```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import plotly.express as px
import plotly.io as pio
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report

import warnings
warnings.filterwarnings("ignore")
import plotly.express as px
import plotly.io as pio


df=pd.read_csv("/content/titanic.csv")


df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
```

```
Pclass          0
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

```
df.drop("Cabin",axis=1,inplace=True)
```

```
df["Embarked"].fillna("S",inplace=True)
```

```
def fillage(cols):
 Age = cols[0]
 Pclass = cols[1]

 if(pd.isnull(Age)):
    if(Pclass==1):
        return 38
    elif(Pclass==2):
         return 29
    else:
        return 24
 else:
     return Age
```

```
df["Age"]= df[["Age", "Pclass"]].apply(fillage, axis=1)
```

```
df.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

```
df["Embarked"].value_counts()
```

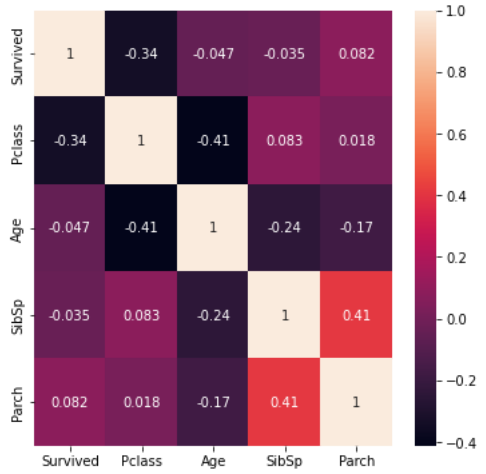
```
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

```
df.drop(["PassengerId","Name","Ticket","Fare"],axis=True,inplace=True)
```

```
df.head()
```

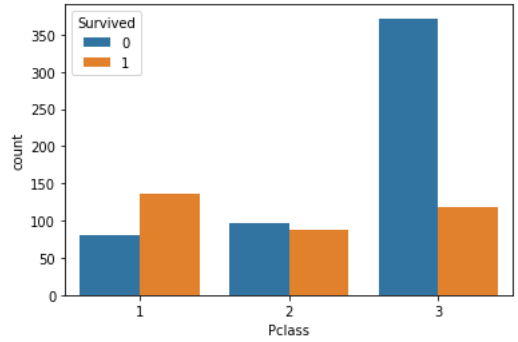
	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S

```
plt.figure(figsize=(6,6))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



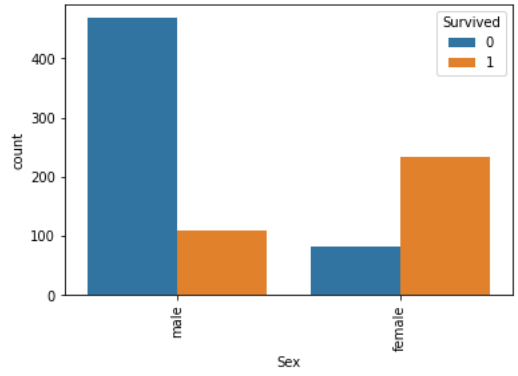

```
sns.countplot(df["Pclass"],hue = df["Survived"],data = df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff63d9fda60>
```




```
sns.countplot(df["Sex"],hue=df["Survived"],data =df)
plt.xticks(rotation=90)
```

```
(array([0, 1]), <a list of 2 Text major ticklabel objects>)
```

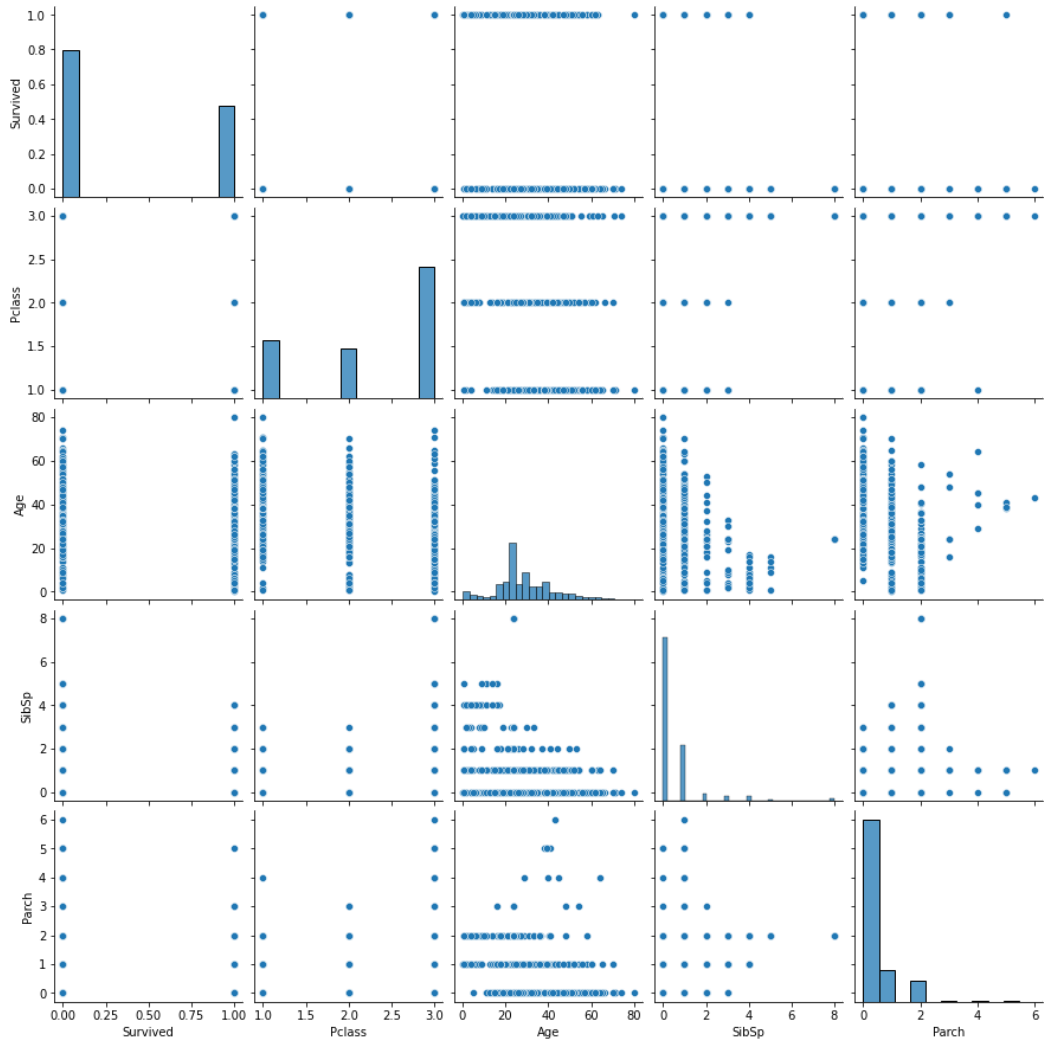



```
sns.relplot(x='Pclass',y='Sex',hue='Age',size='SibSp',data=df)
```

