```
import pandas as pd
```

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

```
data = pd.read_csv("/content/ML Case Study - Data.csv")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.options.display.float_format = '{:,.2f}'.format
from IPython.core.display import display, HTML
display(HTML("<style>.container {width : 98% !important; }</style>"))
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = [10,10]
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
```

```
from scipy import stats
from sklearn import metrics
```

```
import yellowbrick as yb
```

Checking Data Types

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  5000 non-null   int64
 1   Age                 5000 non-null   int64
```

Statistical Summary

```
 6   CCAvg               5000 non-null   float64
```

```
1 data.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mo
count	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5
mean	2,500.50	45.34	20.10	73.77	93,152.50	2.40	1.94	1.88	
std	1,443.52	11.46	11.47	46.03	2,121.85	1.15	1.75	0.84	
min	1.00	23.00	-3.00	8.00	9,307.00	1.00	0.00	1.00	
25%	1,250.75	35.00	10.00	39.00	91,911.00	1.00	0.70	1.00	
50%	2,500.50	45.00	20.00	64.00	93,437.00	2.00	1.50	2.00	
75%	3,750.25	55.00	30.00	98.00	94,608.00	3.00	2.50	3.00	

```
1 data.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mo
count	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5
mean	2,500.50	45.34	20.10	73.77	93,152.50	2.40	1.94	1.88	
std	1,443.52	11.46	11.47	46.03	2,121.85	1.15	1.75	0.84	
min	1.00	23.00	-3.00	8.00	9,307.00	1.00	0.00	1.00	
25%	1,250.75	35.00	10.00	39.00	91,911.00	1.00	0.70	1.00	
50%	2,500.50	45.00	20.00	64.00	93,437.00	2.00	1.50	2.00	
75%	3,750.25	55.00	30.00	98.00	94,608.00	3.00	2.50	3.00	

```
1 # from pandas_profiling import ProfileReport
2 # prof = ProfileReport(data)
3 # prof.to_file("profile.html")
```

Checking for Null Values

There were none

```
data.isnull().sum()
```

```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

Checking For Incorrect Values

```
data.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mo
count	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5
mean	2,500.50	45.34	20.10	73.77	93,152.50	2.40	1.94	1.88	
std	1,443.52	11.46	11.47	46.03	2,121.85	1.15	1.75	0.84	
min	1.00	23.00	-3.00	8.00	9,307.00	1.00	0.00	1.00	
25%	1,250.75	35.00	10.00	39.00	91,911.00	1.00	0.70	1.00	
50%	2,500.50	45.00	20.00	64.00	93,437.00	2.00	1.50	2.00	
75%	3,750.25	55.00	30.00	98.00	94,608.00	3.00	2.50	3.00	

Incorrect values are in experience as it connot be negative

```
for i,k in enumerate(data["Experience"]):
    if k < 0:
```

```
        data["Experience"][i] = 0
```

```
data.columns
```

```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

Corrected the Experience Values

```
data.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mo
count	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5,000.00	5
mean	2,500.50	45.34	20.12	73.77	93,152.50	2.40	1.94	1.88	
std	1,443.52	11.46	11.44	46.03	2,121.85	1.15	1.75	0.84	
min	1.00	23.00	0.00	8.00	9,307.00	1.00	0.00	1.00	
25%	1,250.75	35.00	10.00	39.00	91,911.00	1.00	0.70	1.00	
50%	2,500.50	45.00	20.00	64.00	93,437.00	2.00	1.50	2.00	
75%	3,750.25	55.00	30.00	98.00	94,608.00	3.00	2.50	3.00	

Unique value in each column

```
for i in list( data.columns):
    print(i,data[i].nunique())
```

```
ID 5000
```

```
Income 162
```

Number Of People with 0 Mortgage

```
Mortgage 347
```

```
(data["Mortgage"] == 0).sum()
```

```
3462
```

```
CreditCard 2
```

Number of people 0 credit card spending

```
(data["CCAvg"] == 0).sum()
```

```
106
```

```
data.drop("ID", axis = 1, inplace= True)
```

```
data.drop("ZIP Code", axis = 1, inplace= True)
```

Value Counts of diffrent categorical variable

