





```
In [18]: Rural
Out[18]: 1128.0,
114.0,
133.0,
112.0,
106.0,
320.0,
135.0,
165.0,
nan,
120.0,
152.0,
185.0,
175.0,
180.0,
195.0,
93.0,
370.0,
nan,
182.0,
84.0,
129.0,
185.0,
225.0,
118.0,
152.0,
113.0,
160.0,
120.0,
158.0,
182.0,
112.0,
139.0,
95.0,
81.0,
116.0,
101.0,
67.0,
66.0,
58.0,
168.0,
188.0,
48.0,
120.0,
187.0,
59.0,
135.0,
170.0,
200.0,
42.0,
120.0,
108.0,
104.0,
255.0,
115.0,
304.0,
120.0,
151.0,
135.0,
126.0,
124.0,
nan,
89.0,
120.0,
187.0,
139.0,
151.0,
132.0,
144.0,
135.0,
138.0,
120.0,
173.0,
300.0,
176.0,
160.0,
174.0,
120.0,
46.0,
152.0,
187.0,
308.0,
95.0,
105.0,
145.0,
141.0,
133.0,
124.0,
130.0,
126.0,
109.0,
65.0,
194.0,
66.0,
360.0,
218.0,
110.0,
112.0,
138.0,
121.0,
81.0,
138.0,
180.0,
175.0,
130.0,
110.0,
55.0,
150.0,
125.0,
149.0,
90.0,
136.0,
153.0,
140.0,
113.0,
162.0,
150.0,
154.0,
113.0,
131.0,
80.0,
119.0,
107.0,
209.0,
113.0,
142.0,
243.0,
100.0,
311.0,
150.0,
95.0,
100.0,
490.0,
400.0,
216.0,
110.0,
126.0,
66.0,
157.0,
159.0,
145.0,
110.0,
nan,
132.0,
211.0,
182.0,
123.0,
107.0,
61.0,
146.0,
172.0,
142.0,
110.0,
187.0,
180.0,
175.0,
172.0,
157.0,
108.0,
71.0,
40.0]
```

```
In [19]: SemiUrban
Out[19]: 1158.0,
149.0,
151.0,
151.0,
152.0,
114.0,
141.0,
110.0,
134.0,
144.0,
100.0,
120.0,
91.0,
112.0,
138.0,
136.0,
97.0,
81.0,
187.0,
308.0,
95.0,
105.0,
145.0,
141.0,
133.0,
124.0,
130.0,
126.0,
109.0,
65.0,
194.0,
66.0,
360.0,
218.0,
110.0,
112.0,
138.0,
121.0,
81.0,
138.0,
180.0,
175.0,
130.0,
110.0,
55.0,
150.0,
125.0,
149.0,
90.0,
136.0,
153.0,
140.0,
113.0,
162.0,
150.0,
154.0,
113.0,
131.0,
80.0,
119.0,
107.0,
209.0,
113.0,
142.0,
243.0,
100.0,
311.0,
150.0,
95.0,
100.0,
490.0,
400.0,
216.0,
110.0,
126.0,
66.0,
157.0,
159.0,
145.0,
110.0,
nan,
132.0,
211.0,
182.0,
123.0,
107.0,
61.0,
146.0,
172.0,
142.0,
110.0,
187.0,
180.0,
175.0,
172.0,
157.0,
108.0,
71.0,
40.0]
```

```
In [20]: Urban = [i for i in Urban if pd.notnull(i)]
SemiUrban = [i for i in SemiUrban if pd.notnull(i)]
Rural = [i for i in Rural if pd.notnull(i)]

#extract median values to impute in the data set
Median_Ur = statistics.median(Urban)
Median_Ru = statistics.median(Rural)
Median_SuR = statistics.median(SemiUrban)
```

```
#filling the median values
def fillloanamount(cols):
    pa = cols[0]
    la = cols[1]

    if pd.isnull(la):
        if pa == "Urban":
            return Median_Ur
        elif pa == "SemiUrban":
            return Median_SuR
        else:
            return Median_Ru
    else:
        return la

data["LoanAmount"] = data[["Property_Area", "LoanAmount"]].apply(fillloanamount, axis= 1)
```

```
In [21]: #Analysing and imputing missing values of Loan_Amount_Term column
sns.jointplot(x= data["LoanAmount"], y= data["Loan_Amount_Term"])
data[["Loan_Amount_Term"]].fillna(data["Loan_Amount_Term"].mode()[0], inplace = True)
```

```
In [22]: #Analysing and imputing missing values of Credit_History
sns.pairplot(data)
data[["Credit_History"]].fillna( data["Credit_History"].mode()[0], inplace = True)
```

```
In [23]: #checking the Nan Values
cParams['figure.figsize'] = 10,2
sns.setmap(data.isnull(), yticklabels=False, char=False, cmap='viridis')
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x2d82f2f9188>
```

```
In [24]: x = data.iloc[:,0:11]
y = data.iloc[:,1]

