In [2]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
```

In [3]:
```
iris=load_iris()
X=iris.data
y=iris.target
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1)
```

In [5]:
```
X.shape
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References
```

```
   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [22]:
```python
sc=StandardScaler()
X_train_new=sc.fit_transform(X_train)
X_test_new=sc.transform(X_test)
model=KNeighborsClassifier(n_neighbors=5)
model.fit(X_train_new,y_train)
pred_train=model.predict(X_train_new)
pred_test=model.predict(X_test_new)
print("Train Score:",accuracy_score(y_train,pred_train))
print("Test Score:",accuracy_score(y_test,pred_test))
```

```
Train Score: 0.9553571428571429
Test Score: 0.9736842105263158
```

In [23]:
```python
model.predict([[4.6,1.3,3.9,.5]])
```

Out[23]:
```
array([2])
```

In [25]:
```python
sc=StandardScaler()
X_train_new=sc.fit_transform(X_train)
X_test_new=sc.transform(X_test)
model=KNeighborsClassifier(n_neighbors=5)
model.fit(X_train_new,y_train)
print(model.score(X_train_new,y_train))
print(model.score(X_test_new,y_test))
```

```
0.9553571428571429
0.9736842105263158
```

In [26]:
```python
import pandas as pd
```

In [27]:
```python
df=pd.read_excel("f:/dataset/classification/creditcard_copy.xlsx")
df.shape
```

Out[27]:
```
(5078, 31)
```

In [28]:
```python
df
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...
...	...	...	...	...	...	...	...	...	...	...	...

5073	169142	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...
5074	169347	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...
5075	169351	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...
5076	169966	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...
5077	170348	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...

5078 rows × 31 columns

In [29]:
```
df.Class.value_counts()
```

Out[29]:
```
0    4586
1     492
Name: Class, dtype: int64
```

In [30]:
```
X=df.iloc[:,1:-2].values
y=df.iloc[:,-1].values
```

In [31]:
```
df.describe()
```

Out[31]:

	Time	V1	V2	V3	V4	V5	V6	V7
count	5078.000000	5078.000000	5078.000000	5078.000000	5078.000000	5078.000000	5078.000000	5078.000000
mean	26382.185112	-0.705344	0.584418	-0.103781	0.425607	-0.309467	-0.108354	-0.414131
std	52274.968414	2.863990	2.026708	3.356264	2.102100	2.234907	1.412898	2.988556
min	0.000000	-30.552380	-15.732974	-31.103685	-4.657545	-32.092129	-7.465603	-43.557242
25%	978.000000	-1.179498	-0.167130	-0.265157	-0.800510	-0.678437	-0.838720	-0.490238
50%	2098.500000	-0.462149	0.447474	0.628581	0.224502	-0.106695	-0.292258	0.087072
75%	3321.750000	1.082291	1.037522	1.291780	1.185083	0.433680	0.374438	0.603359
max	170348.000000	2.355634	22.057729	4.017561	12.114672	11.095089	21.393069	34.303177

8 rows × 31 columns

In [32]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1)
model=KNeighborsClassifier()
model.fit(X_train,y_train)
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.9855567226890757
0.9921259842519685
```

In [33]:
```
from sklearn.model_selection import cross_val_score
```

In [34]:
```
model=KNeighborsClassifier()
cross_val_score(model,X,y,cv=5).mean()
```

Out[34]:
```
0.9836552112020479
```

accuracy score---->overall performence of model by considering all classes

f1 score---------->performence of model with class 1

In [35]:
```
from sklearn.metrics import f1_score
```

In [37]:
```
model=KNeighborsClassifier()
model.fit(X_train,y_train)
pred_train=model.predict(X_train)
pred_test=model.predict(X_test)
print("Train F1:",f1_score(y_train,pred_train))
print("Test F1:",f1_score(y_test,pred_test))
```

```
Train F1: 0.9237170596393898
Test F1: 0.9523809523809523
```

In [39]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1)
sc=StandardScaler()
X_train_new=sc.fit_transform(X_train)
X_test_new=sc.transform(X_test)
model=KNeighborsClassifier()
model.fit(X_train_new,y_train)
pred_train=model.predict(X_train_new)
pred_test=model.predict(X_test_new)
print("Train F1:",f1_score(y_train,pred_train))
print("Test F1:",f1_score(y_test,pred_test))
```

```
Train F1: 0.9103641456582633
Test F1: 0.9326923076923077
```

In [40]:
```
print("Train F1:",f1_score(y_train,pred_train,average=None))
print("Test F1:",f1_score(y_test,pred_test,average=None))
```

```
Train F1: [0.99072733 0.91036415]
Test F1: [0.99399657 0.93269231]
```

In [41]:
```
#SMOTE
```

```
  Cell In[41], line 2
    pip install imblearn
        ^
SyntaxError: invalid syntax
```

In [42]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in c:\users\ducat\anaconda3\lib\site-pac
kages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\ducat\anaconda3\lib\site-packag
es (from imbalanced-learn->imblearn) (1.24.3)
Requirement already satisfied: scipy>=1.3.2 in c:\users\ducat\anaconda3\lib\site-package
s (from imbalanced-learn->imblearn) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\ducat\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\ducat\anaconda3\lib\site-packag
es (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ducat\anaconda3\lib\site
-packages (from imbalanced-learn->imblearn) (2.2.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
Note: you may need to restart the kernel to use updated packages.
```

In [43]:
```
from imblearn.over_sampling import SMOTE
```

In [47]:
```
sm=SMOTE()
X1,y1=sm.fit_resample(X,y)
```

In [48]:
```
X.shape
```

Out[48]:
```
(5078, 28)
```

In [49]:
```
X1.shape
```

Out[49]:
```
(9172, 28)
```

In [50]:
```
X_train,X_test,y_train,y_test=train_test_split(X1,y1,random_state=1)
sc=StandardScaler()
X_train_new=sc.fit_transform(X_train)
X_test_new=sc.transform(X_test)
model=KNeighborsClassifier()
model.fit(X_train_new,y_train)
pred_train=model.predict(X_train_new)
pred_test=model.predict(X_test_new)
print("Train F1:",f1_score(y_train,pred_train,average=None))
print("Test F1:",f1_score(y_test,pred_test,average=None))
```

```
Train F1: [0.98699021 0.98713315]
Test F1: [0.9845338  0.98493328]
```

In [ ]:
```

```