Credit\_History',data=bank,hue='Loan\_Status')



>>sns.countplot(x='Property\_Area',data=bank,hue='Loan\_Status')



>>sns.countplot(x='Loan\_Status',data=bank,hue='Loan\_Status')



3. **Data Pre-processing Descriptions** :

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format. In this data set we LabelEncoder, OneHotEncoder for data pre processing .

>>from sklearn.preprocessing import LabelEncoder, OneHotEncoder

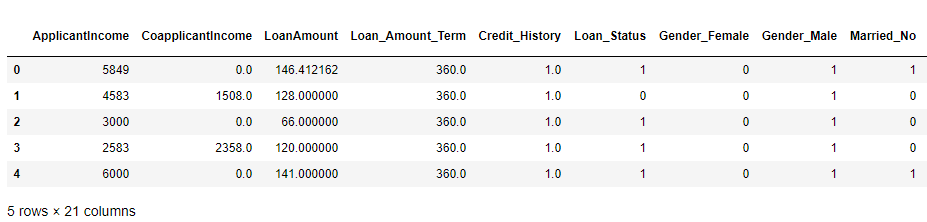
# Create a label encoder object

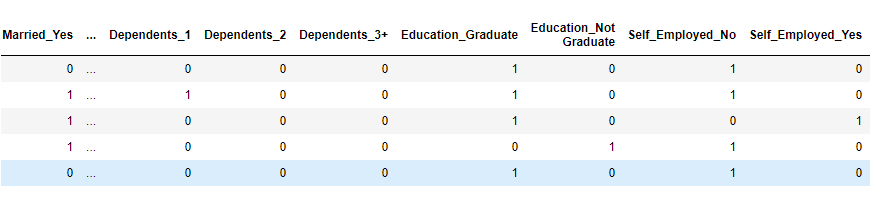
>>le = LabelEncoder()

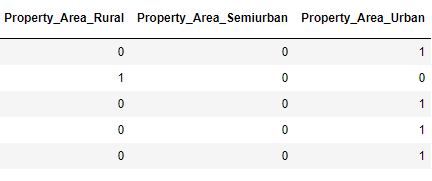
>>bank['Loan\_Status'] = le.fit\_transform(bank['Loan\_Status'])

>>bank\_new = pd.get\_dummies(bank.drop('Loan\_ID', axis = 1))

>>bank\_new.head()







Now we can separate input variables and target variable.

>>Y = bank\_new['Loan\_Status']

>>x = bank\_new.drop('Loan\_Status', axis = 1)

>>from sklearn.preprocessing import StandardScaler

>>sc = StandardScaler()

>>X\_std = sc.fit\_transform(x)

>>X = pd.DataFrame(X\_std,columns=x.columns)

4. **Building Machine Learning Models** :

#Prediction - Classification Algorithms

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC, LinearSVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

>>from sklearn import metrics

#Split the data into train and test set for classifcation predictions

>>from sklearn.model\_selection import train\_test\_split,

cross\_val\_score

>>x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0.3,random\_state=9,stratify=Y)

>> x\_train.shape, y\_train.shape, x\_test.shape

>>KNN=KNeighborsClassifier(n\_neighbors=6)

>>SV=SVC()

>>LR=LogisticRegression()

>>DT=DecisionTreeClassifier(random\_state=6)

>>GNB=GaussianNB()

>>RFC=RandomForestClassifier(n\_estimators=1000,random\_state=0)

Model=[]

score=[]

cvs=[]

rocscore=[]

for name,model in models:

print('###############################',name,'##################################\n')

Model.append(name)

model.fit(x\_train,y\_train)

print(model)

pre=model.predict(x\_test)

print('\n')