```
for i in list(  ['Family',
       'Education',  'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard']):
    print(data[i].value_counts())
    print()
```

```
1    1472
2    1296
4    1222
3    1010
Name: Family, dtype: int64

1    2096
3    1501
2    1403
Name: Education, dtype: int64

0    4520
1     480
Name: Personal Loan, dtype: int64

0    4478
```

```
pd.pivot_table(data , values="Personal Loan", index = "Family",columns="Education", aggfun
```

Education	1	2	3
Family			
1	9	40	58
2	4	50	52
3	40	44	49
4	40	48	46

Can be seen than that as the family and and education grows the number of people that go for personal loans have increased

Data Preproceesing

```
x = data.loc[:,['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
        'Mortgage', 'Securities Account', 'CD Account',
        'Online', 'CreditCard']]
y = data["Personal Loan"]
print(x.shape,y.shape)
```

```
(5000, 11) (5000,)
```

```
dic = {0:0,1:1}
for k in y:
    dic[k] +=1
# temp = dic[0]
# dic[0] = dic[1]
```

```
# dic[1] = temp
```

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train, Y_test = train_test_split(x,y,test_size = 0.3,random_state=7)
```

First Model Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lg_reg = LogisticRegression(max_iter =100000,class_weight=dic)
lg_reg.fit(X_train, Y_train)
print("test_accuracy",lg_reg.score(X_test,Y_test))
print("train_accuracy",lg_reg.score(X_train,Y_train))
```

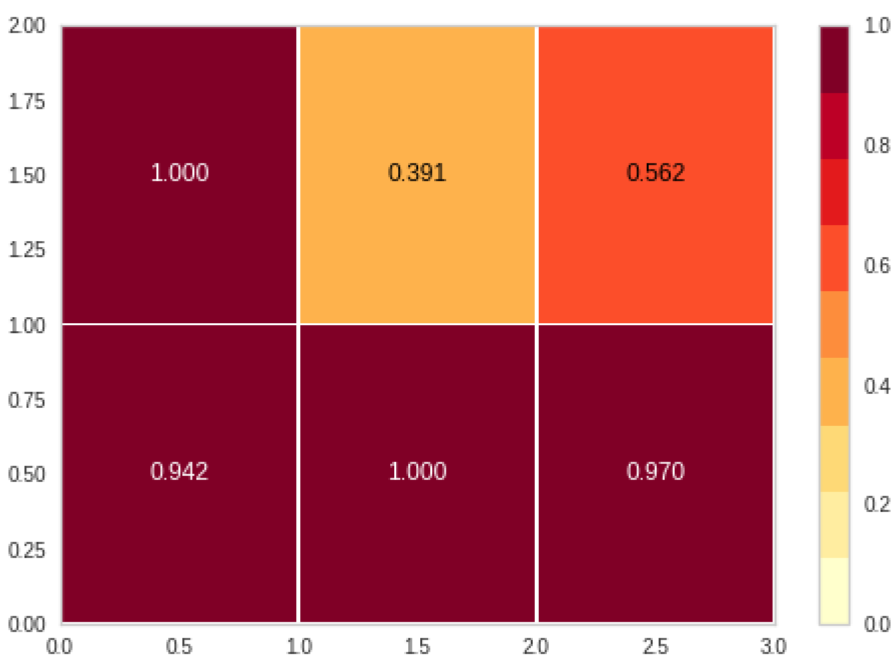
```
test_accuracy 0.944
train_accuracy 0.9382857142857143
```

```
y_pred = lg_reg.predict(X_test)
```

```
from yellowbrick.classifier import ClassificationReport
viz = ClassificationReport(lg_reg)
viz.fit(X_train, Y_train)
viz.score(X_test, Y_test)

```