#checking the if the dataset is balanced or imbalanced
sns.countplot(y)

#this is an imbalanced data set of yes:no = 1:0.5 , thus no need to use any sampling techniques
#ROS = RandomOverSampler()
#x_ros,y_ros = ROS.fit_sample(x,y)

#the data set is balanced now
#sns.countplot(y_ros)

x.info()
#dummy coding and scaling
x_dummy = x.select_dtypes(exclude="number")
x_dummy

columns_to_drop = x_dummy.columns
x_scaling = x.drop(columns_to_drop, axis = 1)
x_scaling

columns_scaling = x_scaling.columns
loan_id = pd.DataFrame(x_dummy.iloc[:,0])
x_dummy = pd.get_dummies(x_dummy.iloc[:,1:], drop_first = True)

X = pd.concat([x_scaling, x_dummy], axis = 1)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2, random_state=0)

scaling = StandardScaler()
x_train = scaling.fit_transform(x_train)
x_test = scaling.transform(x_test)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 11 columns):
#    Column                Non-Null Count  Dtype
---  -
0    Loan_ID                614 non-null   object
1    Gender                 614 non-null   object
2    Married                614 non-null   object
3    Dependents             614 non-null   object
4    Education              614 non-null   object
5    Self_Employed          614 non-null   object
6    ApplicantIncome        614 non-null   int64
7    CoapplicantIncome      614 non-null   float64
8    LoanAmount             614 non-null   float64
9    Loan_Amount_Term       614 non-null   float64
10   Credit_History          614 non-null   float64
dtypes: float64(4), int64(1), object(6)
memory usage: 52.9+ KB
```

```
In [25]: #1st model
#Logistic Regression
LR = LogisticRegression()
LR.fit(x_train,y_train)
lr_params = {'solver': ['newton-og', 'lbfgs', 'sag'], 'penalty': ['l1', 'l2'], 'C': [100, 10, 1.0, 0.1, 0.01], 'fit_intercept': ['True', False], 'l1_ratio': np.linspace(0,1,10)},
{'solver': ['saga'], 'penalty': ['elasticnet'], 'C': [100, 10, 1.0, 0.1, 0.01], 'fit_intercept': ['True', False], 'l1_ratio': np.linspace(0,1,10)},
{'solver': ['liblinear'], 'penalty': ['l1', 'l2'], 'C': [100, 10, 1.0, 0.1, 0.01], 'fit_intercept': ['True', False], 'intercept_scaling': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}

rndcv_LR = RandomizedSearchCV(estimator = LR, param_distributions = lr_params, cv = 10, n_iter = 40, scoring= 'accuracy')
rndcv_LR.fit(x_train,y_train)
print(rndcv_LR.best_params_)
LR_mean_accuracy = rndcv_LR.best_score_
print(LR_mean_accuracy)

('solver': 'liblinear', 'penalty': 'l1', 'intercept_scaling': 0.9, 'fit_intercept': False, 'C': 10)
0.804489795918366
```

```
In [26]: LR = LogisticRegression (solver = "liblinear",penalty = "l1",intercept_scaling = 0.9, fit_intercept = F)
LR.fit(x_train,y_train)
y_pred = LR.predict(x_test)
LR_accuracy = accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))
plot_confusion_matrix(LR,x_test, y_test)
```

```
precision    recall  f1-score   support

N           0.88      0.98      0.60        33
Y           0.83      0.45      0.90        90

accuracy         0.86      0.72      0.75      123
macro avg       0.85      0.72      0.68      123
weighted avg    0.84      0.84      0.82      123
```

```
Out[28]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d82f2c7b88>
```

```
In [36]: #2nd model
#KNN
KNN = KNeighborsClassifier()