AS=accuracy\_score(y\_test,pre)

print('Accuracy\_score= ',AS)

score.append(AS\*100)

print('\n')

sc=cross\_val\_score(model,X,Y,cv=10,scoring='accuracy').mean()

print('Cross\_val\_score=',sc)

cvs.append(sc\*100)

print('\n')

false\_positive\_rate,true\_positive\_rate,thresholds=roc\_curve(y\_test,pre)

roc\_auc=auc(false\_positive\_rate,true\_positive\_rate)

print('roc\_auc\_score= ',roc\_auc)

rocscore.append(roc\_auc\*100)

print('\n')

print('classification\_report\n',classification\_report(y\_test,pre),'\n')

cm=confusion\_matrix(y\_test,pre)

print('Confusion Matrix\n',cm,'\n')

plt.figure(figsize=(10,40))

plt.subplot(911)

plt.title(name)

plt.plot(false\_positive\_rate,true\_positive\_rate,label='AUC=%0.2f'%roc\_auc)

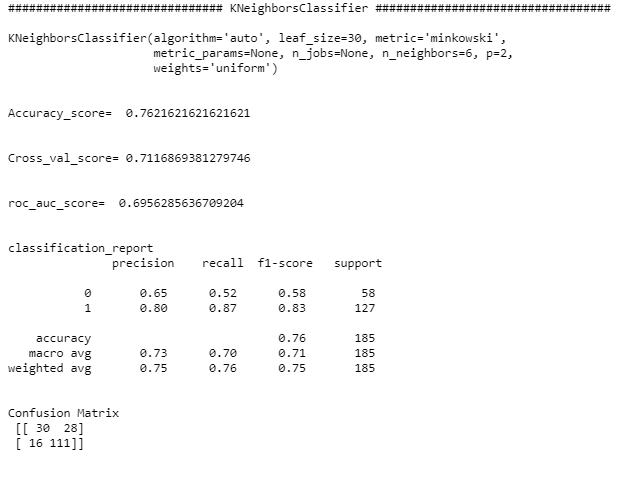
plt.plot([0,1],[0,1],'r--')

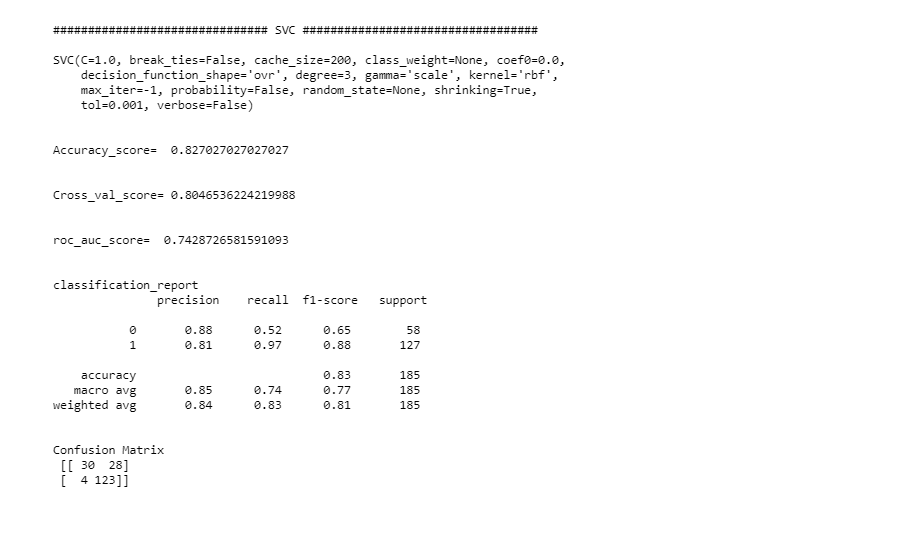
plt.legend(loc='lower right')

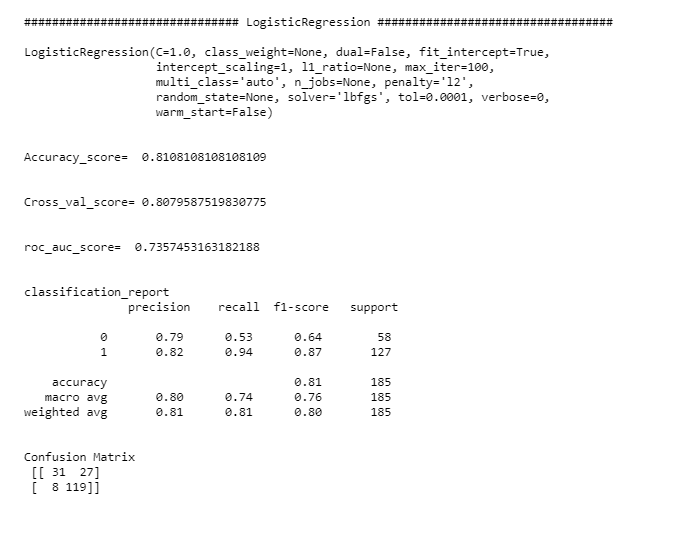
plt.ylabel('True Positive rate')