```
0.944
```




```
lg_reg.get_params()
```

```
{'C': 1.0,
 'class_weight': {0: 4520, 1: 481},
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100000,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,y_pred)
```

```
array([[1362,    0],
       [  84,   54]])
```

Performance Metrics

We choose ***f1_score*** as the evaluation matric due to the class size imbalance. 1 are 481 and 0 are 4520.

Metrics from http://onlineconfusionmatrix.com/

Measure	Value
Sensitivity	0.8365
Specificity	0.9635
Precision	0.6304
Negative Predictive Value	0.9875
False Positive Rate	0.0365
False Discovery Rate	0.3696
False Negative Rate	0.1635
Accuracy	0.9547
F1 Score	0.7190
Matthews Correlation Coefficient	0.7031

Tweaking Hyperparameters

Since the most there is a imbalance in the number of dataset of each class.

```
from sklearn.metrics import f1_score
Lr_rate = [0.001,0.01,0.1,0.3,0.9,1.0,3,10.0]
for i in Lr_rate:
    lg_reg = LogisticRegression(C = i,max_iter =1000,class_weight="balanced")
    lg_reg.fit(X_train, Y_train)
    Y_pred = lg_reg.predict(X_test)
    print(f1_score(Y_test,Y_pred),"C =" + str(i))
```

```
0.5617977528089888 C =0.001
0.5794392523364487 C =0.01
0.5975903614457831 C =0.1
0.5980861244019139 C =0.3
0.6024096385542169 C =0.9
0.6038647342995169 C =1.0
0.6038647342995169 C =3
0.6024096385542169 C =10.0
```

Ways To Improving our model

Since there are 5 categorical variables in the data using **Random Forest** and **descision trees** classifiers make more sense. But we will work our way upto it.

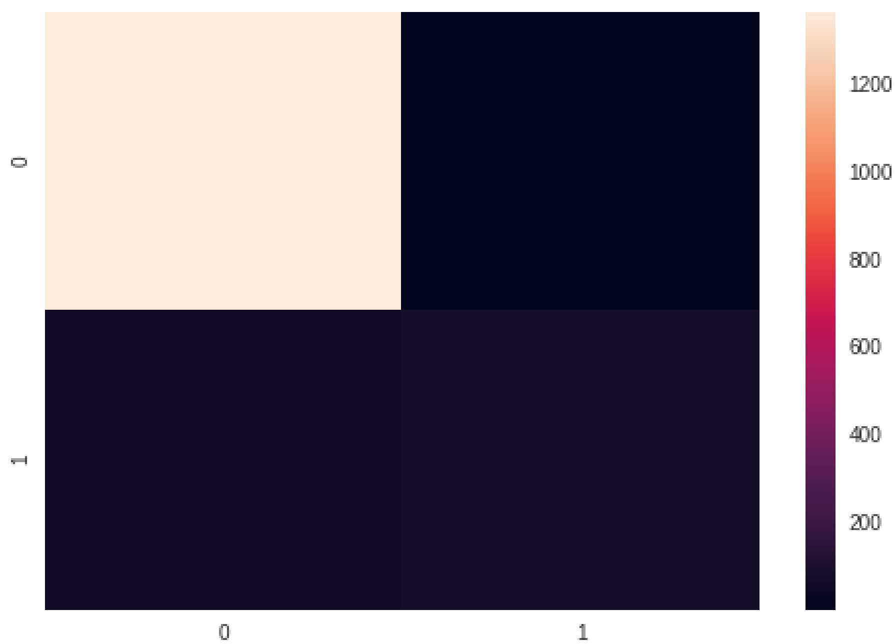
SVM Based Classifier

```
from sklearn.svm import SVC
Spvm = SVC(kernel = "poly",class_weight=dic)
Spvm.fit(X_train,Y_train)
Spvm.score(X_test,Y_test)
```

```
0.9566666666666667
```

```
Y_pred = Spvm.predict(X_test)
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(Y_test,Y_pred))
f1_score(Y_test,Y_pred)
```

