

# TITANIC SURVIVAL

## Importing libraries

---

```
In [24]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from scipy.stats import chi2_contingency
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

## Loading dataset

---

```
In [25]: df = pd.read_csv('/kaggle/input/test-file/tested.csv')
```

In [26]: `df.head()`

Out[26]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN

## Analyzing data

```
In [27]: def summary(df):
sum = pd.DataFrame(df.dtypes, columns=['dtypes'])
sum['missing#'] = df.isna().sum().values
sum['missing%'] = (df.isna().sum().values*100)/len(df)
sum['uniques'] = df.nunique().values
sum['count'] = df.count().values
#sum['skew'] = df.skew().values
desc = pd.DataFrame(df.describe().T)
sum['min'] = desc['min']
sum['max'] = desc['max']
sum['mean'] = desc['mean']
return sum

summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

Out[27]:

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>PassengerId</b>	int64	0	0.000000	418	418	892.000000	1309.000000	1100.500000
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Name</b>	object	0	0.000000	418	418	nan	nan	nan
<b>Sex</b>	object	0	0.000000	2	418	nan	nan	nan
<b>Age</b>	float64	86	20.574163	79	332	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Ticket</b>	object	0	0.000000	363	418	nan	nan	nan
<b>Fare</b>	float64	1	0.239234	169	417	0.000000	512.329200	35.627188
<b>Cabin</b>	object	327	78.229665	76	91	nan	nan	nan
<b>Embarked</b>	object	0	0.000000	3	418	nan	nan	nan

In [28]: df.shape

Out[28]: (418, 12)

- We drop the cabin values since it has a lot of null values

In [29]: df.drop(columns=['Cabin'], inplace=True)

```
In [30]: summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

```
Out[30]:
```

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>PassengerId</b>	int64	0	0.000000	418	418	892.000000	1309.000000	1100.500000
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Name</b>	object	0	0.000000	418	418	nan	nan	nan
<b>Sex</b>	object	0	0.000000	2	418	nan	nan	nan
<b>Age</b>	float64	86	20.574163	79	332	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Ticket</b>	object	0	0.000000	363	418	nan	nan	nan
<b>Fare</b>	float64	1	0.239234	169	417	0.000000	512.329200	35.627188
<b>Embarked</b>	object	0	0.000000	3	418	nan	nan	nan

- We use fillna method to fill the null values in Fare and Age columns using mean fill

```
In [31]: df['Age'].fillna(df['Age'].mean(), inplace=True)
```

```
In [32]: df['Fare'].fillna(df['Fare'].mean(), inplace=True)
```

```
In [33]: summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

Out[33]:

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>PassengerId</b>	int64	0	0.000000	418	418	892.000000	1309.000000	1100.500000
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Name</b>	object	0	0.000000	418	418	nan	nan	nan
<b>Sex</b>	object	0	0.000000	2	418	nan	nan	nan
<b>Age</b>	float64	0	0.000000	80	418	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Ticket</b>	object	0	0.000000	363	418	nan	nan	nan
<b>Fare</b>	float64	0	0.000000	170	418	0.000000	512.329200	35.627188
<b>Embarked</b>	object	0	0.000000	3	418	nan	nan	nan

- We drop the passengerid and the name column

```
In [34]: df.drop(columns=['PassengerId', 'Name'], inplace=True)
```

```
In [35]: summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

Out[35]:

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Sex</b>	object	0	0.000000	2	418	nan	nan	nan
<b>Age</b>	float64	0	0.000000	80	418	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Ticket</b>	object	0	0.000000	363	418	nan	nan	nan
<b>Fare</b>	float64	0	0.000000	170	418	0.000000	512.329200	35.627188
<b>Embarked</b>	object	0	0.000000	3	418	nan	nan	nan

```
In [36]: df['Ticket'].sample(10)
```

```
Out[36]: 113      329944
338      C.A. 31029
39       1601
302      S.O./P.P. 2
223      350053
306      113781
221      342684
94       13905
69       19950
370      28133
Name: Ticket, dtype: object
```

- We also drop the ticket column

