Industry Assignment 1 - Machine Learning

Loan Approval Project

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from imblearn.combine import SMOTETomek
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, precision_score, make_scorer, accuracy_score, classification_repo
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

Reading and Loading the Dataset

In [2]:
```
data = pd.read_csv('ML Project 1 Dataset.csv')

data.head()
```

Out[2]:

	APP_ID	CIBIL_SCORE_VALUE	NEW_CUST	CUS_CATGCODE	EMPLOYMENT_TYPE	AGE	SEX	NO_OF_DEPENDENTS	MARITAL	EDU_QU
0	12345	0	YES	1	0	31	F	3	0	
1	12347	0	NO	1	1	40	F	2	1	
2	12349	0	YES	1	0	27	F	3	0	
3	12351	2	NO	1	1	33	M	2	0	
4	12353	2	NO	1	1	29	F	1	0	

In [3]:
```
data = data.rename(columns=lambda x:x.strip())
```

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13299 entries, 0 to 13298
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   APP_ID             13299 non-null  int64
 1   CIBIL_SCORE_VALUE  13299 non-null  int64
 2   NEW_CUST           13299 non-null  object
 3   CUS_CATGCODE       13299 non-null  int64
 4   EMPLOYMENT_TYPE    13299 non-null  int64
 5   AGE                13299 non-null  int64
 6   SEX                13299 non-null  object
 7   NO_OF_DEPENDENTS   13299 non-null  int64
 8   MARITAL            13299 non-null  int64
 9   EDU_QUA            13299 non-null  int64
 10  P_RESTYPE          13299 non-null  int64
 11  P_CATEGORY         13299 non-null  int64
 12  EMPLOYEE_TYPE      13299 non-null  int64
 13  MON_IN_OCC         13299 non-null  int64
 14  INCOM_EXP_GMI      13299 non-null  int64
 15  LTV                13299 non-null  float64
 16  TENURE             13299 non-null  int64
 17  STATUS             13299 non-null  int64
dtypes: float64(1), int64(15), object(2)
memory usage: 1.8+ MB
```

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
APP_ID                0
CIBIL_SCORE_VALUE     0
NEW_CUST              0
CUS_CATGCODE          0
EMPLOYMENT_TYPE       0
AGE                   0
SEX                   0
NO_OF_DEPENDENTS      0
MARITAL               0
EDU_QUA               0
P_RESTYPE             0
P_CATEGORY            0
EMPLOYEE_TYPE         0
MON_IN_OCC            0
INCOM_EXP_GMI         0
LTV                   0
TENURE                0
STATUS                0
dtype: int64
```