```
0.6948356807511737
```



Decision Tree Classifier

As predicted we get a lot better prediction scores.

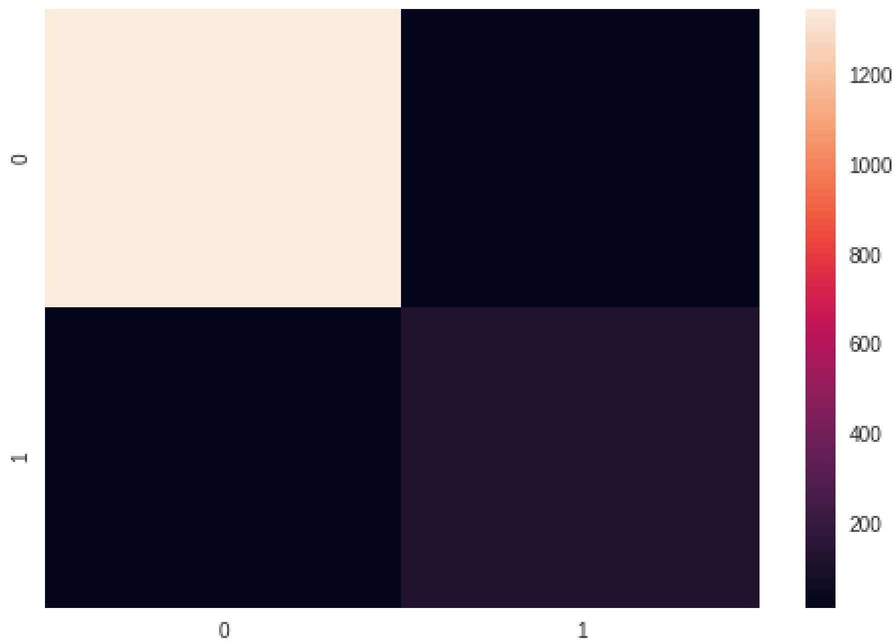
```
from sklearn.tree import DecisionTreeClassifier
Dt = DecisionTreeClassifier()
Dt.fit(X_train,Y_train)
```

```
Dt.fit(X_train,Y_train)
print(Dt.score(X_test,Y_test))
Y_pred = Dt.predict(X_test)
f1_score(Y_test,Y_pred)
```

```
0.9833333333333333
0.9110320284697508
```

```
Y_pred = Dt.predict(X_test)
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test,Y_pred))
sns.heatmap(confusion_matrix(Y_test,Y_pred))
```

```
[[1347   15]
 [  10  128]]
<matplotlib.axes._subplots.AxesSubplot at 0x7f3253c10438>
```



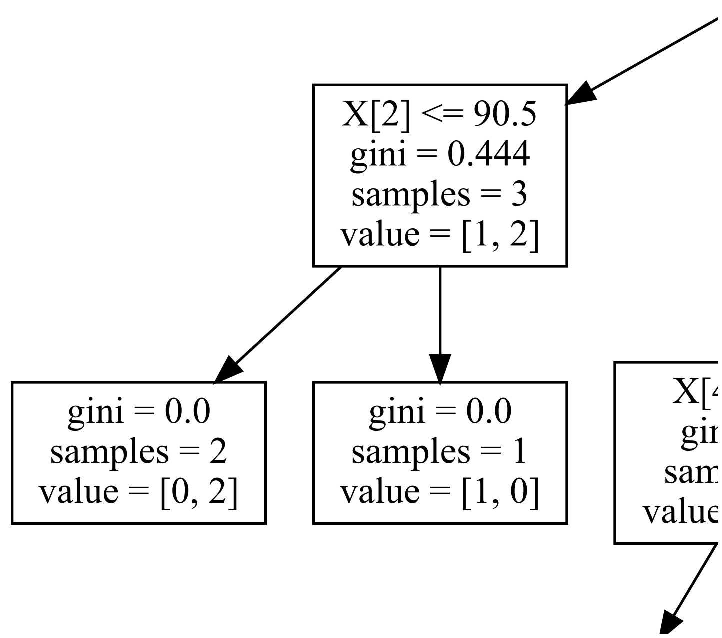