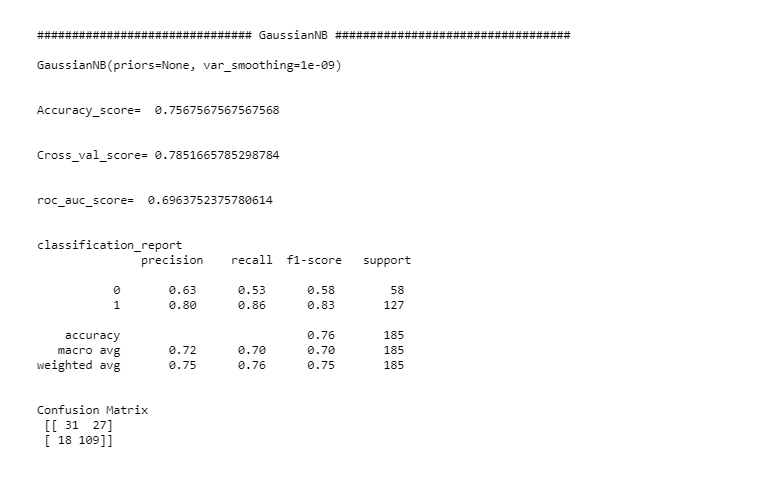
plt.xlabel('false Positive rate')

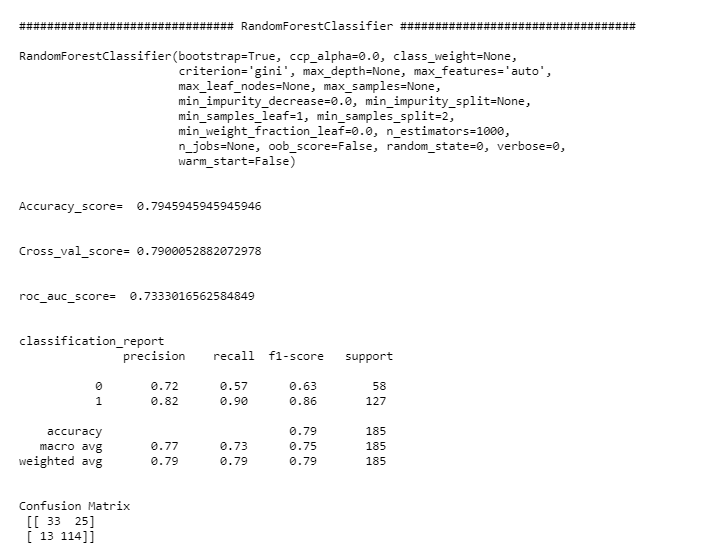
print('\n\n')

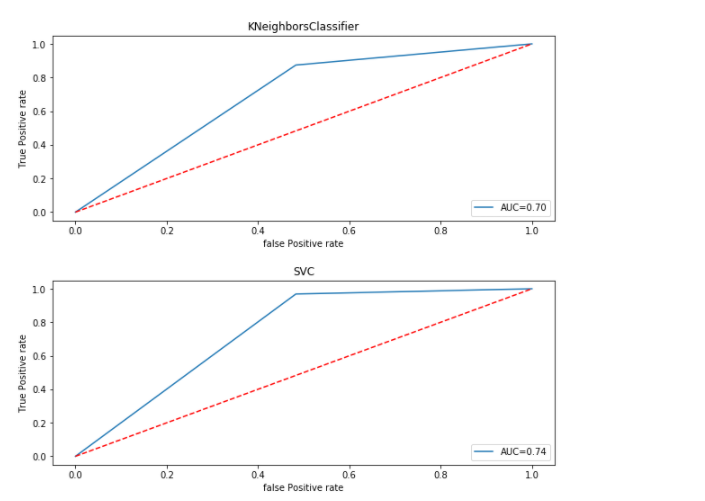


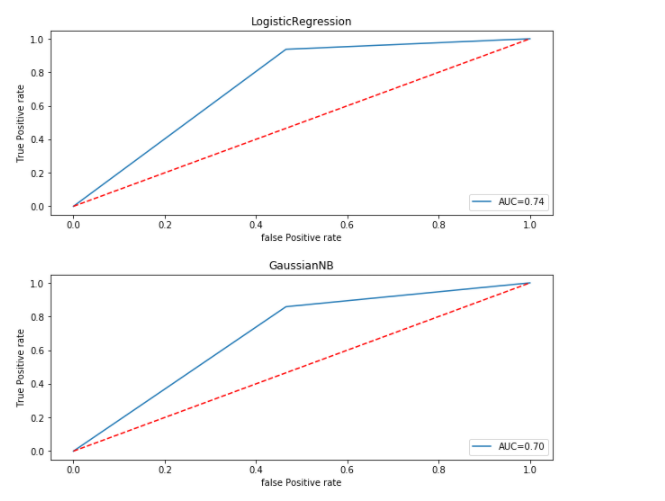


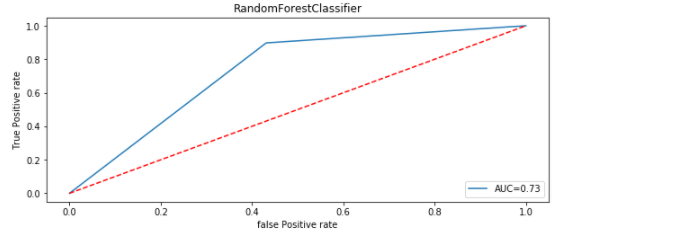


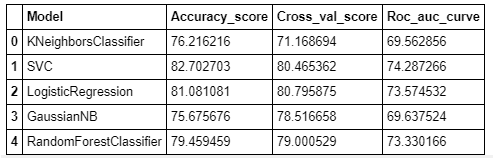












Now we can see that Accuracy\_score,Cross\_val\_score and Roc\_auc curve results and finalize best model for prediction target variable is LinearSVC .

>>from sklearn.externals import joblib

>>joblib.dump(SV,'Model\_Fraud\_Loan\_Prediction.obj')

5. **Concluding Remarks** :

The work has confirmed the efficiency of the support vector machine in a new context. A high accuracy is attained with a drastically reduced false positive and high true positives. The usage of technology to salvage loss in loan administration is displayed by this approach. In the future, testing the strength of an ensemble approach to find out the given result is to be pursued to see what it promises to deliver.

**1.  Problem Definition :**

Corona-viruses are a large family of viruses which may cause illness in animals or humans. In humans, several corona-viruses are known to cause respiratory infections ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS). The most recently discovered corona-virus causes corona-virus disease COVID-19 - [World Health Organization](https://www.who.int/news-room/q-a-detail/q-a-coronaviruses)

The number of new cases are increasing day by day around the world. This dataset has information from US states.

**2. Data Analysis :**

Data analysis is a process of inspecting, cleansing, transforming and modeling data with the goal of discovering useful information, informing conclusions and supporting decision-making.

The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.

2.1.**Used library Descriptions** :

#import libary for use method

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

2.2.**Variable Descriptions** :

| **Variable** | **Description** |
| --- | --- |
| Province\_State | The name of the State within the USA |
| Country\_Region | The name of the Country (US) |
| Last\_Update | The most recent date the file was pushed |
| Lat  Long\_ | Latitude |
| Longitude |
| Confirmed | Aggregated confirmed case count for the state |
| Deaths | Aggregated Death case count for the state |
| Recovered | Aggregated Recovered case count for the state |
| Active | Aggregated confirmed cases that have not been resolved (Active = Confirmed - Recovered - Deaths) |
| FIPS | Federal Information Processing Standards code that uniquely identifies counties within the USA |
| Incident\_Rate | confirmed cases per 100,000 persons |
| People\_Tested | Total number of people who have been tested |
| People\_Hospitalized | Total number of people hospitalized |
| Mortality\_Rate | Number recorded deaths \* 100/ Number confirmed cases |
| UID | Unique Identifier for each row entry |
| ISO3 | Officialy assigned country code identifiers |
| Testing\_Rate | Total number of people tested per 100,000 persons |
| Hospitalization\_Rate | Total number of people hospitalized \* 100/ Number of confirmed cases. |

2.3.**Data Descriptions** :

In Python we can know data type using pandas property **DataFrame.dtypes**

>>**df.dtypes**

Province\_State object

Country\_Region object

Last\_Update object

Lat float64

Long\_ float64

Confirmed int64

Deaths int64

Recovered float64

Active float64

FIPS int64

Incident\_Rate float64

People\_Tested float64

People\_Hospitalized float64

Mortality\_Rate float64

UID int64

ISO3 object

Testing\_Rate float64

Hospitalization\_Rate float64

dtype: object

2.4.**Data Information Descriptions** :

In Python we can know data information using pandas method DataFrame.info()

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

>>**df.info()**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 58 entries, 0 to 57

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Province\_State 58 non-null object

1 Country\_Region 58 non-null object

2 Last\_Update 58 non-null object

3 Lat 56 non-null float64

4 Long\_ 56 non-null float64

5 Confirmed 58 non-null int64

6 Deaths 58 non-null int64

7 Recovered 42 non-null float64

8 Active 58 non-null float64

9 FIPS 58 non-null int64

10 Incident\_Rate 56 non-null float64

11 People\_Tested 56 non-null float64

12 People\_Hospitalized 33 non-null float64

13 Mortality\_Rate 57 non-null float64

14 UID 58 non-null int64

15 ISO3 58 non-null object

16 Testing\_Rate 56 non-null float64

17 Hospitalization\_Rate 33 non-null float64

dtypes: float64(10), int64(4), object(4)

memory usage: 8.3+ KB

2.5.**Data shape Descriptions** :

In Python we can know data shape using pandas property DataFrame.shape

This return a tuple representing the dimensionality of the DataFrame.

>>**df.shape**

(58, 18)

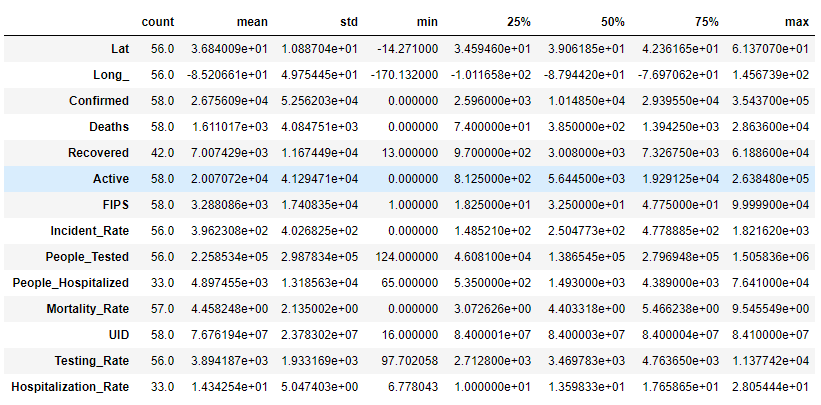
The given dataset shape is 58 row and 18 columns value (58 x 18).

2.6.**Data Statistical Descriptions** :

In Python we can know data Descriptive statistics using pandas property DataFrame.describe()

This provide Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.

>> **df.describe().T**

****

2.7.**Data Null Value Descriptions** :

In Python we can know data null values using pandas method DataFrame.isnull()

It's detect missing values for an array-like object.