In [6]:
```
data.describe()
```

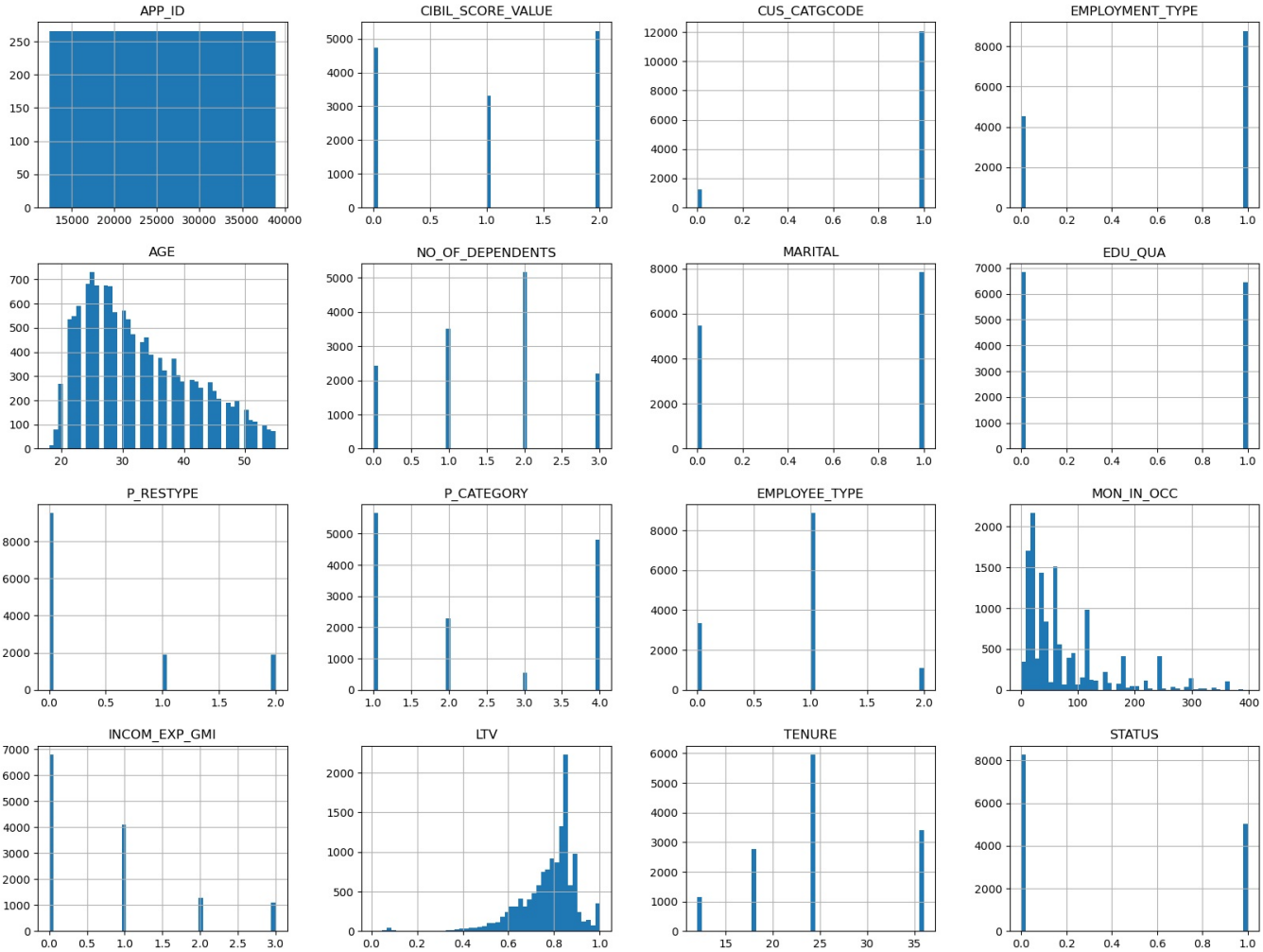
Out[6]:

	APP_ID	CIBIL_SCORE_VALUE	CUS_CATGCODE	EMPLOYMENT_TYPE	AGE	NO_OF_DEPENDENTS	MARITAL	EDU
count	13299.0000	13299.000000	13299.000000	13299.000000	13299.000000	13299.000000	13299.000000	13299.0
mean	25643.0000	1.037898	0.908640	0.658922	32.473870	1.536281	0.590044	0.4
std	7678.4699	0.865391	0.288132	0.474089	8.804317	0.971671	0.491844	0.4
min	12345.0000	0.000000	0.000000	0.000000	18.000000	0.000000	0.000000	0.0
25%	18994.0000	0.000000	1.000000	0.000000	25.000000	1.000000	0.000000	0.0
50%	25643.0000	1.000000	1.000000	1.000000	31.000000	2.000000	1.000000	0.0
75%	32292.0000	2.000000	1.000000	1.000000	38.000000	2.000000	1.000000	1.0
max	38941.0000	2.000000	1.000000	1.000000	55.000000	3.000000	1.000000	1.0

In [ ]:
```
```

In [7]:
```
data.hist(bins=50, figsize=(20,15))
```

Out[7]:
```
array([[<Axes: title={'center': 'APP_ID'}>,
        <Axes: title={'center': 'CIBIL_SCORE_VALUE'}>,
        <Axes: title={'center': 'CUS_CATGCODE'}>,
        <Axes: title={'center': 'EMPLOYMENT_TYPE'}>],
       [<Axes: title={'center': 'AGE'}>,
        <Axes: title={'center': 'NO_OF_DEPENDENTS'}>,
        <Axes: title={'center': 'MARITAL'}>,
        <Axes: title={'center': 'EDU_QUA'}>],
       [<Axes: title={'center': 'P_RESTYPE'}>,
        <Axes: title={'center': 'P_CATEGORY'}>,
        <Axes: title={'center': 'EMPLOYEE_TYPE'}>,
        <Axes: title={'center': 'MON_IN_OCC'}>],
       [<Axes: title={'center': 'INCOM_EXP_GMI'}>,
        <Axes: title={'center': 'LTV'}>,
        <Axes: title={'center': 'TENURE'}>,
        <Axes: title={'center': 'STATUS'}>]], dtype=object)
```