```
from sklearn import tree
import graphviz
dt_data = tree.export_graphviz(Dt, out_file=None)
graph = graphviz.Source(dt_data)
graph.render("iris")
tree.export_graphviz(Dt, out_file=None,
                     feature_names=x.columns,
                      class_names = "Personal Loan",                        filled=True, rou
                        special_characters=True)
```

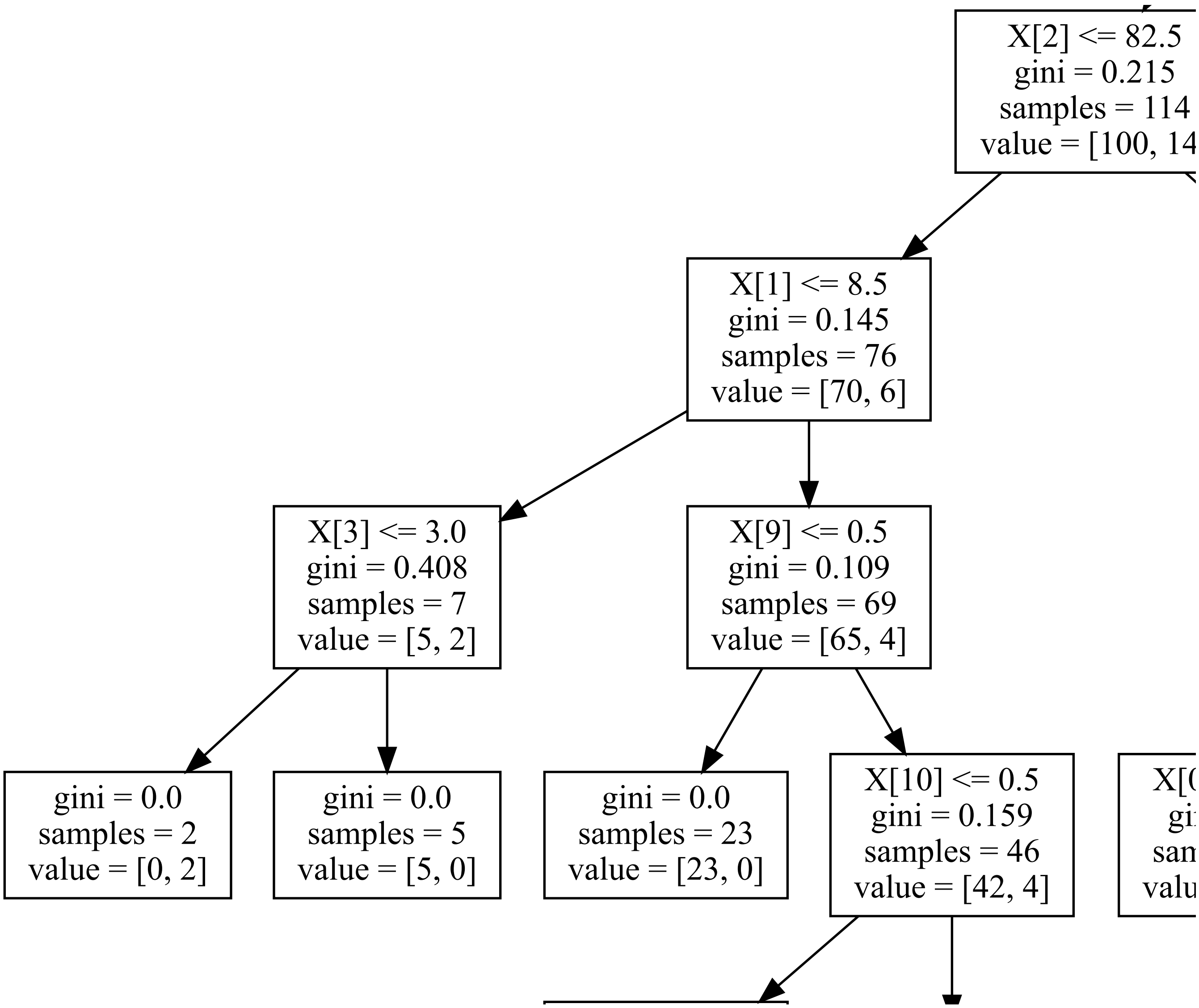

```
'digraph Tree {\nnode [shape=box, style="filled, rounded", color="black", fontname=helve
[fontname=helvetica] ;\n0 [label=<Income &le; 100.5<br/>gini = 0.176<br/>samples = 3500<
158, 342]<br/>class = P>, fillcolor="#e88f4e"] ;\n1 [label=<CCAvg &le; 2.95<br/>gini = 0
es = 2638<br/>value = [2607, 31]<br/>class = P>, fillcolor="#e5823b"] ;\n0 -> 1 [labeldi
belangle=45, headlabel="True"] ;\n2 [label=<gini = 0.0<br/>samples = 2472<br/>value = [2
ass = P>, fillcolor="#e58139"] ;\n1 -> 2 ;\n3 [label=<CD Account &le; 0.5<br/>gini = 0.3
= 166<br/>value = [135, 31]<br/>class = P>, fillcolor="#eb9e66"] ;\n1 -> 3 ;\n4 [label=<
```

```
x.columns
```

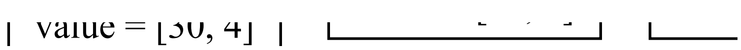
```
Index(['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
       'Mortgage', 'Securities Account', 'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

```
graph = graphviz.Source(dt_data)
graph
```


X[2] <= 90.5
gini = 0.444
samples = 3
value = [1, 2]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]



RandomForestClassifier




```
from sklearn.ensemble import RandomForestClassifier
n_est = [25,50,75,100,125]
max_depth = [None,3,4,5,6]
acc = []
for i in n_est:
    for k in max_depth:
        rbf_clf = RandomForestClassifier(n_estimators=i,max_depth=k)
        rbf_clf.fit(X_train, Y_train)
        Y_pred = rbf_clf.predict(X_test)
        print(rbf_clf.score(X_test, Y_test),f1_score(Y_test,Y_pred), "N_est = " +str(i),"m
```

```
0.9886666666666667 0.9368029739776952 N_est = 25 max_depth = None
0.9606666666666667 0.730593607305936 N_est = 25 max_depth = 3
0.9793333333333333 0.8734693877551021 N_est = 25 max_depth = 4
0.984 0.9069767441860466 N_est = 25 max_depth = 5
0.982 0.8957528957528957 N_est = 25 max_depth = 6
0.9886666666666667 0.9368029739776952 N_est = 50 max_depth = None
0.9573333333333334 0.7009345794392524 N_est = 50 max_depth = 3
0.972 0.8220338983050847 N_est = 50 max_depth = 4
0.9833333333333333 0.9034749034749036 N_est = 50 max_depth = 5
0.9853333333333333 0.9166666666666666 N_est = 50 max_depth = 6
0.986 0.9207547169811321 N_est = 75 max_depth = None
0.9433333333333334 0.5595854922279793 N_est = 75 max_depth = 3
0.964 0.7589285714285714 N_est = 75 max_depth = 4
0.9793333333333333 0.8774703557312253 N_est = 75 max_depth = 5
0.9846666666666667 0.9118773946360155 N_est = 75 max_depth = 6
0.9866666666666667 0.9259259259259259 N_est = 100 max_depth = None
0.944 0.5670103092783505 N_est = 100 max_depth = 3
0.9733333333333334 0.8347107438016529 N_est = 100 max_depth = 4
0.9846666666666667 0.9118773946360155 N_est = 100 max_depth = 5
```

```
rbf_clf = RandomForestClassifier(n_estimators=100,max_depth = None)
rbf_clf.fit(X_train, Y_train)
Y_pred = rbf_clf.predict(X_test)
print(rbf_clf.score(X_test, Y_test),f1_score(Y_test,Y_pred))
```

```
0.9906666666666667 0.9477611940298507
```

Best F1 Score 94 % Accuracy 99

Since the accuracy varies a lot so let us boost it to its maximum accuracy.(I'll use XGBoost)

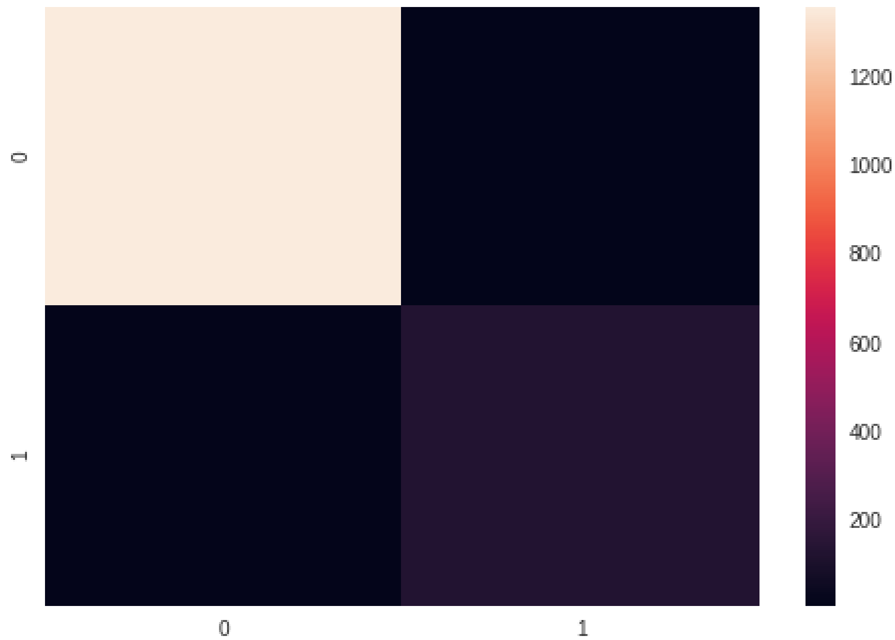
```
import pickle
pickle.dump(rbf_clf,open("rbf_clf.pkl","wb"))
```

```
rbf_clf.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
```

```
Y_pred = rbf_clf.predict(X_test)
print(confusion_matrix(Y_test,Y_pred))
sns.heatmap(confusion_matrix(Y_test,Y_pred))
f1_score(Y_test,Y_pred)
```

```
[[1359    3]
 [  11  127]]
0.9477611940298507
```



Measure	Value

```
X_train.columns
```

```
Index(['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
       'Mortgage', 'Securities Account', 'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

Negative Predictive Value	0.9920

```
# data.groupby(['Loan_Status', 'Gender', 'Property_Area'])['Married'].value_counts()
# from yellowbrick.classifier import ClassificationReport
# viz = ClassificationReport(LR_model)
# viz.fit(XT, YT)
# viz.score(Xt, Yt)
# viz.show()
# import statsmodels.api as sm
# X2 = sm.add_constant(XT)
# regressor2 = sm.Logit(YT, X2).fit()
# print("p-Values for each column: ")
# print()
# print(regressor2.pvalues)
```

```
from xgboost import XGBClassifier
model = XGBClassifier(max_depth=3,n_estimators=100)
model.fit(X_train, Y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
print(model.score(X_test,Y_test),model.score(X_train,Y_train))
```

```
0.9893333333333333 0.9908571428571429
```

```
Y_pred = model.predict(X_test)
```

```
print(confusion_matrix(Y_test,Y_pred))
sns.heatmap(confusion_matrix(Y_test,Y_pred))
f1_score(Y_test,Y_pred)
```