**>>print(df.isnull().sum())**

Province\_State 0

Country\_Region 0

Last\_Update 0

Lat 2

Long\_ 2

Confirmed 0

Deaths 0

Recovered 16

Active 0

FIPS 0

Incident\_Rate 2

People\_Tested 2

People\_Hospitalized 25

Mortality\_Rate 1

UID 0

ISO3 0

Testing\_Rate 2

Hospitalization\_Rate 25

dtype: int64

2.8.**Data Imputation Descriptions** :

Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

>>mean\_Lat=df['Lat'].mean()

>>Lat=float(mean\_Lat)

>>df['Lat'].fillna(Lat,inplace=True)

>>mean\_Long=df['Long\_'].mean()

>>Long=float(mean\_Long)

>>df['Long\_'].fillna(Long,inplace=True)

>>mean\_Recovered=df['Recovered'].mean()

>>Recovered=float(mean\_Recovered)

>>df['Recovered'].fillna(Recovered,inplace=True)

>>mean\_Incident\_Rate=df['Incident\_Rate'].mean()

>>Incident\_Rate=float(mean\_Incident\_Rate)

>>df['Incident\_Rate'].fillna(Incident\_Rate,inplace=True)

>>mean\_People\_Tested=df['People\_Tested'].mean()

>>People\_Tested=float(mean\_People\_Tested)

>>df['People\_Tested'].fillna(People\_Tested,inplace=True)

>>mean\_People\_Hospitalized=df['People\_Hospitalized'].mean()

>>People\_Hospitalized=float(mean\_People\_Hospitalized)

>>df['People\_Hospitalized'].fillna(People\_Hospitalized,inplace=True)

>>mean\_Mortality\_Rate=df['Mortality\_Rate'].mean()

>>Mortality\_Rate=float(mean\_Mortality\_Rate)

>>df['Mortality\_Rate'].fillna(Mortality\_Rate,inplace=True)

>>mean\_Testing\_Rate=df['Testing\_Rate'].mean()

>>Testing\_Rate=float(mean\_Testing\_Rate)

>>df['Testing\_Rate'].fillna(Testing\_Rate,inplace=True)

>>mean\_Hospitalization\_Rate=df['Hospitalization\_Rate'].mean()

>>Hospitalization\_Rate=float(mean\_Hospitalization\_Rate)

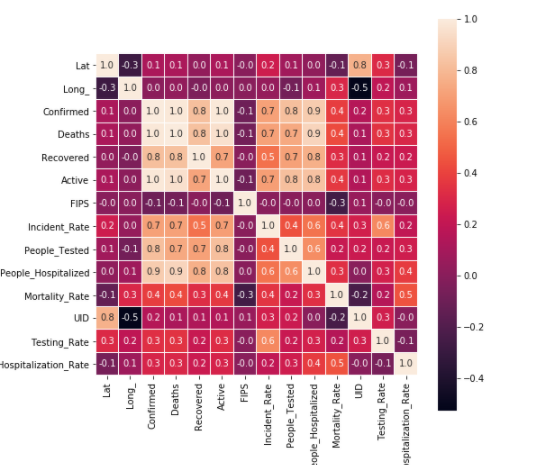
>>df['Hospitalization\_Rate'].fillna(Hospitalization\_Rate,inplace=True)

2.9.**Data Correlation Descriptions** :

Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related.

>>f,ax = plt.subplots(figsize = (8, 8))

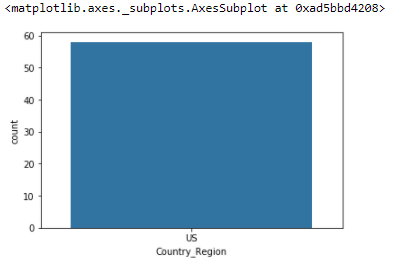
>>sns.heatmap(df.corr(), annot = True, linewidths = 0.1,fmt = '.1f', ax = ax, square = True)



2.10 .**Data visualization Descriptions** :

**Python** allows us to create **visualizations** easily and quickly using Matplotlib and Seaborn.

>>sns.countplot(x='Country\_Region', data=df)



>>r\_data = df.groupby(["Province\_State"])["Deaths", "Confirmed", "Recovered", "Active"].sum().reset\_index()

>>r\_data = r\_data.sort\_values(by='Deaths', ascending=False)

>>r\_data = r\_data[r\_data['Deaths']>50]

>>plt.figure(figsize=(12,6))

>>plt.xticks(rotation=90)

>>plt.plot(r\_data['Province\_State'],r\_data['Deaths'],color='red')

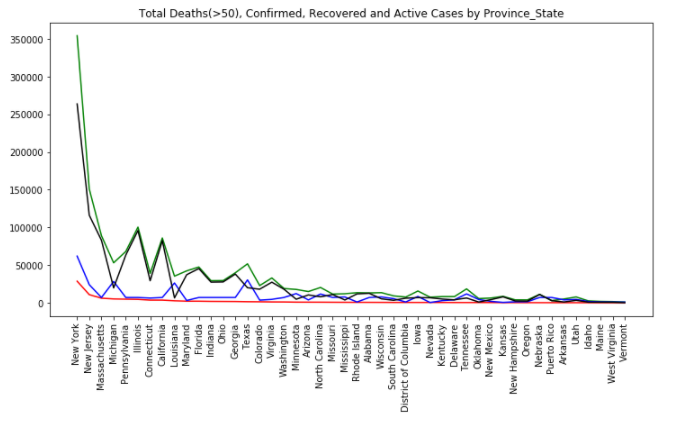
>>plt.plot(r\_data['Province\_State'],r\_data['Confirmed'],color='green')

>>plt.plot(r\_data['Province\_State'], r\_data['Recovered'], color ='blue')

>>plt.plot(r\_data['Province\_State'], r\_data['Active'], color ='black')

>>plt.title('Total Deaths(>50), Confirmed, Recovered and Active Cases by Province\_State')

>>plt.show()



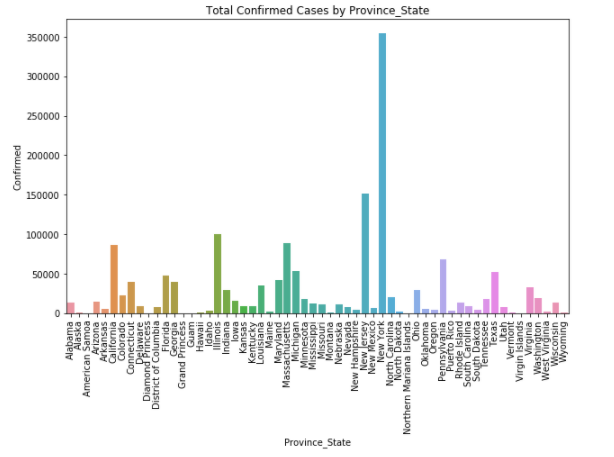
>>plt.figure(figsize=(10,6))

>>plt.xticks(rotation=90)

>>sns.barplot(x='Province\_State',y='Confirmed',data=df)

>>plt.title('Total Confirmed Cases by Province\_State')

>>plt.show()

****

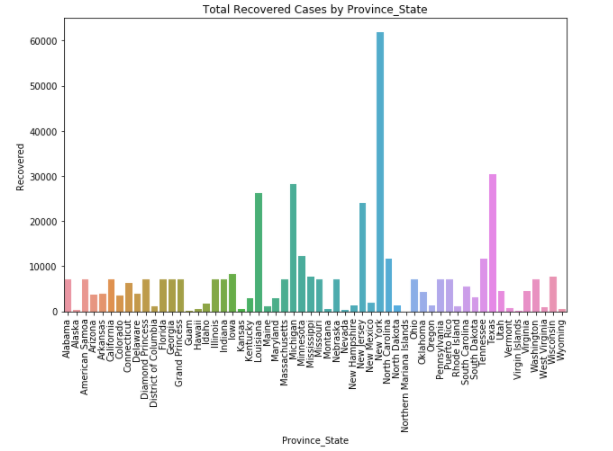
>>plt.figure(figsize=(10,6))

>>plt.xticks(rotation=90)

>>sns.barplot(x='Province\_State',y='Recovered',data=df)

>>plt.title('Total Recovered Cases by Province\_State')

>>plt.show()



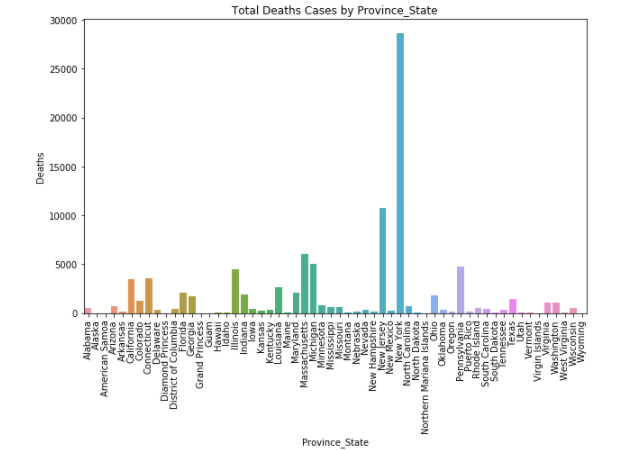
>>plt.figure(figsize=(10,6))

>>plt.xticks(rotation=90)

>>sns.barplot(x='Province\_State',y='Deaths',data=df)

>>plt.title('Total Deaths Cases by Province\_State')

>>plt.show()



3. **Data Pre-processing Descriptions** :

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format. In this data set we LabelEncoder, OneHotEncoder for data pre processing .

>>from sklearn.preprocessing import LabelEncoder, OneHotEncoder

>>df\_new = pd.get\_dummies(df.drop(columns = [ 'Last\_Update' , 'Lat' , 'Long\_','Country\_Region'], axis = 1))

Now we can separate input variables and target variable.

>>y = df\_new['Deaths']

>>x = df\_new.drop('Deaths', axis = 1)

>>print(x.shape)

>>print(y.shape)

**(58, 75)**

**(58,)**

4. **Building Machine Learning Models** :

# we tried all the model and till now ada boost regression is the best

>>from sklearn.model\_selection import train\_test\_split

>>from sklearn.ensemble import AdaBoostRegressor

>>from sklearn.tree import DecisionTreeRegressor

>>from sklearn.metrics import mean\_absolute\_error

>>from sklearn.metrics import mean\_squared\_error

>>from sklearn.metrics import r2\_score

>>x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,random\_state=73,test\_size=0.30)

>>ada\_reg=AdaBoostRegressor()

>>parameters={'learning\_rate':[0.001,0.01,0.1,1],'n\_estimators':[10,100,500,1000],'base\_estimator':[lreg,lsreg,DecisionTreeRegressor()]}

>>clf=GridSearchCV(ada\_reg,parameters,cv=5)

>>clf.fit(x,y)

>>clf.best\_params\_

**{'base\_estimator': Lasso(alpha=0.1, copy\_X=True, fit\_intercept=True, max\_iter=1000,**

**normalize=False, positive=False, precompute=False, random\_state=None,**

**selection='cyclic', tol=0.0001, warm\_start=False),**

**'learning\_rate': 0.001,**

**'n\_estimators': 10}**

>>ada\_reg=AdaBoostRegressor(base\_estimator=lsreg,learning\_rate=0.01,n\_estimators=100)

>>ada\_reg.fit(x\_train,y\_train)

>>y\_pred=ada\_reg.predict(x\_test)

#lets find the rmse and r2\_score using sklearn.metrics

>>import numpy as np

>>from sklearn.metrics import r2\_score

>>from sklearn.metrics import mean\_squared\_error

>>print("RMSE is : ",np.sqrt(mean\_squared\_error(y\_test,y\_pred)))

>>print('r2\_score is : ',r2\_score(y\_test,y\_pred))

**RMSE is : 569.2438730591932**

**r2\_score is : 0.9584006116717257**

Now we can see that RMSE and r2\_score results and finalize best model for prediction target variable is ada boost regression .

>>from sklearn.externals import joblib

>>joblib.dump(ada\_reg,'covid\_Deaths\_model.pkl')

**['covid\_Deaths\_model.pkl']**

5. **Concluding Remarks** :