knn_params = [{'algorithm': ['auto'], 'n_neighbors':np.arange(start=1,stop=100,step=1), "weights":["uniform", "distance"]},
               {'algorithm': ['ball_tree'], 'n_neighbors':np.arange(start=1,stop=100,step=1), "weights":["uniform", "distance"], "leaf_size":np.arange(start=1,stop=100,step=1)},
               {'algorithm': ['kd_tree'], 'n_neighbors':np.arange(start=1,stop=100,step=1), "weights":["uniform", "distance"], "leaf_size":np.arange(start=1,stop=100,step=1)},
               {'algorithm': ['brute'], 'n_neighbors':np.arange(start=1,stop=100,step=1), "weights":["uniform", "distance"]}]

rndcv_KNN = RandomizedSearchCV(estimator = KNN, param_distributions = knn_params, cv = 10, n_iter = 40, scoring= 'accuracy')
rndcv_KNN.fit(x_train,y_train)
print(rndcv_KNN.best_params_)
KNN_mean_accuracy = rndcv_KNN.best_score_
print(KNN_mean_accuracy)

('weights': 'uniform', 'n_neighbors': 13, 'leaf_size': 98, 'algorithm': 'ball_tree')
0.804489795918366
```

```
In [37]: KNN = KNeighborsClassifier(weights = "uniform",n_neighbors= 13,leaf_size= 98,algorithm= "ball_tree")
KNN.fit(x_train,y_train)
y_pred = KNN.predict(x_test)
KNN_accuracy = accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))
plot_confusion_matrix(KNN,x_test, y_test)
```

```
precision    recall  f1-score   support

N           0.68      0.39      0.57        33
Y           0.81      0.93      0.80        90

accuracy         0.75      0.66      0.68      123
macro avg       0.77      0.66      0.68      123
weighted avg    0.77      0.79      0.77      123
```

```
Out[37]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d82f397248>
```

```
In [38]: #3rd model
#RandomForest
randomforest = [{"n_estimators": [10,20,30,40,50,60,70,80,90,100,150,200,300,500,1000], "max_depth": ra
randint(1,5), "min_samples_split": [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9], "min_samples_leaf": randint(1,5),
"max_features": ["sqrt", "log", "auto", "best"], "min_weight_fraction_leaf": [0.0,0.1,0.2,0.3,0.4,0.5], "max_features":
randint(1,12), "bootstrap":["True, False]}

RF = RandomForestClassifier()

rndcv_RF = RandomizedSearchCV(estimator = RF, param_distributions = randomforest, cv= 10, n_iter = 40 , scoring= 'accuracy')
rndcv_RF.fit(x_train,y_train)
print(rndcv_RF.best_params_)
RF_mean_accuracy = rndcv_RF.best_score_
print(RF_mean_accuracy)

('bootstrap': False, 'criterion': 'gini', 'max_depth': 4, 'max_features': 10, 'min_samples_leaf': 2, 'min_samples_split': 0.6, 'min_weight_fraction_leaf': 0.1, 'n_estimators': 90)
0.804489795918366
```

```
In [39]: RF = RandomForestClassifier(bootstrap= False, criterion= 'gini', max_depth= 4, max_features= 10, min_sa
RF.fit(x_train,y_train)
y_pred = RF.predict(x_test)
RF_accuracy = accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))
plot_confusion_matrix(RF,x_test, y_test)
```

```
precision    recall  f1-score   support

N           0.88      0.42      0.57        33
Y           0.82      0.98      0.89        90

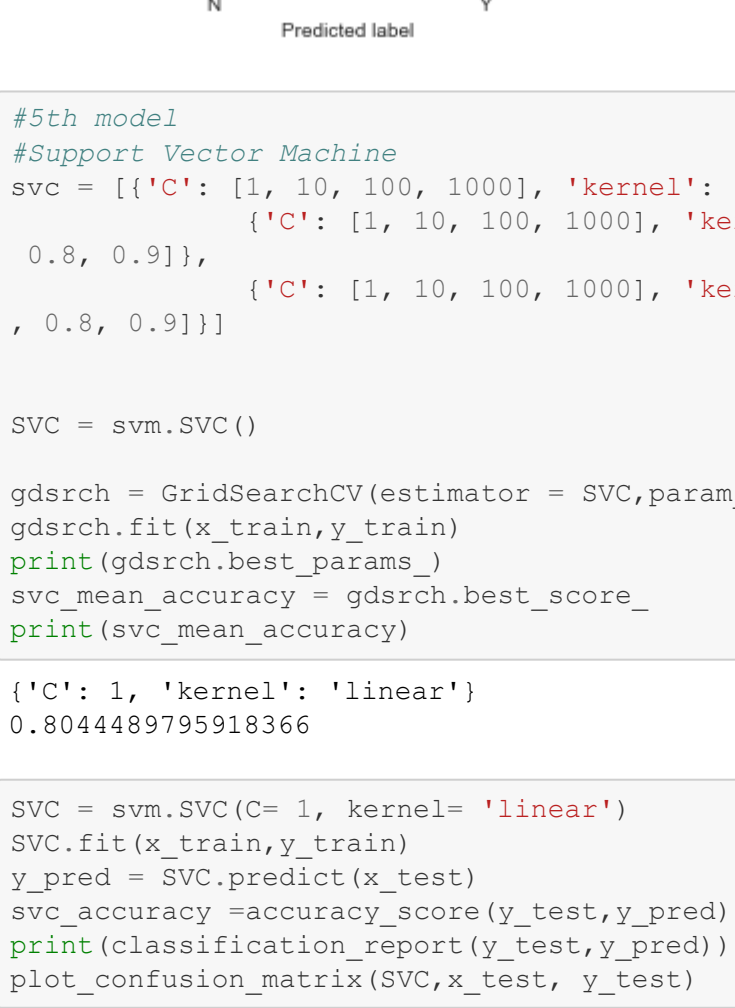
accuracy         0.85      0.70      0.83      123
macro avg       0.84      0.70      0.73      123
weighted avg    0.84      0.83      0.81      123
```

```
Out[39]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d82f3b9c48>
```