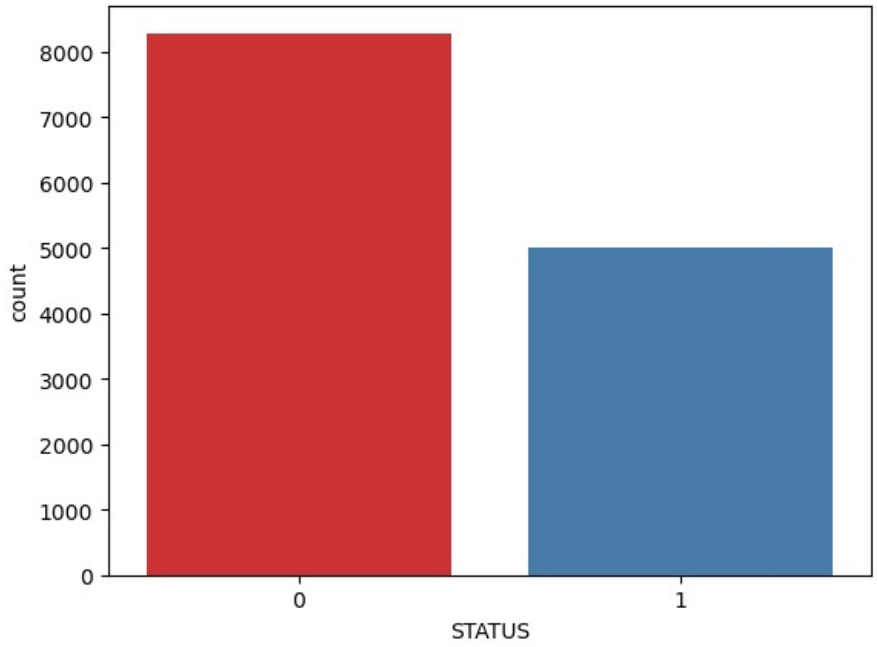
In [8]:
```
print("per of missing  gender is %2f%%" %((data['SEX'].isnull().sum()/data.shape[0])*100))
```

```
per of missing  gender is 0.000000%
```

In [9]:
```
print("Number of people who take loan as group by Marital Status: ")
print(data['STATUS'].value_counts())
sns.countplot(x='STATUS', data=data, palette='Set1')
```

```
Number of people who take loan as group by Marital Status:
0    8283
1    5016
Name: STATUS, dtype: int64
<Axes: xlabel='STATUS', ylabel='count'>
```

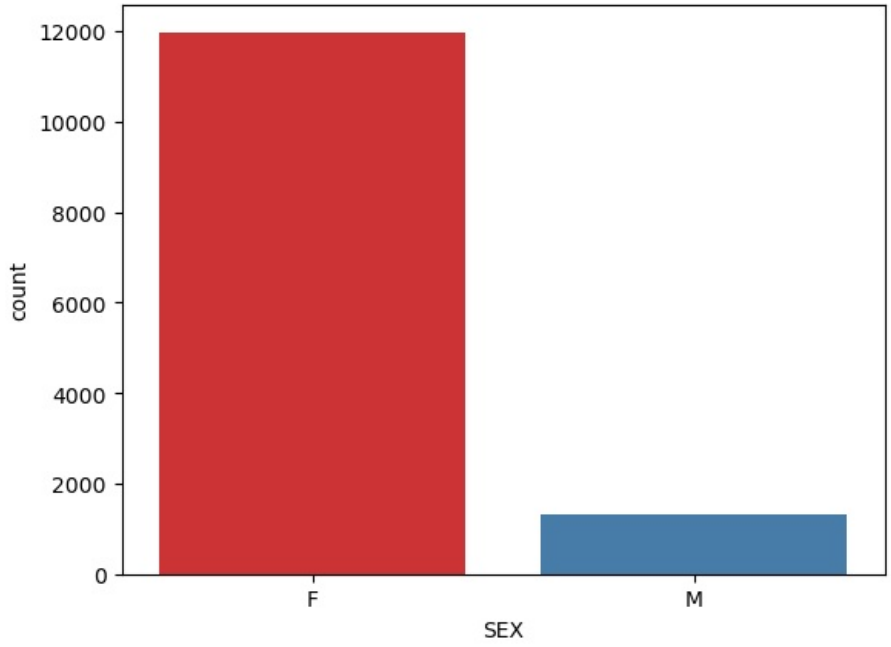
Out[9]:



In [10]:
```
print("Number of people who take loan as group by Sex status: ")
print(data['SEX'].value_counts())
sns.countplot(x='SEX', data=data, palette='Set1')
```

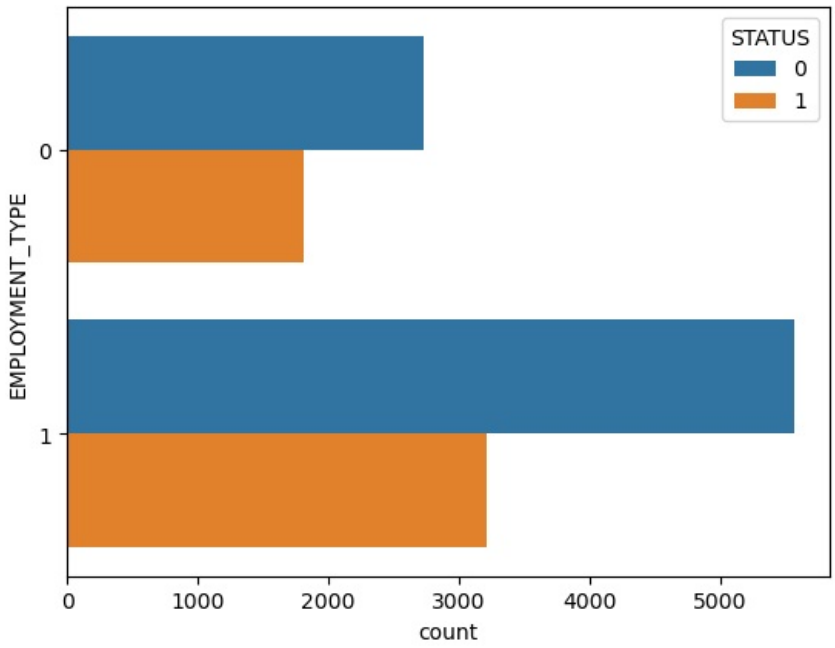
```
Number of people who take loan as group by Sex status:
F    11975
M     1324
Name: SEX, dtype: int64
```

Out[10]:
```
<Axes: xlabel='SEX', ylabel='count'>
```



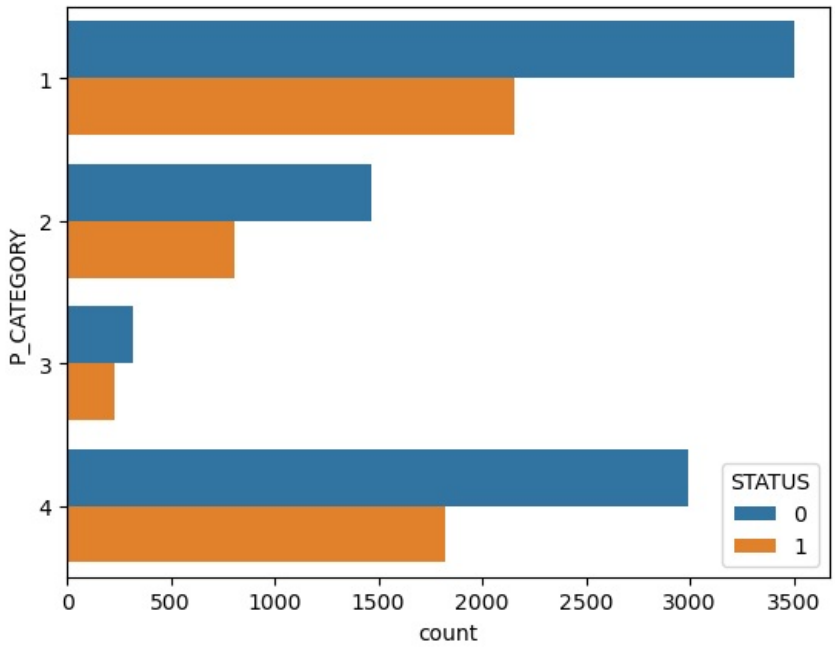
In [11]:
```
sns.countplot(y='EMPLOYMENT_TYPE', hue="STATUS", data=data)
```

Out[11]:
```
<Axes: xlabel='count', ylabel='EMPLOYMENT_TYPE'>
```



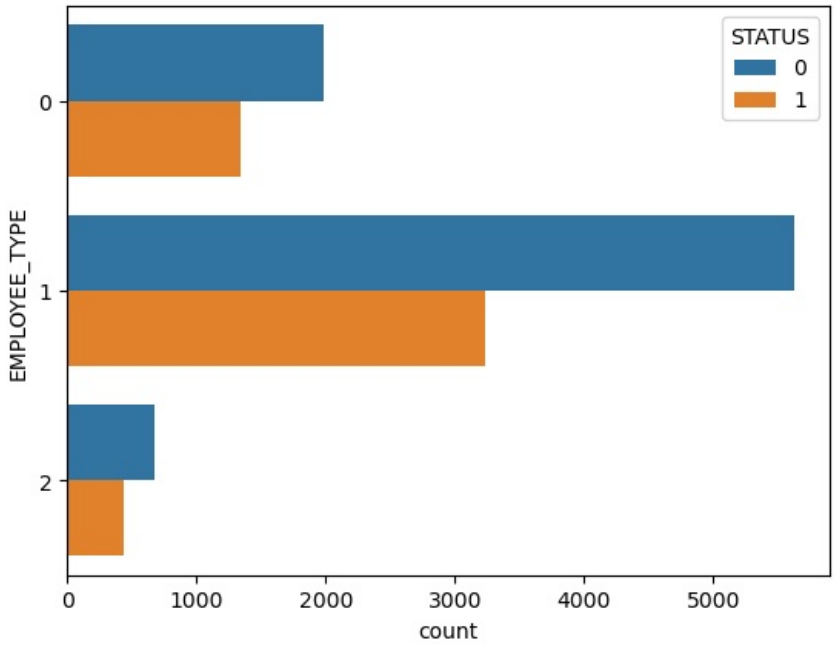
In [12]:
```
sns.countplot(y="P_CATEGORY", hue="STATUS", data=data)
```

Out[12]:
```
<Axes: xlabel='count', ylabel='P_CATEGORY'>
```



In [13]:
```
sns.countplot(y="EMPLOYEE_TYPE", hue="STATUS", data=data)
```

Out[13]:
```
<Axes: xlabel='count', ylabel='EMPLOYEE_TYPE'>
```



In [14]:
```
le = LabelEncoder()
```

In [15]:
```
data.NEW_CUST=le.fit_transform(data['NEW_CUST'])
data.SEX=le.fit_transform(data['SEX'])
data.head()
```

Out[15]:

	APP_ID	CIBIL_SCORE_VALUE	NEW_CUST	CUS_CATGCODE	EMPLOYMENT_TYPE	AGE	SEX	NO_OF_DEPENDENTS	MARITAL	EDU_QU
0	12345	0	1	1	0	31	0	3	0	
1	12347	0	0	1	1	40	0	2	1	
2	12349	0	1	1	0	27	0	3	0	
3	12351	2	0	1	1	33	1	2	0	
4	12353	2	0	1	1	29	0	1	0	

In [16]:
```
data = data.drop(["APP_ID"],axis=1)
```

In [17]:
```
ss=StandardScaler()
x=data.drop(columns=['STATUS'],axis=1)
y=data['STATUS']
```

In [18]:
```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [19]:
```
smt=SMOTETomek(random_state=42)
x_train, y_train=smt.fit_resample(x_train,y_train)
x_test, y_test=smt.fit_resample(x_test, y_test)
```

In [20]:
```
logistic_regression=LogisticRegression()
logistic_regression.fit(x_train, y_train)
```

```
C:\Users\khush\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs fai
led to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[20]:
```
▾ LogisticRegression
LogisticRegression()
```

In [21]:
```
y_pred=logistic_regression.predict(x_test)
```

In [22]:
```
score=accuracy_score(y_pred, y_test)
print(score)
print(classification_report(y_pred, y_test))
```

```
0.68234100135318
              precision    recall  f1-score   support

           0       0.76      0.66      0.71      1709
           1       0.60      0.72      0.66      1247

    accuracy                           0.68      2956
   macro avg       0.68      0.69      0.68      2956
weighted avg       0.69      0.68      0.68      2956
```

In [23]:
```
svc=SVC()
svc.fit(x_train, y_train)
```

Out[23]:
```
SVC()
```

In [24]:
```
svc_y_pred=svc.predict(x_test)
```

In [25]:
```
score1=accuracy_score(svc_y_pred, y_test)
print(score1)
print(classification_report(svc_y_pred, y_test))
```

```
0.5581867388362652
              precision    recall  f1-score   support

           0       0.64      0.55      0.59      1730
           1       0.47      0.57      0.52      1226

    accuracy                           0.56      2956
   macro avg       0.56      0.56      0.55      2956
weighted avg       0.57      0.56      0.56      2956
```

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
decision_tree=DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
```

Out[26]:
```
DecisionTreeClassifier()
```

In [27]:
```
decision_y_pred=decision_tree.predict(x_test)
```

In [28]:
```
score2=accuracy_score(decision_y_pred, y_test)
print(score2)
print(classification_report(decision_y_pred, y_test))
```

```
0.5869418132611637
              precision    recall  f1-score   support

           0       0.61      0.58      0.59      1535
           1       0.57      0.59      0.58      1421

    accuracy                           0.59      2956
   macro avg       0.59      0.59      0.59      2956
weighted avg       0.59      0.59      0.59      2956
```

In [29]:
```
rf_clf=RandomForestClassifier()
rf_clf.fit(x_train, y_train)
```

Out[29]:
```
RandomForestClassifier()
```

In [30]:
```
y_pred=rf_clf.predict(x_test)
score3=metrics.accuracy_score(y_pred, y_test)

print("Acc of Random forest clf is: ", metrics.accuracy_score(y_pred,y_test))
y_pred
```

```
Acc of Random forest clf is:  0.6728687415426252
array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

In [31]:
```
kn_clf=KNeighborsClassifier()
kn_clf.fit(x_train, y_train)
```

Out[31]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [32]:
```
y_pred=kn_clf.predict(x_test)
score4=metrics.accuracy_score(y_pred, y_test)

print("acc of KN clf is", metrics.accuracy_score(y_pred, y_test))
y_pred
```

```
acc of KN clf is 0.5324763193504736
```
Out[32]:
```
array([0, 1, 1, ..., 1, 1, 0], dtype=int64)
```

In [33]:
```
param_grid={
    'n_estimators':[50,100,150], #Number of trees in the random forest
    'max_depth':[None,5,10], #Maximum depth of each tree
    'min_samples_split':[2,5,10], #Minimum number of samples required to split a node
    'min_samples_leaf':[1,2,4], #Minimum number of samples required at each leaf node
}
```

In [34]:
```
rf_clf=RandomForestClassifier(random_state=42)
```

In [35]:
```
grid_search = GridSearchCV(rf_clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
```

Out[35]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [41]:
```
print("Best Hyperparameters: ",grid_search.best_params_)
```

```
Best Hyperparameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 150}
```

In [37]:
```
best_model=grid_search.best_estimator_
test_accuracy=best_model.score(x_test,y_test)
print("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 0.6748985115020297
```

In [42]:
```
scorer = make_scorer(precision_score)
```

In [39]:
```
print("Best Hyperparameters: ", grid_search.best_params_)
```

```
Best Hyperparameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 150}
```

In [43]:
```
best_model=grid_search.best_estimator_
y_pred=best_model.predict(x_test)
test_precision=precision_score(y_test, y_pred, )
print("Test Precision: ", test_precision)
```

```
Test Precision:  0.7027450980392157
```

In [ ]:
```

```