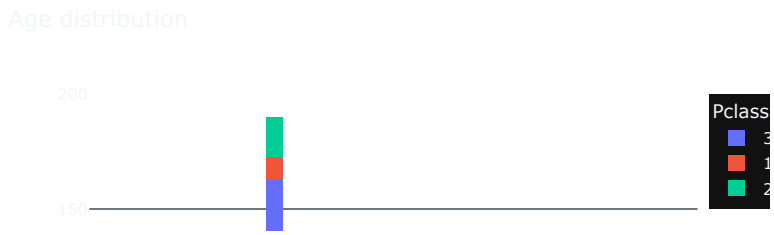
```
<seaborn.axisgrid.FacetGrid at 0x7ff640340070>
```


```
sns.pairplot(df)
```

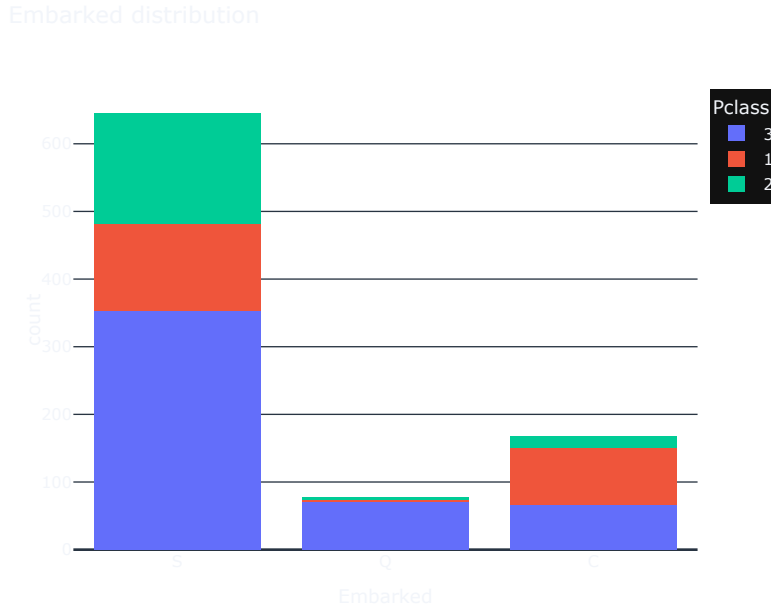
```
<seaborn.axisgrid.PairGrid at 0x7ff63d481fa0>
```



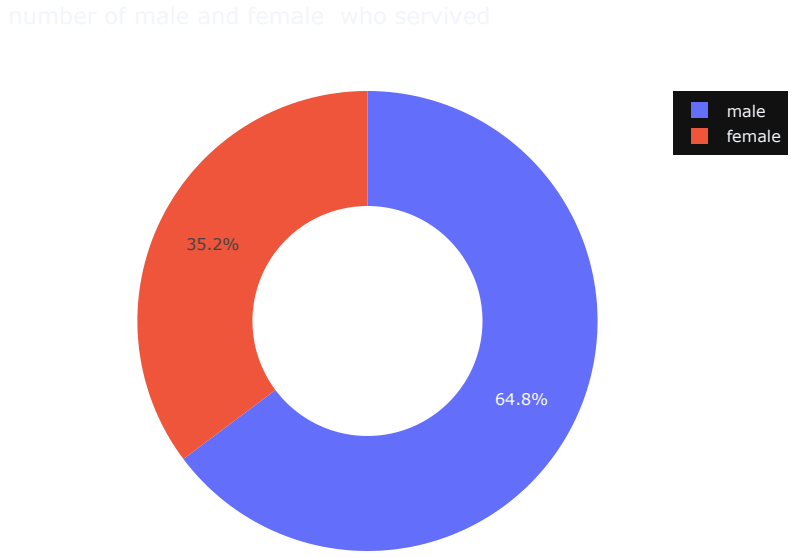

```
ax= px.histogram(df,x= "Age", template= "plotly_dark",color= "Pclass",title='Age distribution')
ax.show()
```



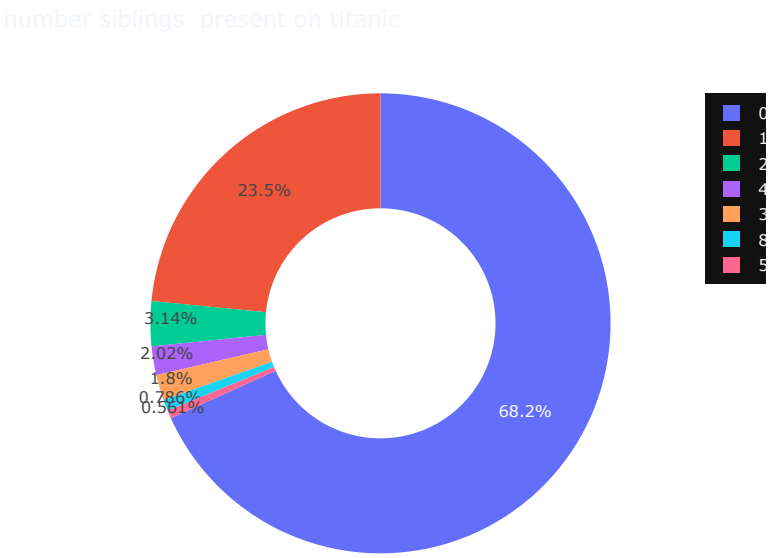

```
ax= px.histogram(df,x= "Embarked", template= "plotly_dark",color= "Pclass",title='Embarked distribution')
ax.show()
```



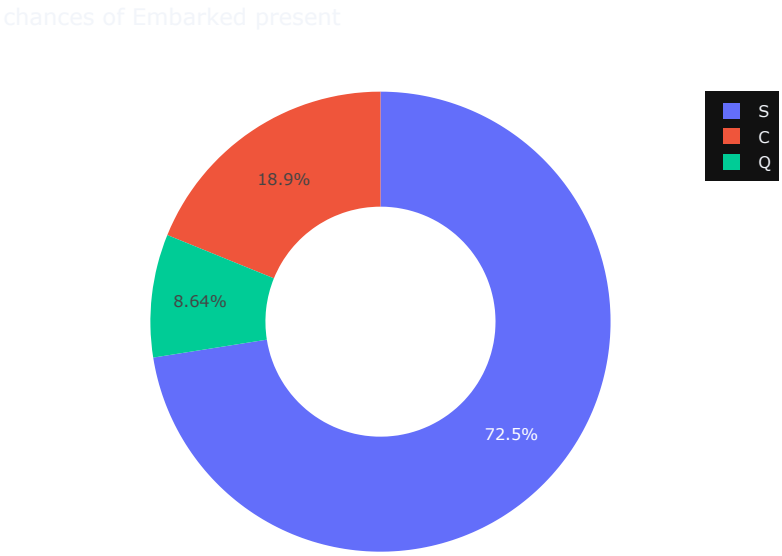

```
ax= px.pie(df, names= "Sex",template= "plotly_dark",title= "number of male and female  who servived",hole= 0.5)
ax.show()
```



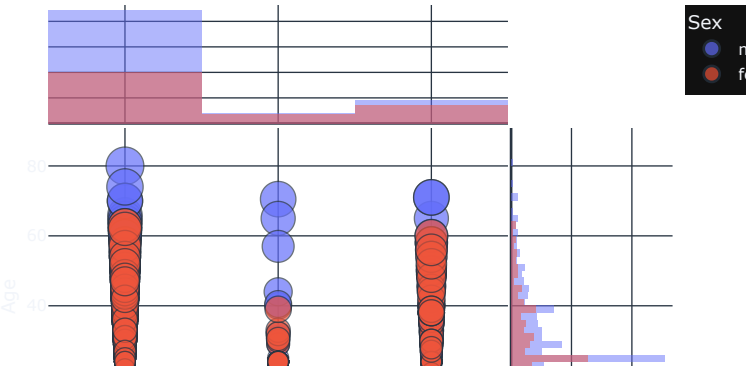
```
ax= px.pie(df, names= "SibSp",template= "plotly_dark",title= "number siblings  present on titanic",hole= 0.5)
ax.show()
```



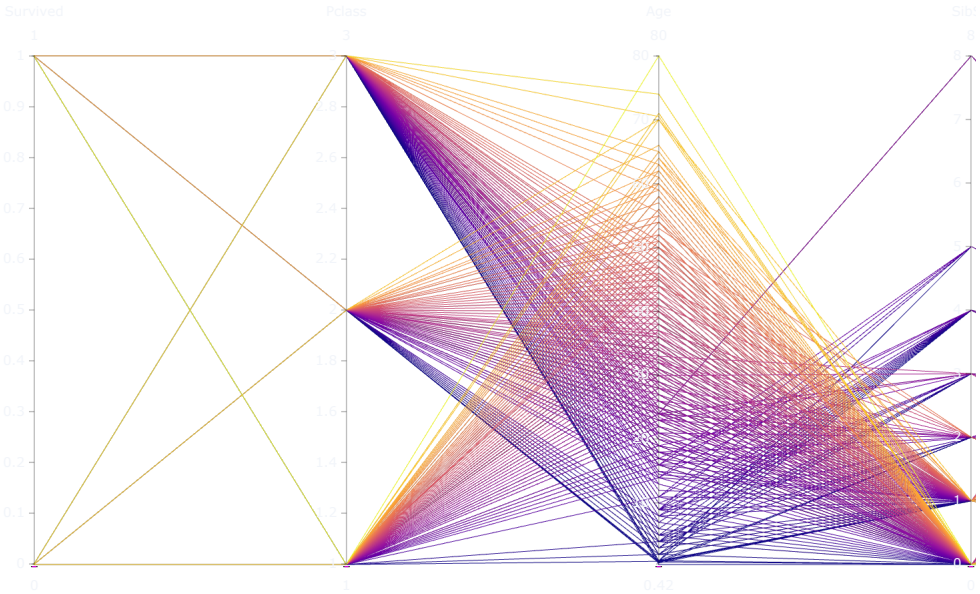

```
ax= px.pie(df, names= "Embarked",template= "plotly_dark",title= "chances of Embarked present",hole= 0.5)
ax.show()
```



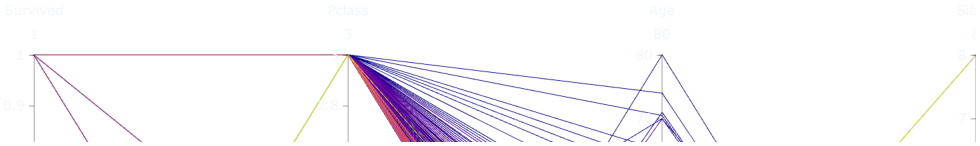

```
ax= px.scatter(df,x= "Embarked",y= "Age",marginal_x='histogram', marginal_y='histogram',size="Age", size_max=20,
               template= "plotly_dark",color= "Sex",title="age and sex correlation")
ax.show()
```



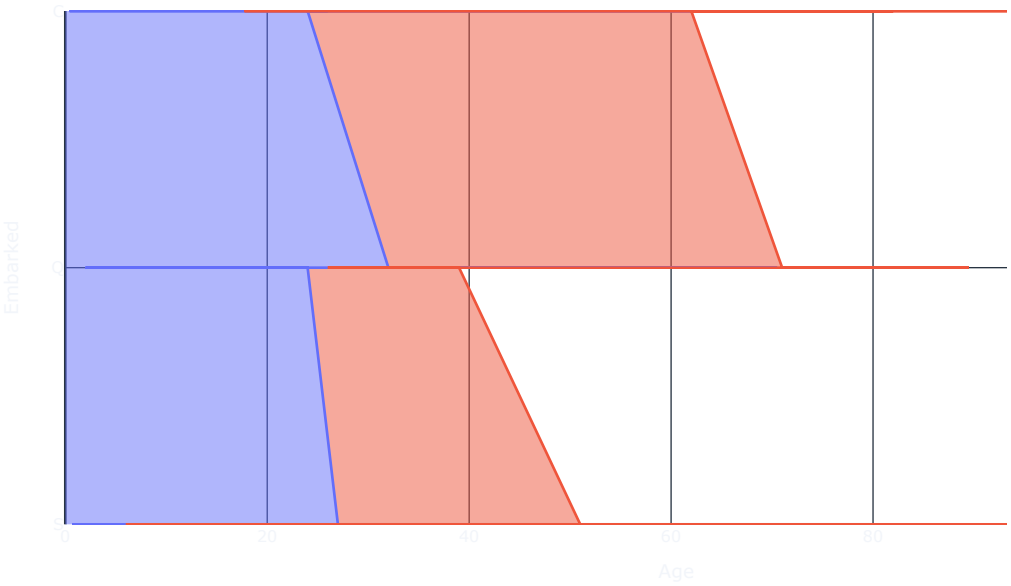

```
ax= px.parallel_coordinates(df, color="Age",template= "plotly_dark")
ax.show()
```




```
ax= px.parallel_coordinates(df, color="SibSp",template= "plotly_dark")
ax.show()
```




```
fig = px.area(df, x="Age", y="Embarked", color="Sex",template= "plotly_dark")
fig.show()
```




```
#splittig x and y for predictoions

x = df.iloc[:, 1:]
y = df.iloc[:,0]
```

x

	Pclass	Sex	Age	SibSp	Parch	Embarked
0	3	male	22.0	1	0	S
1	1	female	38.0	1	0	C
2	3	female	26.0	0	0	S
3	1	female	35.0	1	0	S
4	3	male	35.0	0	0	S
...	...	...	...	...	...	...
886	2	male	27.0	0	0	S
887	1	female	19.0	0	0	S
888	3	female	24.0	1	2	S
889	1	male	26.0	0	0	C
890	3	male	32.0	0	0	Q

891 rows × 6 columns

y

```
0      0
1      1
2      1
3      1
```

```
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

```
ct = ColumnTransformer(transformers= [("encoder", OneHotEncoder(), ["Sex", "Embarked"])],
 remainder="passthrough")
x= ct.fit_transform(x)
```

```
x[0]
```

```
array([ 0.,  1.,  0.,  0.,  1.,  3., 22.,  1.,  0.])
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=0.3, random_state=1)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

```
logreg = LogisticRegression()
knn = KNeighborsClassifier()
svm = SVC()
dt=DecisionTreeClassifier()
```

```
def mymodel(model):
 model.fit(xtrain, ytrain)
 ypred = model.predict(xtest)
 print(classification_report(ytest, ypred))
 return model
```

```
mymodel(logreg)
```

```
                  precision    recall  f1-score   support

               0       0.79      0.85      0.82       153
               1       0.78      0.70      0.73       115

        accuracy                           0.78       268
       macro avg       0.78      0.77      0.78       268
    weighted avg       0.78      0.78      0.78       268

    LogisticRegression()
```

```
mymodel(knn)
```

```
                  precision    recall  f1-score   support

               0       0.74      0.86      0.79       153
               1       0.76      0.59      0.66       115

        accuracy                           0.74       268
       macro avg       0.75      0.72      0.73       268
    weighted avg       0.74      0.74      0.74       268

    KNeighborsClassifier()
```

```
mymodel(svm)
```

```
                  precision    recall  f1-score   support
```

```
                0       0.58      0.95      0.72       153
                1       0.53      0.07      0.12       115

         accuracy                           0.57       268
        macro avg       0.56      0.51      0.42       268
     weighted avg       0.56      0.57      0.46       268

     SVC()


mymodel(dt)

                   precision    recall  f1-score   support

                0       0.76      0.82      0.79       153
                1       0.73      0.65      0.69       115

         accuracy                           0.75       268
        macro avg       0.74      0.73      0.74       268
     weighted avg       0.74      0.75      0.74       268

     DecisionTreeClassifier()


#feature scaling


from sklearn.preprocessing import StandardScaler


sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)


xtest=sc.transform(xtest)


#hypertuning using solver parameter


logreg=LogisticRegression(solver="saga")
logreg.fit(xtrain,ytrain)
ypred=logreg.predict(xtest)


print(classification_report(ytest,ypred))

                   precision    recall  f1-score   support

                0       0.79      0.85      0.82       153
                1       0.78      0.70      0.74       115

         accuracy                           0.79       268
        macro avg       0.79      0.78      0.78       268
     weighted avg       0.79      0.79      0.79       268



print(f"Actual Values -: {ytest[:25].values}")
print(f"Predicted Values -: {ypred[:25]}")

     Actual Values -: [1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0]
     Predicted Values -: [1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1]


logreg.predict_proba(xtest)
```

```
               [0.47819558, 0.52180442],
               [0.21670649, 0.78329351],
               [0.68582996, 0.31417004],
               [0.58788727, 0.41211273],
               [0.8549722 , 0.1450278 ],
               [0.69351746, 0.30648254],
               [0.81861852, 0.18138148],
               [0.95111157, 0.04888843],
               [0.37304832, 0.62695168],
               [0.43484102, 0.56515898],
               [0.28032669, 0.71967331],
               [0.26078607, 0.73921393],
               [0.93127642, 0.06872358],
               [0.95279168, 0.04720832],
               [0.84341575, 0.15658425],
               [0.90265995, 0.09734005],
               [0.93667139, 0.06332861],
               [0.96018782, 0.03981218],
               [0.93390692, 0.06609308],
               [0.17323198, 0.82676802],
               [0.20841932, 0.79158068],
               [0.68460681, 0.31539319],
               [0.06909762, 0.93090238],
               [0.9571042 , 0.0428958 ],
               [0.90426396, 0.09573604],
               [0.66581792, 0.33418208],
               [0.36096151, 0.63903849],
               [0.17140913, 0.82859087],
               [0.25002535, 0.74997465],
               [0.75262534, 0.24737466],
               [0.25723598, 0.74276402],
               [0.84290879, 0.15709121],
               [0.44799457, 0.55200543],
               [0.08682418, 0.91317582],
               [0.91578494, 0.08421506],
               [0.30483432, 0.69516568],
               [0.12900723, 0.87099277],
               [0.76624671, 0.23375329],
               [0.8117365 , 0.1882635 ],
               [0.45297127, 0.54702873],
               [0.75880805, 0.24119195]])
```

```
ypredprob=logreg.predict_proba(xtest)[:,1]
```

```
from sklearn.preprocessing import binarize
```

```
ypred=binarize([ypredprob],threshold=0.30)[0]
```

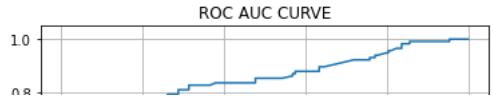
```
print(f"Actual Values -: {ytest[:25].values}")
print(f"Predicted Values -: {ypred[:25]}")
```

```
     Actual Values -: [1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0]
     Predicted Values -: [1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.
      1.]
```

```
from sklearn.metrics import roc_curve
```

```
fpr,tpr,thres=roc_curve(ytest,ypredprob)
```

```
plt.title("ROC AUC CURVE")
plt.plot(fpr,tpr)
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.grid(True)
plt.show()
```




```
#checking overfitted on dt
dt.score(xtrain,ytrain)
```

```
0.21508828250401285
```

```
dt1=DecisionTreeClassifier(max_depth=5)
mymodel(dt1)
```

```
                  precision    recall  f1-score   support

               0       0.75      0.91      0.82       153
               1       0.83      0.59      0.69       115

        accuracy                           0.77       268
       macro avg       0.79      0.75      0.76       268
    weighted avg       0.78      0.77      0.76       268

    DecisionTreeClassifier(max_depth=5)
```

```
for i in range(1,50):
 dt2=DecisionTreeClassifier(max_depth=i)
 dt2.fit(xtrain,ytrain)
 ypred=dt2.predict(xtest)
 print(f"{i}= {accuracy_score(ytest,ypred)}")
```

```
    1= 0.753731343283582
    2= 0.7350746268656716
    3= 0.7723880597014925
    4= 0.7723880597014925
    5= 0.7686567164179104
    6= 0.7611940298507462
    7= 0.7574626865671642
    8= 0.746268656716418
    9= 0.7649253731343284
    10= 0.746268656716418
    11= 0.7388059701492538
    12= 0.7425373134328358
    13= 0.7313432835820896
    14= 0.75
    15= 0.75
    16= 0.753731343283582
    17= 0.7425373134328358
    18= 0.7425373134328358
    19= 0.7425373134328358
    20= 0.75
    21= 0.75
    22= 0.75
    23= 0.7611940298507462
    24= 0.75
    25= 0.753731343283582
    26= 0.75
    27= 0.746268656716418
    28= 0.746268656716418
    29= 0.753731343283582
    30= 0.75
    31= 0.746268656716418
    32= 0.746268656716418
    33= 0.753731343283582
    34= 0.753731343283582
    35= 0.75
    36= 0.753731343283582
    37= 0.746268656716418
    38= 0.75
    39= 0.753731343283582
    40= 0.753731343283582
    41= 0.746268656716418
    42= 0.75
    43= 0.753731343283582
    44= 0.753731343283582
    45= 0.7574626865671642
    46= 0.7574626865671642
    47= 0.75
    48= 0.75
    49= 0.746268656716418
```

```
#best value of max_depth=10
dt3=DecisionTreeClassifier(max_depth=11)
mymodel(dt3)
```

```
                  precision    recall  f1-score   support

               0       0.74      0.84      0.79       153
               1       0.74      0.61      0.67       115

        accuracy                           0.74       268
       macro avg       0.74      0.72      0.73       268
    weighted avg       0.74      0.74      0.73       268

    DecisionTreeClassifier(max_depth=11)
```

```
dt4=DecisionTreeClassifier(min_samples_leaf=10)
mymodel(dt4)
```

```
                  precision    recall  f1-score   support

               0       0.75      0.90      0.82       153
               1       0.81      0.60      0.69       115

        accuracy                           0.77       268
       macro avg       0.78      0.75      0.75       268
    weighted avg       0.78      0.77      0.76       268

    DecisionTreeClassifier(min_samples_leaf=10)
```

```
for i in range(1,75):
 dt5=DecisionTreeClassifier(min_samples_leaf=i)
 dt5.fit(xtrain,ytrain)
 ypred=dt5.predict(xtest)
 print(f"{i}= {accuracy_score(ytest,ypred)}")
```

```
    17= 0.746268656716418
    18= 0.75
    19= 0.75
    20= 0.75
    21= 0.75
    22= 0.75
    23= 0.75
    24= 0.75
    25= 0.75
    26= 0.75
    27= 0.75
    28= 0.7611940298507462
    29= 0.7611940298507462
    30= 0.7611940298507462
    31= 0.7611940298507462
    32= 0.7611940298507462
    33= 0.7611940298507462
    34= 0.7611940298507462
    35= 0.7611940298507462
    36= 0.7611940298507462
    37= 0.7723880597014925
    38= 0.7723880597014925
    39= 0.7723880597014925
    40= 0.7723880597014925
    41= 0.7723880597014925
    42= 0.7723880597014925
    43= 0.7723880597014925
    44= 0.7723880597014925
    45= 0.746268656716418
```

```
62= 0.7462686567164178
63= 0.746268656716418
64= 0.746268656716418
65= 0.746268656716418
66= 0.746268656716418
67= 0.746268656716418
68= 0.746268656716418
69= 0.746268656716418
70= 0.746268656716418
71= 0.746268656716418
72= 0.746268656716418
73= 0.746268656716418
74= 0.746268656716418
```

```
dt6=DecisionTreeClassifier(min_samples_leaf=41)
mymodel(dt6)
```

```
                  precision    recall  f1-score   support

               0       0.74      0.92      0.82       153
               1       0.85      0.57      0.68       115

        accuracy                           0.77       268
       macro avg       0.79      0.75      0.75       268
    weighted avg       0.79      0.77      0.76       268

    DecisionTreeClassifier(min_samples_leaf=41)
```

```
dt7=DecisionTreeClassifier(max_depth=11,min_samples_leaf=41)
mymodel(dt7)
```

```
                  precision    recall  f1-score   support

               0       0.74      0.92      0.82       153
               1       0.85      0.57      0.68       115

        accuracy                           0.77       268
       macro avg       0.79      0.75      0.75       268
    weighted avg       0.79      0.77      0.76       268

    DecisionTreeClassifier(max_depth=11, min_samples_leaf=41)
```

```
#gini
dt8=DecisionTreeClassifier(criterion="gini",min_samples_leaf=15)
mymodel(dt8)
```

```
                  precision    recall  f1-score   support

               0       0.76      0.84      0.80       153
               1       0.75      0.64      0.69       115

        accuracy                           0.75       268
       macro avg       0.75      0.74      0.74       268
    weighted avg       0.75      0.75      0.75       268

    DecisionTreeClassifier(min_samples_leaf=15)
```

```
dt9=DecisionTreeClassifier(criterion="gini",max_depth=35)
mymodel(dt9)
```

```
                  precision    recall  f1-score   support

               0       0.76      0.82      0.79       153
               1       0.74      0.66      0.70       115

        accuracy                           0.75       268
       macro avg       0.75      0.74      0.74       268
    weighted avg       0.75      0.75      0.75       268

    DecisionTreeClassifier(max_depth=35)
```

```
for i in range(1,70):
 dt11=DecisionTreeClassifier(criterion="entropy",max_depth=i)
 dt11.fit(xtrain,ytrain)
 ypred=dt11.predict(xtest)
 print(f"{i}= {accuracy_score(ytest,ypred)}")
```

```
15= 0.7649253731343284
16= 0.753731343283582
17= 0.7649253731343284
18= 0.7574626865671642
19= 0.7611940298507462
20= 0.7649253731343284
21= 0.7649253731343284
22= 0.7574626865671642
23= 0.7574626865671642
24= 0.7574626865671642
25= 0.7611940298507462
26= 0.7574626865671642
27= 0.753731343283582
28= 0.7611940298507462
29= 0.7574626865671642
30= 0.7574626865671642
31= 0.753731343283582
32= 0.7574626865671642
33= 0.7649253731343284
34= 0.7611940298507462
35= 0.7611940298507462
36= 0.7611940298507462
37= 0.7611940298507462
38= 0.7649253731343284
39= 0.7649253731343284
40= 0.7574626865671642
41= 0.7686567164179104
42= 0.7574626865671642
43= 0.7649253731343284
44= 0.7574626865671642
45= 0.7611940298507462
46= 0.7611940298507462
47= 0.7574626865671642
48= 0.7611940298507462
49= 0.7611940298507462
50= 0.7574626865671642
51= 0.7649253731343284
52= 0.7574626865671642
53= 0.7611940298507462
54= 0.7649253731343284
55= 0.7574626865671642
56= 0.7574626865671642
57= 0.7574626865671642
58= 0.7649253731343284
59= 0.7649253731343284
60= 0.753731343283582
61= 0.7611940298507462
62= 0.7611940298507462
63= 0.7611940298507462
64= 0.7611940298507462
65= 0.7574626865671642
66= 0.753731343283582
67= 0.7574626865671642
68= 0.7611940298507462
69= 0.7574626865671642
```

```
#best value of max_depth when criterion=entropy
dt12=DecisionTreeClassifier(criterion="entropy",max_depth=50)
mymodel(dt12)
```

```
                  precision    recall  f1-score   support

               0       0.76      0.84      0.80       153
               1       0.75      0.65      0.70       115

        accuracy                           0.76       268
       macro avg       0.76      0.74      0.75       268
    weighted avg       0.76      0.76      0.75       268

    DecisionTreeClassifier(criterion='entropy', max_depth=50)
```

```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier



# instantiate the classifier

rfc = RandomForestClassifier(random_state=0)
```

```
# fit the model

rfc.fit(xtrain, ytrain)



# Predict the Test set results

ypred = rfc.predict(xtest)



# Check accuracy score

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'. format(accuracy_score(ytest, ypred)))
```

```
Model accuracy score with 10 decision-trees : 0.7612
```

```
from sklearn.ensemble import BaggingClassifier



bg=BaggingClassifier(LogisticRegression())
bg.fit(xtrain,ytrain)
ypred=bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                  precision    recall  f1-score   support

               0       0.78      0.85      0.82       153
               1       0.77      0.69      0.73       115

        accuracy                           0.78       268
       macro avg       0.78      0.77      0.77       268
    weighted avg       0.78      0.78      0.78       268
```

```
bg=BaggingClassifier(DecisionTreeClassifier())
bg.fit(xtrain,ytrain)
ypred=bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                  precision    recall  f1-score   support

               0       0.77      0.83      0.80       153
               1       0.75      0.67      0.71       115

        accuracy                           0.76       268
       macro avg       0.76      0.75      0.75       268
    weighted avg       0.76      0.76      0.76       268
```

```
from sklearn.ensemble import VotingClassifier



models=[]
accuracy=[]
models.append(("logistic regression",LogisticRegression()))
models.append(("Decision Tree",DecisionTreeClassifier()))


models
```

```
[('logistic regression', LogisticRegression()),
 ('Decision Tree', DecisionTreeClassifier())]
```

```
vc=VotingClassifier(estimators=models)
vc.fit(xtrain,ytrain)
ypred=vc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                   precision    recall  f1-score   support

                0       0.75      0.92      0.83       153
                1       0.85      0.59      0.70       115

         accuracy                           0.78       268
        macro avg       0.80      0.76      0.76       268
     weighted avg       0.79      0.78      0.77       268
```

```
# Boosting
```

```
from sklearn.ensemble import AdaBoostClassifier
```

```
ada=AdaBoostClassifier()
ada.fit(xtrain,ytrain)
ypred=ada.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                   precision    recall  f1-score   support

                0       0.78      0.86      0.82       153
                1       0.79      0.68      0.73       115

         accuracy                           0.78       268
        macro avg       0.78      0.77      0.77       268
     weighted avg       0.78      0.78      0.78       268
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gd=GradientBoostingClassifier()
gd.fit(xtrain,ytrain)
ypred=gd.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                   precision    recall  f1-score   support

                0       0.76      0.90      0.83       153
                1       0.83      0.63      0.71       115

         accuracy                           0.78       268
        macro avg       0.80      0.76      0.77       268
     weighted avg       0.79      0.78      0.78       268
```

```
!pip install xgboost
```

```
     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: xgboost in /usr/local/lib/python3.8/dist-packages (0.90)
     Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from xgboost) (1.21.6)
     Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from xgboost) (1.7.3)
```

```
from xgboost import XGBClassifier
```

```
xgb=XGBClassifier()
xgb.fit(xtrain,ytrain)
ypred=xgb.predict(xtest)
print(classification_report(ytest,ypred))
```

```
                   precision    recall  f1-score   support

                0       0.76      0.89      0.82       153
                1       0.81      0.63      0.71       115

         accuracy                           0.78       268
        macro avg       0.78      0.76      0.76       268
     weighted avg       0.78      0.78      0.77       268
```

```
#to build Kmeans clustering algo.
from sklearn.cluster import KMeans


# HEre we don't have domain knowledge hence we can not decide the value of K first.
#so we are using elbow method to decide the value of K.
```

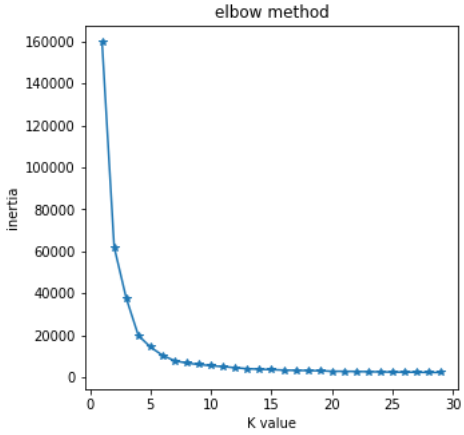
Elbow method

```
wcss=[]
for k in range(1,30):
    km=KMeans(n_clusters=k,init="k-means++",n_init=10,max_iter=300,random_state=1)
    km.fit(x)
    wcss.append(km.inertia_)


wcss
```

```
[159700.33583681268,
 62011.22425566829,
 37602.10299005371,
 19832.843062919033,
 14318.751284305654,
 10356.359086495722,
 7796.661861148099,
 6706.279314529209,
 6027.344367636138,
 5420.47716234644,
 5007.082214956865,
 4487.974594261294,
 4015.372098910241,
 3746.6794587917902,
 3572.0591849146335,
 3349.988360017239,
 3210.0218631791395,
 3145.1931965704966,
 3010.607264204694,
 2788.1071514477967,
 2750.948990271509,
 2686.666232582653,
 2571.2502901080907,
 2520.070042975459,
 2410.0171788691814,
 2403.482421726899,
 2351.3365342243605,
 2226.307559796983,
 2237.764747824095]
```

```
plt.figure(figsize=(5,5))
plt.title("elbow method")
plt.plot(range(1,30),wcss,marker="*")
plt.xlabel("K value")
plt.ylabel("inertia")
plt.show()
```




```
#final K value from elbow method is 5 , so we will build the final with k value as 5
```

```
km1=KMeans(n_clusters=5,init="k-means++",n_init=10,max_iter=300,random_state=1)
labels=km1.fit_predict(x)
```

```
#printing value of inertia when k=5
km1.inertia_
```

```
12570.984096718534
```

```
#printing values of centroid
km1.cluster_centers_
```

```
array([[30.63756614,  0.29100529],
       [56.9       ,  0.3       ],
       [ 5.00239437,  1.81690141],
       [22.        ,  0.5234375 ],
       [40.61377246,  0.34131737]])
```

```
labels
```

```
array([3, 4, 3, 0, 0, 3, 1, 2, 0, 3, 2, 1, 3, 4, 3, 1, 2, 0, 0, 3, 0, 0,
       3, 0, 2, 4, 3, 3, 3, 3, 4, 4, 3, 1, 0, 4, 3, 3, 3, 3, 4, 0, 3, 2,
       3, 3, 3, 3, 3, 3, 2, 3, 1, 0, 1, 4, 3, 0, 2, 2, 3, 4, 4, 2, 4, 3,
       0, 3, 3, 3, 0, 3, 3, 3, 0, 3, 3, 3, 2, 0, 3, 0, 3, 0, 3, 0, 3, 3,
       3, 3, 0, 3, 4, 3, 1, 3, 1, 3, 0, 0, 0, 3, 3, 0, 4, 0, 3, 3, 4, 3,
       4, 3, 3, 3, 3, 3, 1, 0, 3, 2, 3, 3, 0, 0, 1, 2, 3, 3, 3, 4, 0, 3,
       4, 0, 3, 3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 4, 4, 1, 3, 1, 4,
       3, 1, 3, 0, 3, 3, 4, 4, 3, 3, 2, 2, 4, 4, 4, 0, 1, 2, 2, 3, 1, 3,
       3, 1, 0, 4, 3, 0, 2, 2, 2, 4, 3, 4, 4, 4, 0, 3, 3, 2, 4, 1, 3, 4,
       3, 3, 0, 3, 0, 4, 3, 2, 0, 3, 3, 4, 3, 0, 3, 0, 3, 0, 0, 4, 0, 0,
       3, 0, 1, 3, 4, 3, 3, 3, 3, 3, 0, 0, 1, 2, 3, 3, 4, 2, 3, 0, 3, 3,
       0, 3, 0, 4, 3, 3, 4, 1, 3, 0, 1, 0, 4, 0, 4, 0, 0, 1, 3, 2, 1, 4,
       3, 4, 3, 3, 1, 0, 4, 3, 4, 4, 3, 1, 4, 0, 2, 0, 1, 0, 3, 3, 4, 0,
       0, 3, 4, 3, 3, 3, 4, 3, 3, 4, 3, 2, 4, 1, 3, 3, 3, 0, 3, 2, 4, 3,
       0, 0, 3, 3, 3, 0, 4, 3, 3, 1, 0, 4, 3, 0, 0, 3, 3, 4, 1, 4, 0, 3,
       3, 4, 4, 3, 4, 3, 0, 4, 4, 4, 2, 3, 0, 3, 4, 3, 4, 3, 2, 4, 3, 4,
       3, 3, 3, 0, 3, 4, 3, 3, 4, 0, 4, 0, 3, 0, 1, 3, 3, 3, 3, 3, 3, 3,
       2, 4, 3, 0, 3, 3, 4, 2, 0, 0, 3, 3, 2, 4, 3, 3, 4, 3, 0, 3, 3, 3,
       0, 4, 3, 0, 4, 3, 3, 0, 3, 0, 1, 2, 3, 3, 3, 3, 0, 0, 4, 3, 0, 3,
       0, 2, 3, 3, 0, 0, 3, 3, 0, 3, 3, 0, 0, 3, 4, 3, 1, 3, 3, 3, 1, 0,
       4, 3, 3, 0, 3, 2, 2, 0, 2, 1, 4, 3, 0, 1, 3, 0, 1, 4, 1, 3, 4, 0,
       4, 4, 3, 4, 0, 1, 3, 2, 3, 4, 0, 3, 3, 4, 0, 0, 3, 2, 2, 0, 1, 1,
       3, 3, 0, 1, 0, 2, 3, 3, 1, 1, 3, 3, 1, 3, 3, 3, 3, 3, 3, 4, 3, 3,
       0, 4, 0, 3, 0, 3, 4, 1, 3, 4, 0, 3, 4, 0, 0, 3, 3, 4, 3, 4, 1, 4,
       4, 3, 2, 3, 3, 3, 0, 2, 4, 0, 3, 3, 4, 2, 2, 0, 1, 1, 3, 0, 0, 2,
       3, 0, 3, 3, 3, 1, 4, 4, 4, 4, 3, 4, 0, 3, 3, 3, 3, 0, 3, 0, 1, 1,
       4, 3, 3, 3, 0, 4, 3, 0, 3, 4, 1, 4, 3, 3, 4, 1, 3, 3, 0, 1, 4, 3,
       4, 4, 0, 1, 3, 1, 3, 3, 4, 4, 0, 4, 0, 0, 3, 4, 4, 3, 3, 3, 0, 3,
       0, 3, 2, 3, 0, 4, 3, 3, 3, 1, 1, 3, 3, 3, 1, 1, 0, 4, 2, 0, 0, 0,
       4, 3, 3, 3, 2, 3, 2, 4, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 1,
       1, 4, 4, 4, 3, 0, 3, 3, 4, 4, 4, 0, 1, 0, 0, 3, 3, 3, 4, 4, 3, 0,
       3, 3, 1, 3, 3, 3, 3, 3, 0, 2, 3, 3, 1, 1, 4, 3, 1, 4, 3, 0, 3, 3,
       3, 4, 4, 4, 3, 3, 3, 4, 4, 0, 1, 3, 4, 0, 3, 0, 2, 3, 0, 1, 0, 3,
       0, 3, 3, 3, 0, 2, 0, 3, 3, 0, 4, 0, 3, 3, 4, 4, 3, 3, 0, 1, 3, 0,
       3, 0, 2, 2, 0, 3, 4, 2, 0, 3, 0, 0, 3, 4, 3, 4, 3, 1, 4, 0, 3, 0,
       3, 4, 1, 3, 1, 3, 3, 2, 3, 4, 2, 3, 0, 3, 3, 3, 3, 2, 2, 4, 3, 3,
       3, 4, 3, 4, 1, 0, 0, 0, 0, 0, 2, 2, 0, 0, 4, 3, 4, 0, 3, 4, 0, 2,
       0, 4, 3, 0, 4, 2, 1, 0, 4, 0, 2, 3, 3, 2, 3, 1, 3, 2, 3, 3, 3, 4,
       3, 3, 0, 4, 3, 3, 0, 0, 3, 4, 3, 0, 0, 4, 2, 1, 2, 3, 4, 3, 4, 1,
       3, 3, 4, 3, 4, 3, 3, 4, 0, 0, 3, 2, 3, 4, 0, 4, 0, 3, 3, 3, 3, 1,
       3, 0, 3, 0, 3, 4, 0, 3, 3, 3, 0], dtype=int32)
```

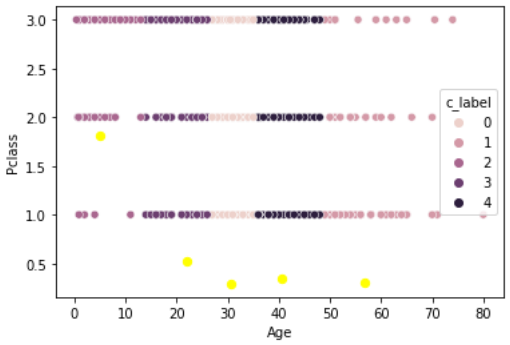
```
df["c_label"]=labels
```

```
df
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked	c_label
0	0	3	male	22.0	1	0	S	3
1	1	1	female	38.0	1	0	C	4
2	1	3	female	26.0	0	0	S	3
3	1	1	female	35.0	1	0	S	0
4	0	3	male	35.0	0	0	S	0

```
centroid_df=pd.DataFrame(km1.cluster_centers_,columns=["x","y"])


sns.scatterplot(data=df,x=df["Age"],y="Pclass",hue=df["c_label"])
plt.scatter(centroid_df["x"],centroid_df["y"],s=40,color='yellow')
plt.show()
```



✓ 0s completed at 20:06