In [40]: `#4th model`  
`#Naive Bayes`  
`NB = GaussianNB()`  
`NB.fit(x_train,y_train)`  
`y_pred = NB.predict(x_test)`  
`NB_accuracy = accuracy_score(y_test,y_pred)`  
`print(classification_report(y_test,y_pred))`  
`plot_confusion_matrix(NB,x_test, y_test)`

	precision	recall	f1-score	support
N	0.83	0.45	0.59	33
Y	0.83	0.97	0.89	96
accuracy			0.83	123
macro avg	0.83	0.71	0.74	123
weighted avg	0.83	0.83	0.81	123

Out[40]: `<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d82f373648>`



In [41]: `#5th model`  
`#Support Vector Machine`  
`svc = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']}], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]],`  
`['C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]]`  
`SVC = svm.SVC()`  
`gdsearch = GridSearchCV(estimator = SVC,param_grid = svc,scoring = 'accuracy',cv = 10)`  
`gdsearch.fit(x_train,y_train)`  
`print(gdsearch.best_params_)`  
`svc_mean_accuracy = gdsearch.best_score_`  
`print(svc_mean_accuracy)`  
`{'C': 1, 'kernel': 'linear'}`  
`0.8044489795918366`

In [42]: `SVC = svm.SVC(C= 1, kernel= 'linear')`  
`SVC.fit(x_train,y_train)`  
`y_pred = SVC.predict(x_test)`  
`svc_accuracy=accuracy_score(y_test,y_pred)`  
`print(classification_report(y_test,y_pred))`  
`plot_confusion_matrix(SVC,x_test, y_test)`

	precision	recall	f1-score	support
N	0.88	0.42	0.57	33
Y	0.82	0.98	0.89	96
accuracy			0.83	123
macro avg	0.85	0.70	0.73	123
weighted avg	0.84	0.83	0.81	123

Out[42]: `<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d82f3dcbc8>`



In [43]: `# achieved accuracy for testing data`  
`print("For Logistic Regression ==" + str(lr_accuracy*100))`  
`print("For KNearest Neighbors ==" + str(KNN_accuracy*100))`  
`print("For Random Forest ==" + str(RF_accuracy*100))`  
`print("For Naive Bayes ==" + str(NB_accuracy*100))`  
`print("For Support Vector Machine ==" + str(svc_accuracy*100))`

For Logistic Regression =83.73983739837398  
For KNearest Neighbors ~78.86178861788618  
For Random Forest ~82.92682926829268  
For Naive Bayes =82.92682926829268  
For Support Vector Machine =82.92682926829268

In [ ]: