In [161]:
```
import pandas as pd
import sklearn
```

Read the csv file using pandas object

In [162]:
```
TravelDF_Original = pd.read_csv('F:\\titanic.csv', header=0)
```

Check on the schema

In [278]:
```
TravelDF_Original.dtypes
```

Out[278]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Verify if data loaded correct

we will only focus on age, pclass, gender, and embarked to predict survival status

In [164]:
```
TravelDF = TravelDF_Original[['Survived','Pclass','Age','Sex','Embarked']].cop
y()
TravelDF.head()
```

Out[164]:

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

Check for null values in dataframe

In [165]:
```
TravelDF.isnull().any()
```

Out[165]:
```
Survived    False
Pclass      False
Age          True
Sex         False
Embarked    False
dtype: bool
```

There were any nulls found for Age and Cabin attribute ,so replace the missing age values with average age

Find average mean value

In [166]:
```
Age_mean=TravelDF['Age'].mean()
print(Age_mean)
```

```
29.69911764705882
```

In [167]:
```
TravelDF = TravelDF.fillna(Age_mean)
```

Verify if the nulls got replaced for age attribute

In [168]:
```
TravelDF.isnull().any()
```

Out[168]:
```
Survived    False
Pclass      False
Age         False
Sex         False
Embarked    False
dtype: bool
```

Assignment questions

1. What is the shape of the data contained in titanic.csv?

In [169]:
```
TravelDF.shape
```

Out[169]:
```
(891, 5)
```

2. What features (or attributes) are recorded for each passenger in titanic.csv?

In [170]:
```
TravelDF.columns
```

Out[170]:
```
Index(['Survived', 'Pclass', 'Age', 'Sex', 'Embarked'], dtype='object')
```

3. Provide a schema of the columns to be included in your model for this assignment. Comment on columns that may require transformation(s). An example of transformation is that of creating dummy variables. List these columns and explain why and what transformation is required. Include these comments in your notebook.

In [171]:
```
TravelDF.dtypes
TravelDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 5 columns):
Survived    891 non-null int64
Pclass      891 non-null int64
Age         891 non-null float64
Sex         891 non-null object
Embarked    891 non-null object
dtypes: float64(1), int64(2), object(2)
memory usage: 34.9+ KB
```

3 columns need transformation: 'Pclass','Sex',and 'Embarked'.since 'sex' and 'embarked' are both categorical variables and 'Pclass' is nominal variable,they need to be converted to numeric values to be used in regression model. This is because machine learning models only consume data that are numeric.

For column "Pclass", there are 3 classes - 1st, 2nd and 3rd. hence we will create 2 dummy variables. Each Class will be assigned a 1 or 0 depending on whether or not that class is present.

For column "Sex", there are 2 categories - male and female, hence we will create 1 dummy variable. Each Category will be assigned a 1 or 0 depending on whether or not that category is present.

For column "Embarked", there are 3 categories - Southampton, Queenstown, and Cherbourg, hence we will create 2 dummy variables. Each Category will be assigned a 1 or 0 depending on whether or not that category is present.

4. Comment on the balance of data in titanic.csv with regards to each input variable as well as your target variable. Support your comments with appropriate statistics.

In [172]:
```
TravelDF.describe()
```

Out[172]:

	Survived	Pclass	Age
count	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118
std	0.486592	0.836071	13.002015
min	0.000000	1.000000	0.420000
25%	0.000000	2.000000	22.000000
50%	0.000000	3.000000	29.699118
75%	1.000000	3.000000	35.000000
max	1.000000	3.000000	80.000000

In [173]:
```
TravelDF.groupby('Survived').size()
```

Out[173]:
```
Survived
0    549
1    342
dtype: int64
```

survival status as 'No' has maximum number of people. This means more than half of the dataset contain records for those who didnot survive.

In [174]:
```
TravelDF.groupby('Pclass').size()
```

Out[174]:
```
Pclass
1    216
2    184
3    491
dtype: int64
```

Passenger class is defined as 1 for first class, or suite, 2 for a second class, and 3 for third class.Most passengers were in third class, followed by first class/suite, followed by second class

In [175]:
```
TravelDF.groupby('Sex').size()
```

Out[175]:
```
Sex
female    314
male      577
dtype: int64
```

Male passengers are more in number than female passengers

In [176]:
```
TravelDF.groupby('Embarked').size()
```

Out[176]:
```
Embarked
C    168
Q     78
S    645
dtype: int64
```

embarked(port of embarkation – S for Southampton, Q for Queenstown, and C for Cherbourg).More than half of dataset shows that passengers embarked from Southampton,followed by Cherbourg, followed by Queenstown.

In [177]:
```
TravelDF.groupby('Survived').mean()
```

Out[177]:

	Pclass	Age
Survived		
0	2.531876	30.415100
1	1.950292	28.549778

age and passenger class is not determining since its about same for both status and doesn't make a significant difference on survival status.

In [178]:
```
TravelDF.groupby('Sex').mean()
```

Out[178]:

	Survived	Pclass	Age
Sex			
female	0.742038	2.159236	28.216730
male	0.188908	2.389948	30.505824

average number of female passengers who survived is higher than male passengers.age and passenger class is not determining since its about same for both status and doesn't make a significant difference on gender.

In [179]:
```
TravelDF.groupby('Pclass').mean()
```

Out[179]:

Pclass	Survived	Age
1	0.629630	37.048118
2	0.472826	29.866958
3	0.242363	26.403259

There is strong connection between survival rate and passenger class.On average - passengers in first class have higher survival rate, followed by second class, followed by third class with least number.Average age of passengers in first class is close to 37 years, followed by second class with age close to 29 years and followed by third class with age close to 26 years.

In [228]:
```
dummy_sex_df = pd.get_dummies(TravelDF["Sex"], prefix="Sex", drop_first=True)
dummy_sex_df.head()
```

Out[228]:

	Sex_male
0	1
1	0
2	0
3	0
4	1

Sex input categorical variable is converted to numeric dummy variable where the 1 is for male and 0 is for female.

In [261]:
```
dummy_embarked_df = pd.get_dummies(TravelDF["Embarked"], prefix="Emb",drop_fir
st=True)
dummy_embarked_df.head()
```

Out[261]:

	Emb_Q	Emb_S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

Embarked input categorical variable is converted to numeric dummy variables for port of embarkation – s for Southampton, Q for Queenstown, and C for Cherbourg

In [263]:
```
dummy_pclass_df=pd.get_dummies(TravelDF["Pclass"], prefix="Pclass", drop_first
=True)
dummy_pclass_df.head()
```

Out[263]:

	Pclass_2	Pclass_3
0	0	1
1	0	0
2	0	1
3	0	0
4	0	1

Pclass input categorical variable is converted to numeric dummy variables for passenger classes 1,2,3

In [264]:
```
TravelFinalDF=pd.concat([TravelDF,dummy_sex_df,dummy_embarked_df,dummy_pclass_
df], axis=1)
TravelFinalDF.head()
```

Out[264]:

	Survived	Pclass	Age	Sex	Embarked	Sex_male	Emb_Q	Emb_S	Pclass_2	Pclass_3
0	0	3	22.0	male	S	1	0	1	0	1
1	1	1	38.0	female	C	0	0	0	0	0
2	1	3	26.0	female	S	0	0	1	0	1
3	1	1	35.0	female	S	0	0	1	0	0
4	0	3	35.0	male	S	1	0	1	0	1

In [265]:
```
TravelFinalDF=TravelFinalDF.drop(['Sex','Embarked','Pclass'],axis=1)
TravelFinalDF.head()
```

Out[265]:

	Survived	Age	Sex_male	Emb_Q	Emb_S	Pclass_2	Pclass_3
0	0	22.0	1	0	1	0	1
1	1	38.0	0	0	0	0	0
2	1	26.0	0	0	1	0	1
3	1	35.0	0	0	1	0	0
4	0	35.0	1	0	1	0	1

In [266]:
```
TravelFinalDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
Survived    891 non-null int64
Age         891 non-null float64
Sex_male    891 non-null uint8
Emb_Q       891 non-null uint8
Emb_S       891 non-null uint8
Pclass_2    891 non-null uint8
Pclass_3    891 non-null uint8
dtypes: float64(1), int64(1), uint8(5)
memory usage: 18.4 KB
```

5. Perform the transformations, if any, identified in step # 3. Perform feature engineering if and where needed, including Vectorization of relevant input variables. Provide a printout of the schema of your feature-engineered data.

Logistics Regression Analysis

In [267]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In SKLearn, you do not need to create a feature vector. However, because the split into training and test datasets requires the values of feature (X) and target variable (Y) extracted into their own dataframes or arrays, we will extract the features into its own dataframe and label/target variable into its own dataframe. Then, we will split each into training and test

In [268]:
```
TravelTargetDF=TravelFinalDF[['Survived']]
TravelInputDF=TravelFinalDF.drop('Survived', axis = 1)
```

In [269]:
```
TravelInputDF.head()
```

Out[269]:

	Age	Sex_male	Emb_Q	Emb_S	Pclass_2	Pclass_3
0	22.0	1	0	1	0	1
1	38.0	0	0	0	0	0
2	26.0	0	0	1	0	1
3	35.0	0	0	1	0	0
4	35.0	1	0	1	0	1

In [270]:
```
TravelTargetDF.head()
```

Out[270]:

	Survived
0	0
1	1
2	1
3	1
4	0

In [271]:
```
TravelInputDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
Age          891 non-null float64
Sex_male     891 non-null uint8
Emb_Q        891 non-null uint8
Emb_S        891 non-null uint8
Pclass_2     891 non-null uint8
Pclass_3     891 non-null uint8
dtypes: float64(1), uint8(5)
memory usage: 11.4 KB
```

In [272]:
```
TravelTargetDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 1 columns):
Survived     891 non-null int64
dtypes: int64(1)
memory usage: 7.1 KB
```

6. To train and then test your model, split the data from titanic.csv into training and test datasets using an 75/25 split. Like you did in step 4 above, comment on the balance of data in the training and test datasets. Are they representative of the overall data? What can you say about the balance in target classes in both the training and test datasets?

In [273]:
```
X_train, X_test, Y_train, Y_test = train_test_split(TravelInputDF,TravelTarget
DF, test_size=0.25, random_state=101)
```

In [274]:
```
X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
```

Out[274]:
```
((668, 6), (668, 1), (223, 6), (223, 1))
```

In [275]:
```
print(X_train.head())
print()
print(Y_train.head())
```

```
           Age  Sex_male  Emb_Q  Emb_S  Pclass_2  Pclass_3
180  29.699118         0      0      1         0         1
126  29.699118         1      1      0         0         1
132  47.000000         0      0      1         0         1
304  29.699118         1      0      1         0         1
563  29.699118         1      0      1         0         1

     Survived
180         0
126         0
132         0
304         0
563         0
```

In [276]:
```
Y_train.groupby('Survived').size()
```

Out[276]:
```
Survived
0    422
1    246
dtype: int64
```

In [277]:
```
Y_test.groupby('Survived').size()
```

Out[277]:
```
Survived
0    127
1     96
dtype: int64
```

Yes they represent 75% of total 891 records as train 668 and 25% of 891 records as test 223 records.

The balance of data in both test and train sets in not equal the percentage of not survived is more than survived.

For train data set, 63.17% of the data represent not survived and 36.8% of the data represent survived people. For test data set,56.9% of the data represent not survived and 43.04% of the data represent survived people. Where as in our original dat set 61.6% represent not survived and 38.8% of data repesnt survived people.

7. Build and train the Logistic Regression model using SKLearn library. Generate a list of predictions for passenger's survival status (survival = 1) based on the trained model. Display actual, predicted, and probability values for the first 10 rows only. Based on these results, comment on the performance of the model? Is the model predicting likelihood of survival with high probability?

In [245]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,Y_train)
```

```
C:\Users\Deepthi\Anaconda3-3.7\lib\site-packages\sklearn\linear_model\logisti
c.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. S
pecify a solver to silence this warning.
  FutureWarning)
C:\Users\Deepthi\Anaconda3-3.7\lib\site-packages\sklearn\utils\validation.py:
724: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using ra
vel().
  y = column_or_1d(y, warn=True)
```

Out[245]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [246]:
```
logmodel = LogisticRegression(solver='liblinear')
logmodel.fit(X_train,Y_train)
```

```
C:\Users\Deepthi\Anaconda3-3.7\lib\site-packages\sklearn\utils\validation.py:
724: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using ra
vel().
  y = column_or_1d(y, warn=True)
```

Out[246]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose
=0,
                   warm_start=False)
```

Generate predictions to evaluate the trained model using X_Test data. Predicted Y's for first 100 records

In [247]:
```
Y_predict = logmodel.predict(X_test)
```

In [248]:
```
predict_y = Y_predict.reshape(-1,1)
print(predict_y.shape)
```

```
(223, 1)
```

In [249]:
```
test_y = (Y_test.values).reshape(-1,1)
print(test_y.shape)
print(test_y.size)
```

```
(223, 1)
223
```

In [250]:
```
predicted_probs = logmodel.predict_proba(X_test)
print(predicted_probs)
```

```
[[0.72488488 0.27511512]
 [0.05067854 0.94932146]
 [0.53212054 0.46787946]
 [0.85885737 0.14114263]
 [0.77213738 0.22786262]
 [0.95013035 0.04986965]
 [0.61487886 0.38512114]
 [0.90995623 0.09004377]
 [0.12091362 0.87908638]
 [0.38946252 0.61053748]
 [0.77213738 0.22786262]
 [0.93277656 0.06722344]
 [0.22216873 0.77783127]
 [0.91501839 0.08498161]
 [0.88721675 0.11278325]
 [0.92401068 0.07598932]
 [0.44400178 0.55599822]
 [0.89609591 0.10390409]
 [0.87885906 0.12114094]
 [0.18613085 0.81386915]
 [0.93850063 0.06149937]
 [0.86999842 0.13000158]
 [0.38946252 0.61053748]
 [0.38946252 0.61053748]
 [0.95712759 0.04287241]
 [0.80667304 0.19332696]
 [0.8931067  0.1068933 ]
 [0.65083888 0.34916112]
 [0.84277402 0.15722598]
 [0.93473749 0.06526251]
 [0.21134649 0.78865351]
 [0.52301035 0.47698965]
 [0.38946252 0.61053748]
 [0.93277656 0.06722344]
 [0.76626313 0.23373687]
 [0.0866914  0.9133086 ]
 [0.92868978 0.07131022]
 [0.95305049 0.04694951]
 [0.7547153  0.2452847 ]
 [0.5944262  0.4055738 ]
 [0.37512902 0.62487098]
 [0.38946252 0.61053748]
 [0.12794971 0.87205029]
 [0.60647117 0.39352883]
 [0.81984498 0.18015502]
 [0.61487886 0.38512114]
 [0.92437469 0.07562531]
 [0.7217123  0.2782877 ]
 [0.94030527 0.05969473]
 [0.77213738 0.22786262]
 [0.19592808 0.80407192]
 [0.66501821 0.33498179]
 [0.25069706 0.74930294]
 [0.3216606  0.6783394 ]
 [0.38946252 0.61053748]
 [0.92805545 0.07194455]
 [0.85037875 0.14962125]
```

```
[0.89609591 0.10390409]
[0.67396917 0.32603083]
[0.071802   0.928198  ]
[0.09727272 0.90272728]
[0.8543248  0.1456752 ]
[0.96204008 0.03795992]
[0.48913656 0.51086344]
[0.9486065  0.0513935 ]
[0.7387985  0.2612015 ]
[0.14469761 0.85530239]
[0.61487886 0.38512114]
[0.38461523 0.61538477]
[0.61487886 0.38512114]
[0.36569521 0.63430479]
[0.87885906 0.12114094]
[0.86208159 0.13791841]
[0.90185323 0.09814677]
[0.52873353 0.47126647]
[0.132486   0.867514  ]
[0.9366451  0.0633549 ]
[0.91745149 0.08254851]
[0.87296729 0.12703271]
[0.77526967 0.22473033]
[0.92805545 0.07194455]
[0.071802   0.928198  ]
[0.88101761 0.11898239]
[0.92656141 0.07343859]
[0.24186999 0.75813001]
[0.90995623 0.09004377]
[0.83346836 0.16653164]
[0.43413704 0.56586296]
[0.8956436  0.1043564 ]
[0.39551679 0.60448321]
[0.88101761 0.11898239]
[0.50498684 0.49501316]
[0.07945084 0.92054916]
[0.14320883 0.85679117]
[0.48913656 0.51086344]
[0.21810004 0.78189996]
[0.30909285 0.69090715]
[0.89901098 0.10098902]
[0.87885906 0.12114094]
[0.85099461 0.14900539]
[0.92868978 0.07131022]
[0.21274176 0.78725824]
[0.68699303 0.31300697]
[0.91982103 0.08017897]
[0.42840906 0.57159094]
[0.45753312 0.54246688]
[0.13097629 0.86902371]
[0.10852147 0.89147853]
[0.28809129 0.71190871]
[0.61487886 0.38512114]
[0.4262678  0.5737322 ]
[0.64662205 0.35337795]
[0.75240727 0.24759273]
[0.90732453 0.09267547]
```

```
[0.51291021 0.48708979]
[0.59878366 0.40121634]
[0.11445445 0.88554555]
[0.81984498 0.18015502]
[0.66510709 0.33489291]
[0.38946252 0.61053748]
[0.62459803 0.37540197]
[0.95161133 0.04838867]
[0.12091362 0.87908638]
[0.90732453 0.09267547]
[0.61487886 0.38512114]
[0.92212833 0.07787167]
[0.52635596 0.47364404]
[0.91745149 0.08254851]
[0.09075014 0.90924986]
[0.76402978 0.23597022]
[0.11857734 0.88142266]
[0.3297644  0.6702356 ]
[0.071802   0.928198  ]
[0.91249898 0.08750102]
[0.92437469 0.07562531]
[0.90598294 0.09401706]
[0.9146135  0.0853865 ]
[0.95013035 0.04986965]
[0.92805545 0.07194455]
[0.25226902 0.74773098]
[0.93473749 0.06526251]
[0.46541196 0.53458804]
[0.90462397 0.09537603]
[0.91982103 0.08017897]
[0.52635596 0.47364404]
[0.92437469 0.07562531]
[0.05222575 0.94777425]
[0.51509595 0.48490405]
[0.94030527 0.05969473]
[0.84893888 0.15106112]
[0.18137581 0.81862419]
[0.38461523 0.61538477]
[0.63087366 0.36912634]
[0.92805545 0.07194455]
[0.88101761 0.11898239]
[0.61598903 0.38401097]
[0.43403903 0.56596097]
[0.60204614 0.39795386]
[0.11396394 0.88603606]
[0.07681818 0.92318182]
[0.74487017 0.25512983]
[0.88101761 0.11898239]
[0.52635596 0.47364404]
[0.91252041 0.08747959]
[0.42840906 0.57159094]
[0.33680911 0.66319089]
[0.92805545 0.07194455]
[0.10899144 0.89100856]
[0.7625209  0.2374791 ]
[0.70707323 0.29292677]
[0.1399451  0.8600549 ]
```

```
 [0.89901098 0.10098902]
 [0.47311132 0.52688868]
 [0.1399451  0.8600549 ]
 [0.92805545 0.07194455]
 [0.92805545 0.07194455]
 [0.15116643 0.84883357]
 [0.95840986 0.04159014]
 [0.77213738 0.22786262]
 [0.7326342  0.2673658 ]
 [0.75826611 0.24173389]
 [0.92656141 0.07343859]
 [0.09684779 0.90315221]
 [0.12103999 0.87896001]
 [0.38946252 0.61053748]
 [0.94376668 0.05623332]
 [0.92805545 0.07194455]
 [0.52873353 0.47126647]
 [0.96427981 0.03572019]
 [0.92656141 0.07343859]
 [0.9366451  0.0633549 ]
 [0.91501839 0.08498161]
 [0.78471662 0.21528338]
 [0.21667929 0.78332071]
 [0.60088392 0.39911608]
 [0.95444891 0.04555109]
 [0.92805545 0.07194455]
 [0.44967546 0.55032454]
 [0.7217123  0.2782877 ]
 [0.96204008 0.03795992]
 [0.53212054 0.46787946]
 [0.55451337 0.44548663]
 [0.92805545 0.07194455]
 [0.0989098  0.9010902 ]
 [0.93473749 0.06526251]
 [0.76156935 0.23843065]
 [0.90732453 0.09267547]
 [0.51291021 0.48708979]
 [0.29470924 0.70529076]
 [0.97206571 0.02793429]
 [0.67396917 0.32603083]
 [0.70046378 0.29953622]
 [0.90185323 0.09814677]
 [0.879871   0.120129  ]
 [0.64350862 0.35649138]
 [0.82568549 0.17431451]
 [0.79523522 0.20476478]
 [0.08421386 0.91578614]
 [0.92805545 0.07194455]
 [0.38461523 0.61538477]
 [0.32621125 0.67378875]
 [0.57701566 0.42298434]
 [0.61487886 0.38512114]]
```

In [251]:
```
import numpy as np
np.set_printoptions(suppress=True)

prob_results_df = pd.DataFrame(predicted_probs)
prob_results_df["Predicted"] = predict_y
prob_results_df["Actual"] = test_y
prob_results_df.head(15)
```

Out[251]:

	0	1	Predicted	Actual
0	0.724885	0.275115	0	0
1	0.050679	0.949321	1	1
2	0.532121	0.467879	0	0
3	0.858857	0.141143	0	1
4	0.772137	0.227863	0	0
5	0.950130	0.049870	0	0
6	0.614879	0.385121	0	1
7	0.909956	0.090044	0	0
8	0.120914	0.879086	1	1
9	0.389463	0.610537	1	1
10	0.772137	0.227863	0	0
11	0.932777	0.067223	0	0
12	0.222169	0.777831	1	1
13	0.915018	0.084982	0	1
14	0.887217	0.112783	0	1

In [252]:
```
prob_results_df.groupby("Predicted").size()
```

Out[252]:
```
Predicted
0    150
1     73
dtype: int64
```

In [254]:
```
prob_results_df.groupby("Actual").size()
```

Out[254]:
```
Actual
0    127
1     96
dtype: int64
```

The performance of the test model is good as ,the test model is predicting survival status with 67.2% for not survived and 32.7% for survived.Where as the actual model has 56.95% for not survived and 43.04% for survived hence the values are close enough for us to say that the performance is good.

8. Using the test data from the 75/25 split, evaluate the performance of your trained model. Compute and show the values for Confusion Matrix, Accuracy, Recall, Precision, and an F1 score. Comment of general usefulness of the model in predicting the survival status of passengers given their age, gender, pclass and embarked values.