```
In [37]: df.drop(columns=['Ticket'], inplace=True)
```

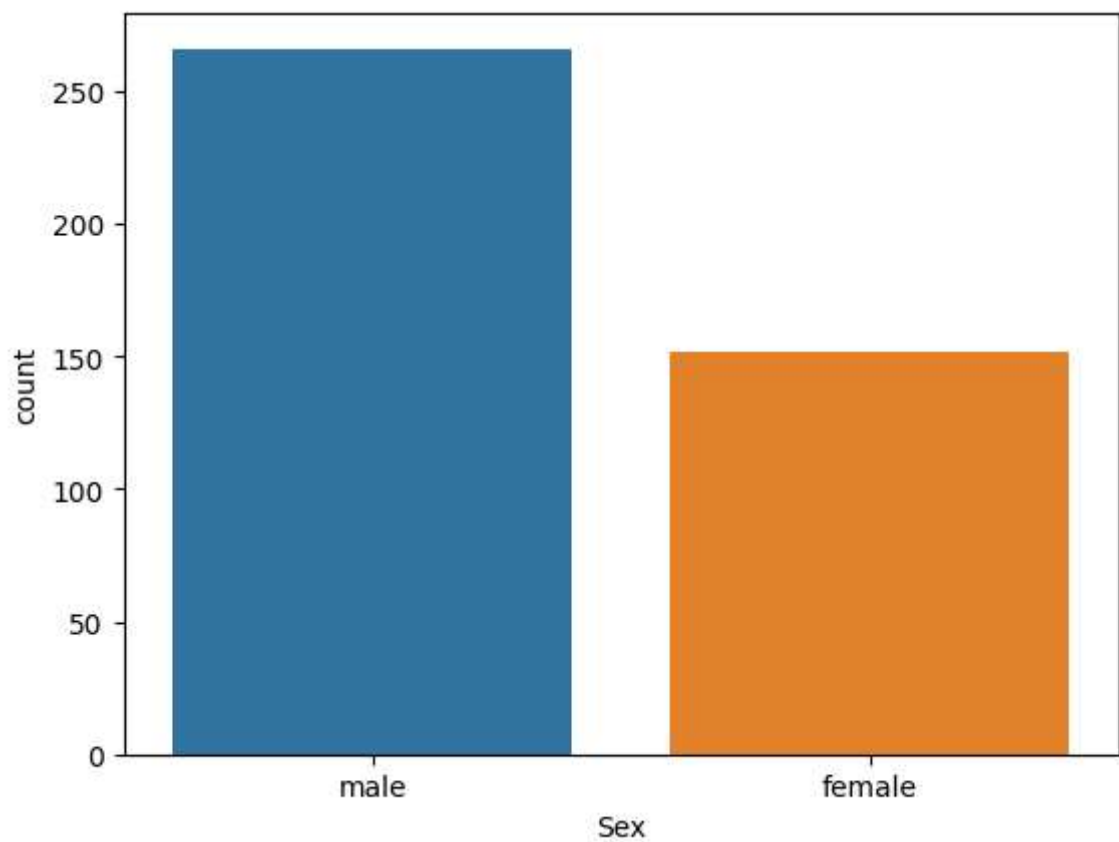
```
In [38]: summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

```
Out[38]:
```

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Sex</b>	object	0	0.000000	2	418	nan	nan	nan
<b>Age</b>	float64	0	0.000000	80	418	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Fare</b>	float64	0	0.000000	170	418	0.000000	512.329200	35.627188
<b>Embarked</b>	object	0	0.000000	3	418	nan	nan	nan

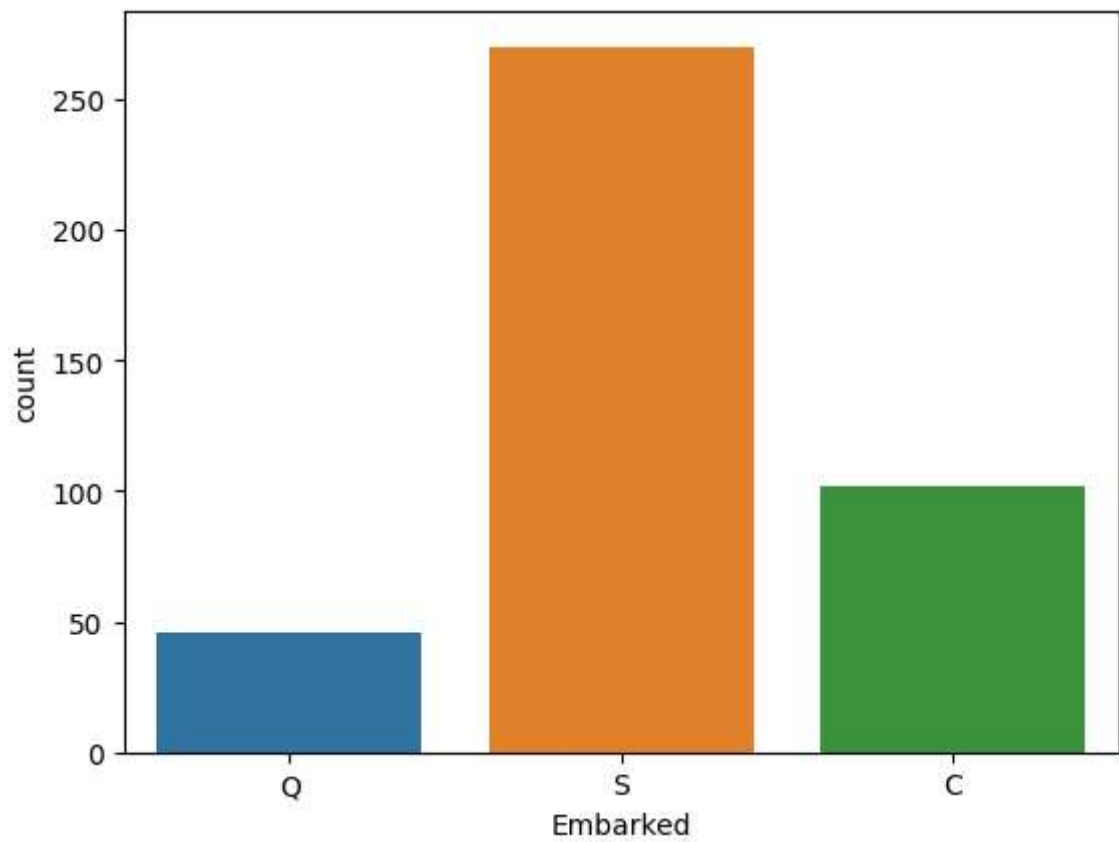
```
In [39]: sns.countplot(x=df['Sex'])
```

```
Out[39]: <Axes: xlabel='Sex', ylabel='count'>
```



```
In [40]: sns.countplot(x=df['Embarked'])
```

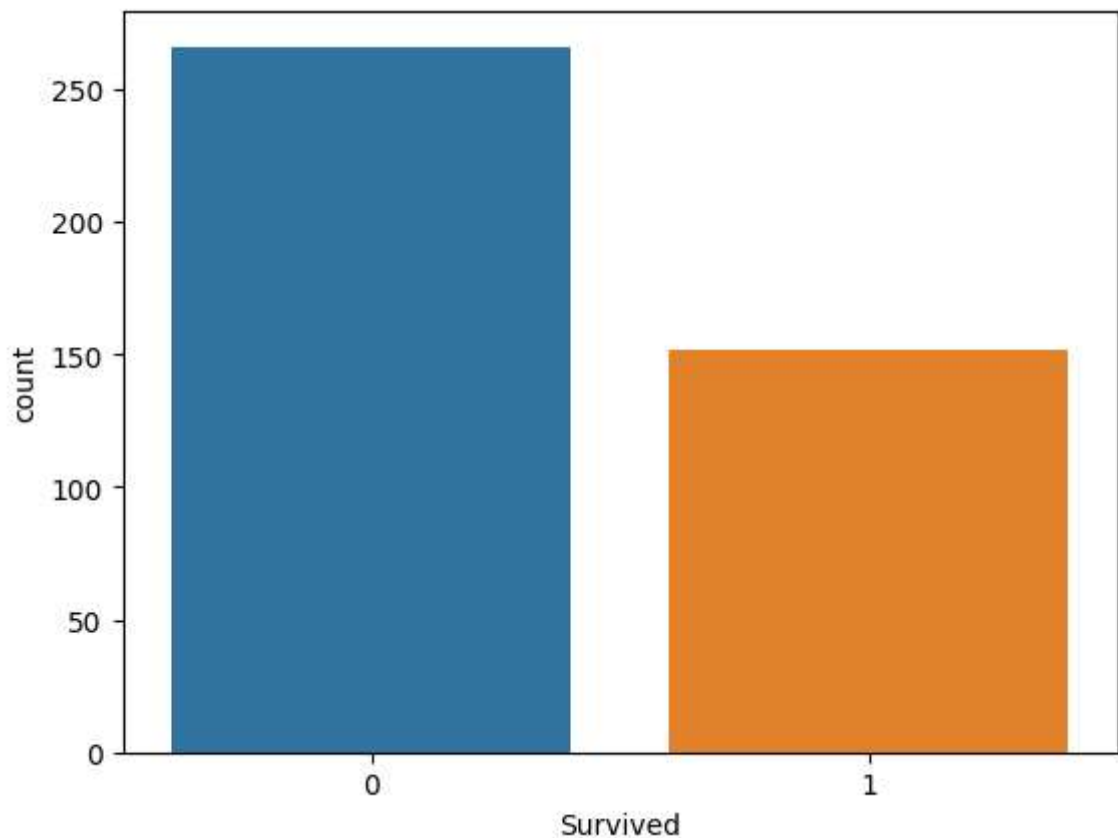
```
Out[40]: <Axes: xlabel='Embarked', ylabel='count'>
```





```
In [41]: sns.countplot(x=df['Survived'])
```

```
Out[41]: <Axes: xlabel='Survived', ylabel='count'>
```



- The data is a bit imbalanced, hence we will use smote methods

## Encoding the variables using LabelEncoder

```
In [42]: le = LabelEncoder()
columnsle=['Sex', 'Embarked']
for i in columnsle:
    df[i] = le.fit_transform(df[i])
```

```
In [43]: summary(df).style.background_gradient(cmap='twilight_shifted_r')
```

Out[43]:

	dtypes	missing#	missing%	uniques	count	min	max	mean
<b>Survived</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.363636
<b>Pclass</b>	int64	0	0.000000	3	418	1.000000	3.000000	2.265550
<b>Sex</b>	int64	0	0.000000	2	418	0.000000	1.000000	0.636364
<b>Age</b>	float64	0	0.000000	80	418	0.170000	76.000000	30.272590
<b>SibSp</b>	int64	0	0.000000	7	418	0.000000	8.000000	0.447368
<b>Parch</b>	int64	0	0.000000	8	418	0.000000	9.000000	0.392344
<b>Fare</b>	float64	0	0.000000	170	418	0.000000	512.329200	35.627188
<b>Embarked</b>	int64	0	0.000000	3	418	0.000000	2.000000	1.401914

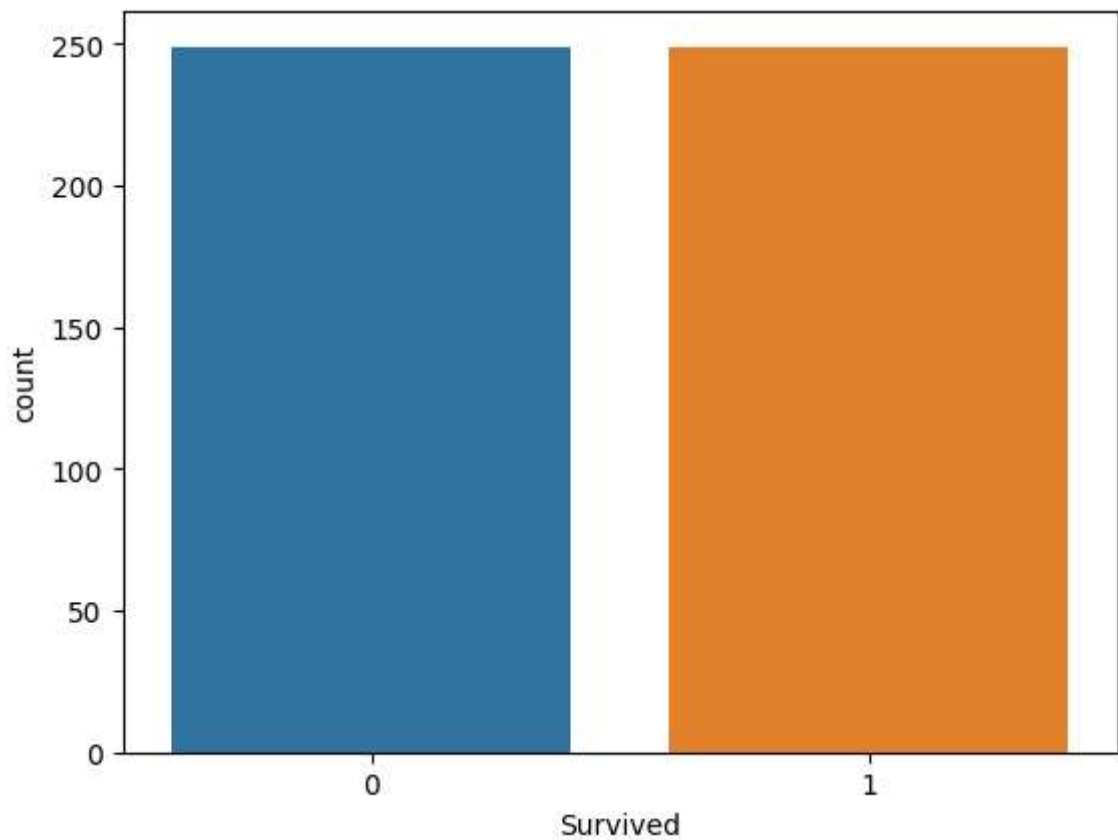
## Balancing the data

```
In [44]: x = df.drop(columns=['Survived'])
         y = df['Survived']
```

```
In [45]: from imblearn.combine import SMOTETomek
         smt=SMOTETomek()
         x,y=smt.fit_resample(x,y)
```

```
In [47]: sns.countplot(x=y)
```

```
Out[47]: <Axes: xlabel='Survived', ylabel='count'>
```



## Applying Models

```
In [48]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=123)
```

```
In [49]: dtree = DecisionTreeClassifier()
rf = RandomForestClassifier()
gb = GradientBoostingClassifier()
ada = AdaBoostClassifier()
knn = KNeighborsClassifier()
lr = LogisticRegression()
```

```
In [50]: dtree.fit(x_train, y_train)
         rf.fit(x_train, y_train)
         gb.fit(x_train, y_train)
         ada.fit(x_train, y_train)
         knn.fit(x_train, y_train)
         lr.fit(x_train, y_train)
```

```
Out[50]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [51]: pdtreetr = dtree.predict(x_train)
         pdtreete = dtree.predict(x_test)

         prftr = rf.predict(x_train)
         prfte = rf.predict(x_test)

         pgbtr = gb.predict(x_train)
         pgbte = gb.predict(x_test)

         padatr = ada.predict(x_train)
         padate = ada.predict(x_test)

         pkntr = knn.predict(x_train)
         pknnte = knn.predict(x_test)
```

```
In [52]: def acc_report(actual,predicted):
         acc_score=accuracy_score(actual,predicted)
         cm_matrix=confusion_matrix(actual,predicted)
         class_rep=classification_report(actual,predicted)
         print('the accuracy of tha model is ',acc_score)
         print(cm_matrix)
         print(class_rep)
```

```
In [53]: print(acc_report(y_train, pdtreetr))
         print(acc_report(y_test, pdtreete))
```

the accuracy of tha model is 1.0

```
[[199  0]
```

```
 [  0 199]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	199
1	1.00	1.00	1.00	199
accuracy			1.00	398
macro avg	1.00	1.00	1.00	398
weighted avg	1.00	1.00	1.00	398

None

the accuracy of tha model is 1.0

```
[[50  0]
```

```
 [  0 50]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	50
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

None

```
In [54]: print(acc_report(y_train, prftr))
         print(acc_report(y_test, prfte))
```

the accuracy of tha model is 1.0

```
[[199  0]
```

```
 [  0 199]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	199
1	1.00	1.00	1.00	199
accuracy			1.00	398
macro avg	1.00	1.00	1.00	398
weighted avg	1.00	1.00	1.00	398

None

the accuracy of tha model is 1.0

```
[[50  0]
```

```
 [  0 50]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	50
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

None

```
In [55]: print(acc_report(y_train, pgbtr))  
         print(acc_report(y_test, pgbte))
```

the accuracy of tha model is 1.0

```
[[199  0]
```

```
 [  0 199]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	199
1	1.00	1.00	1.00	199
accuracy			1.00	398
macro avg	1.00	1.00	1.00	398
weighted avg	1.00	1.00	1.00	398

None

the accuracy of tha model is 1.0

```
[[50  0]
```

```
 [  0 50]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	50
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

None

```
In [56]: print(acc_report(y_train, padatr))
         print(acc_report(y_test, padate))
```

the accuracy of tha model is 1.0

```
[[199  0]
```

```
 [  0 199]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	199
1	1.00	1.00	1.00	199
accuracy			1.00	398
macro avg	1.00	1.00	1.00	398
weighted avg	1.00	1.00	1.00	398

None

the accuracy of tha model is 1.0

```
[[50  0]
```

```
 [  0 50]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	50
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

None



```
In [57]: print(acc_report(y_train, pknntn))
         print(acc_report(y_test, pknnte))
```

the accuracy of tha model is 0.8241206030150754

[[151 48]

[ 22 177]]

	precision	recall	f1-score	support
0	0.87	0.76	0.81	199
1	0.79	0.89	0.83	199
accuracy			0.82	398
macro avg	0.83	0.82	0.82	398
weighted avg	0.83	0.82	0.82	398

None

the accuracy of tha model is 0.8

[[38 12]

[ 8 42]]

	precision	recall	f1-score	support
0	0.83	0.76	0.79	50
1	0.78	0.84	0.81	50
accuracy			0.80	100
macro avg	0.80	0.80	0.80	100
weighted avg	0.80	0.80	0.80	100

None