**K-Nearest Neighbors**

import pandas as pd

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

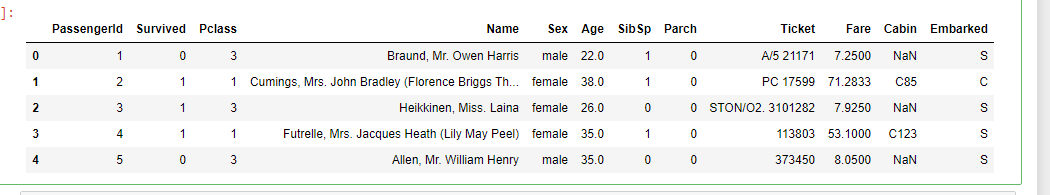
from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn import neighbors

dataset1=pd.read\_csv("C:/Users/PC/Downloads/train.csv")

dataset1.head()



l=preprocessing.LabelEncoder()

dataset1['Sex']=l.fit\_transform(dataset1['Sex'])

dataset1['Embarked']=l.fit\_transform(dataset1['Embarked'])

y1=dataset1['Pclass']

y1.head()

0 3

1 1

2 3

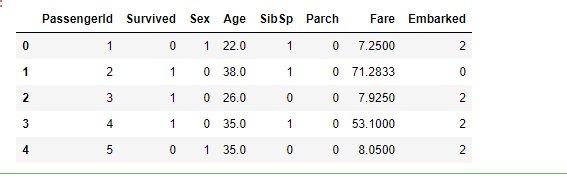
3 1

4 3

Name: Pclass, dtype: int64

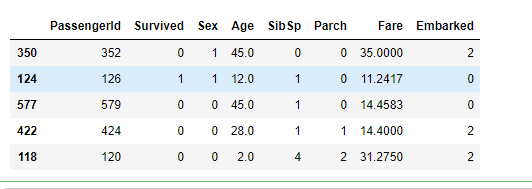
x1=dataset1.drop(columns=['Pclass','Name','Cabin','Ticket'])

x1.head()



X1\_train,X1\_test,y1\_train,y1\_test=train\_test\_split(x1,y1,test\_size=0.3,random\_state=0)

X1\_train.head()



def nearest(k):

knn=neighbors.KNeighborsClassifier(n\_neighbors=k)

knn.fit(X1\_train,y1\_train)

y\_prdict=knn.predict(X1\_test)

y=accuracy\_score(y1\_test,y\_prdict,normalize=True)

return y

for i in range(172):

i=i+1

print(nearest(i))

Output:

0.6741573033707865

0.5880149812734082

0.6816479400749064

0.6704119850187266

0.6591760299625468

0.6554307116104869

0.6779026217228464

0.6741573033707865

0.700374531835206

0.6928838951310862

0.6816479400749064

0.6966292134831461

0.7153558052434457

0.7078651685393258

0.7116104868913857

0.7191011235955056

0.7228464419475655

0.7078651685393258

0.7116104868913857

0.700374531835206

0.7116104868913857

0.7191011235955056

0.7153558052434457

0.7116104868913857

0.700374531835206

0.704119850187266

0.7078651685393258

0.7116104868913857

0.7116104868913857

0.704119850187266

0.6928838951310862

0.6853932584269663

0.6779026217228464

0.6779026217228464

0.6779026217228464

0.6741573033707865

0.6779026217228464

0.6779026217228464

0.6666666666666666

0.6666666666666666

0.6666666666666666

0.6666666666666666

0.6591760299625468

0.6554307116104869

0.6441947565543071

0.6479400749063671

0.6479400749063671

0.6441947565543071

0.6404494382022472

0.6404494382022472

0.6367041198501873

0.6292134831460674

0.6292134831460674

0.6254681647940075

0.6254681647940075

0.6254681647940075

0.6254681647940075

0.6254681647940075

0.6254681647940075

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6217228464419475

0.6179775280898876

0.6104868913857678

0.602996254681648

0.6067415730337079

0.6067415730337079

0.6067415730337079

0.602996254681648

0.602996254681648

0.602996254681648

0.599250936329588

0.602996254681648

0.5955056179775281

0.5917602996254682

0.5880149812734082

0.5880149812734082

0.5880149812734082

0.5880149812734082

0.5880149812734082

0.5880149812734082

0.5842696629213483

0.5842696629213483

0.5842696629213483

0.5842696629213483

0.5805243445692884

0.5767790262172284

0.5767790262172284

0.5767790262172284

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5730337078651685

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5692883895131086

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5655430711610487

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5617977528089888

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

0.5580524344569289

**SVM – Support Vector Machine**

import pandas as pd

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

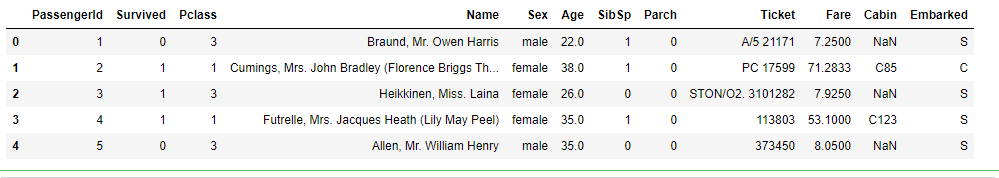
from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn import svm

dataset1=pd.read\_csv("C:/Users/PC/Downloads/train.csv")

dataset1.head()



l=preprocessing.LabelEncoder()

dataset1['Sex']=l.fit\_transform(dataset1['Sex'])

dataset2=dataset1.drop(columns=['Name','Cabin','Ticket','Age','Fare','PassengerId'])

l=['Survived','Pclass','Sex','SibSp','Parch','Embarked']

for i in l:

y1=dataset2[i]

x1=dataset2.drop(columns=i)

X1\_train,X1\_test,y1\_train,y1\_test=train\_test\_split(x1,y1,test\_size=0.3,random\_state=0)

clf=svm.SVC(gamma=0.01,C=100)

clf.fit(X1\_train,y1\_train)

y\_predict=clf.predict(X1\_test)

y=accuracy\_score(y1\_test,y\_predict,normalize=True)

print(y)

Output:

0.7790262172284644

0.5730337078651685

0.7640449438202247

0.6966292134831461

0.7602996254681648

0.7340823970037453