In [255]:
```
from sklearn.metrics import classification_report
from sklearn import metrics

print(classification_report(Y_test,Y_predict))
```

```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83       127
           1       0.84      0.64      0.72        96

    accuracy                           0.79       223
   macro avg       0.80      0.77      0.78       223
weighted avg       0.80      0.79      0.78       223
```

In [256]:
```
conf_matrix = metrics.confusion_matrix(Y_test, Y_predict)
conf_matrix
```

Out[256]:
```
array([[115,  12],
       [ 35,  61]], dtype=int64)
```

True Positives(115) and True Negatives(61) are larger numbers as compared to False Positives(12) and False Negatives (35). Thus, the trained model is doing a good job of classifiying the users correctly into the two survived categories 0 and 1.

In [257]:
```
print("Accuracy:",metrics.accuracy_score(Y_test, Y_predict))
print("Precision:",metrics.precision_score(Y_test, Y_predict))
print("Recall:",metrics.recall_score(Y_test, Y_predict))
```

```
Accuracy: 0.7892376681614349
Precision: 0.8356164383561644
Recall: 0.6354166666666666
```

The model is good in predicting survival status with and accuracy of 78.9% ,precision rate 83.5% and Recall rate 63.5%

9. One of the most important performance measure metric is ROC (Receiver Operating Characteristics) curve and its associated Area Under the Curve (AUC). ROC is a probability curve while AUC represents degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between passengers that survived and those that did not. Calculate the AUC for the titanic data and comment on the worthiness of the trained model to predict survival status.

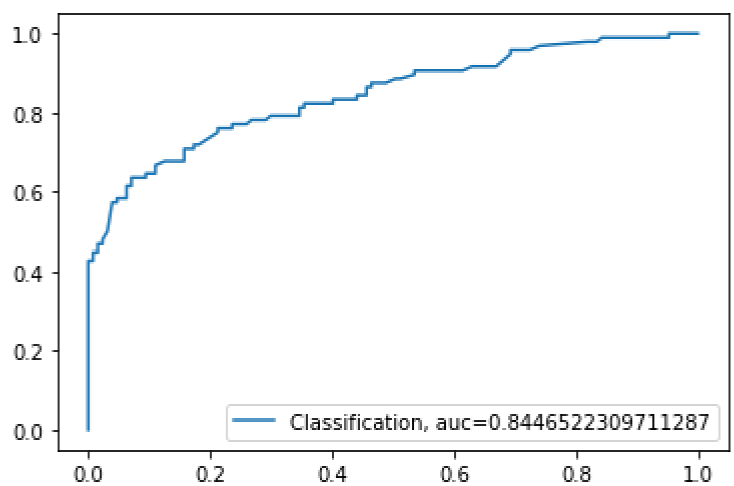
Generating and displaying ROC curve

In [258]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

y_pred_prob = logmodel.predict_proba(X_test)[::,1]

fpr, tpr, _ = metrics.roc_curve(Y_test,  y_pred_prob)

auc = metrics.roc_auc_score(Y_test, y_pred_prob)
plt.plot(fpr,tpr,label="Classification, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



AUC is 84.46% and sinc eit ia high AUC the trained model is good to predict the survival status

10. Please discuss the performance of the trained model in predicting survival status using the values you computed for the AUC, Confusion Matrix, Accuracy, Recall, Precision and F1 score. You answer must be supported by values of these evaluation metrics.

Higher the AUC, better the model is at distinguishing between passengers that survived and those that did not. Based on AUC, model can distinguish between classes(survived or not) around 84.46% Based on Accuracy, Recall, Precision and F1 score calculated for trained model, model can predict the probability of a passenger's survival status at around 78.9% accuracy,83.5% precision and 63.5% F1 Score. Model is useful at predicting probability of passengers survival status.

In [ ]: