sklearn2-2

April 23, 2024

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
train=pd.read_csv('titanic_train.csv')
```

[3]:
```
len(train)
```

[3]: 891

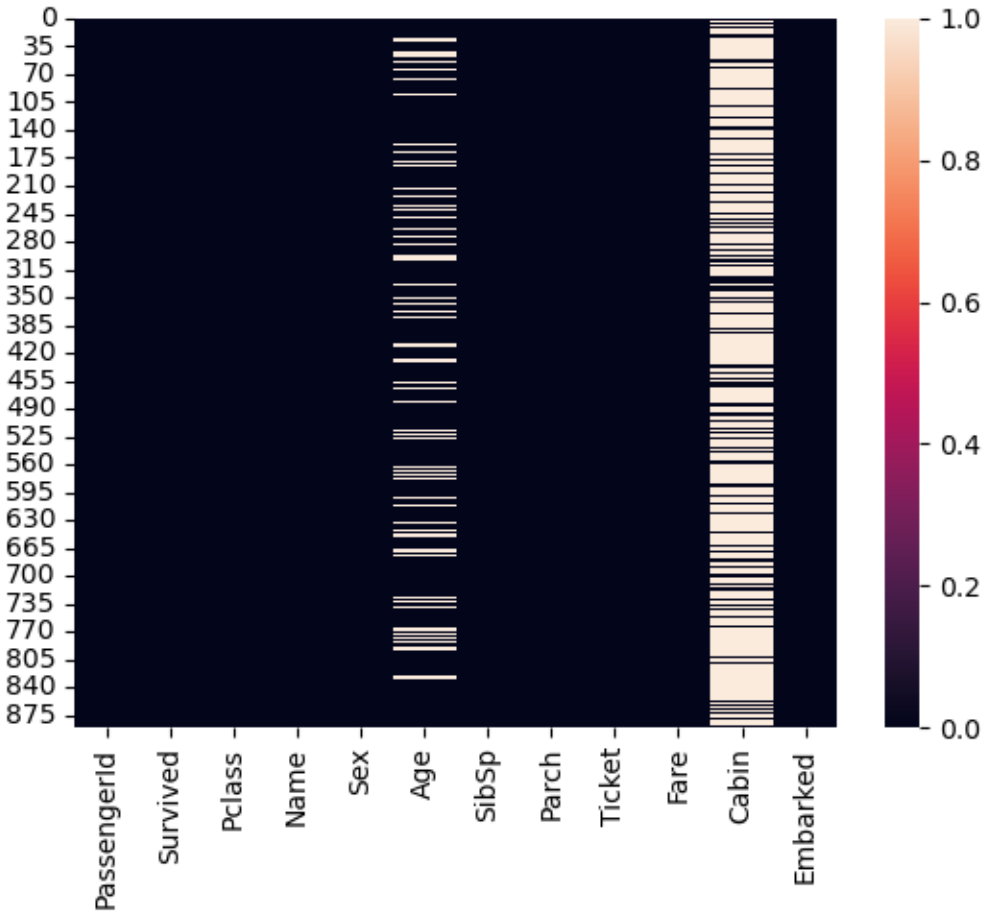
[4]:
```
train.isnull
```

[4]:
```
<bound method DataFrame.isnull of      PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …       …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                              Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0
..                                                 …     …   …    …
886                            Montvila, Rev. Juozas    male  27.0      0
887                     Graham, Miss. Margaret Edith  female  19.0      0
888         Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                            Behr, Mr. Karl Howell    male  26.0      0
```

```
890                              Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     …               …       …   …      …
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]>
```

[5]:
```
sns.heatmap(train.isnull())
```

[5]: <Axes: >



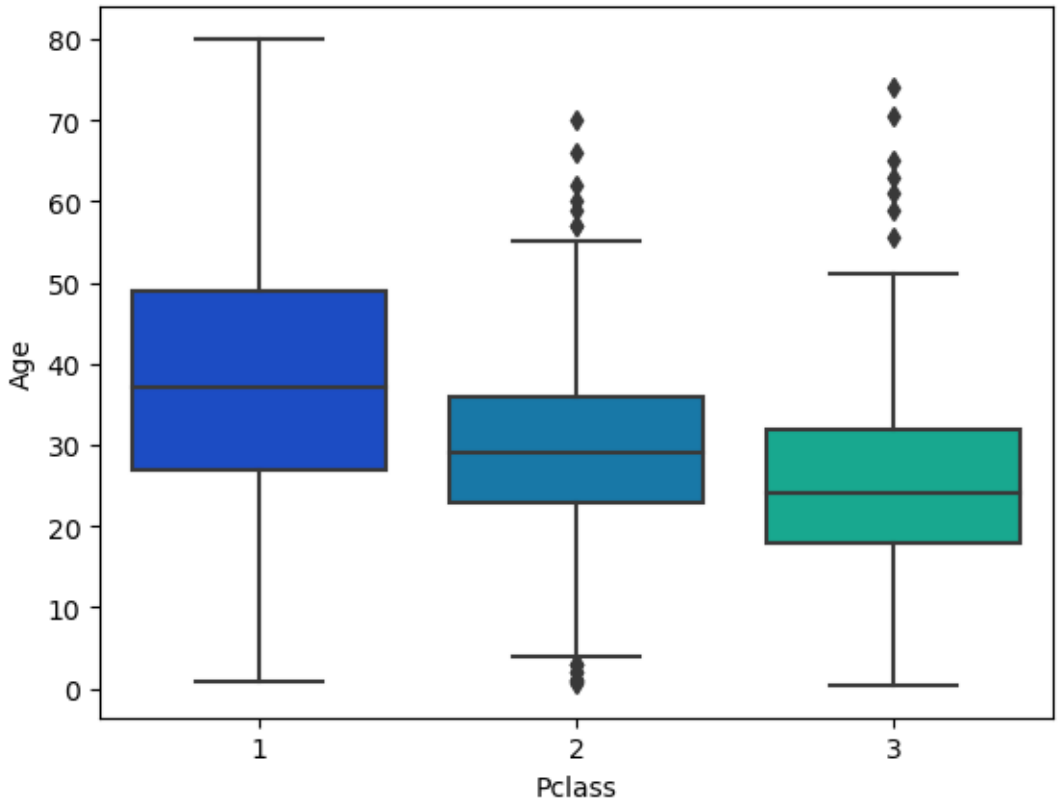
[6]:
```
train.describe()
```

[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[7]:
```
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

[7]:
```
<Axes: xlabel='Pclass', ylabel='Age'>
```



[8]:
```
def impute_age(cols):
    Age = cols[0]
    pclass = cols[1]

    if pd.isnull(Age):

        if pclass ==1:
            return 37
        elif pclass == 2:
            return 29

        else:
            return 24

    else:
            return Age
```

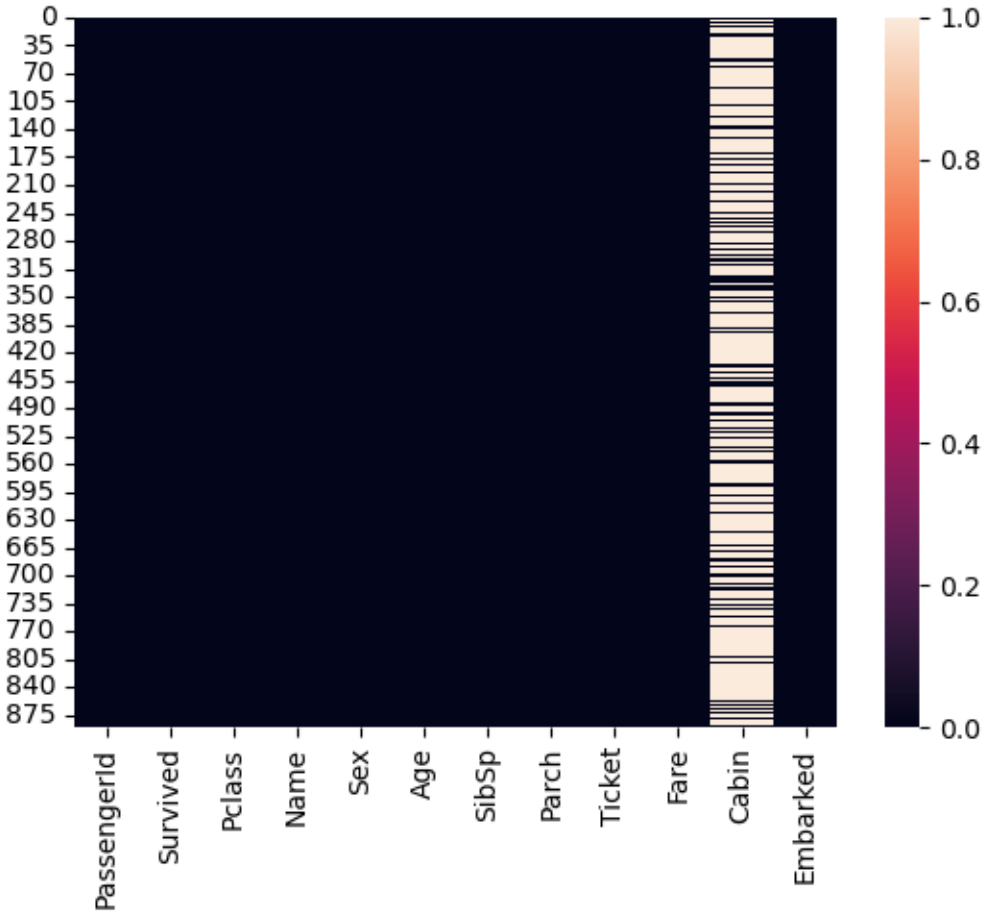
[9]:
```
train['Age'].fillna(method='ffill')
train['Age']=train['Age'].fillna(method='ffill')
```

[10]:
```
train["Age"].describe()
```

[10]:
```
count    891.00000
mean      29.58156
std       14.55459
min        0.42000
25%       20.00000
50%       28.00000
75%       38.00000
max       80.00000
Name: Age, dtype: float64
```

[11]:
```
sns.heatmap(train.isnull())
```

[11]:
```
<Axes: >
```



[12]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

[13]:
```
train.drop(['Sex', 'Embarked', 'Name','Ticket','Cabin'],axis=1,inplace=True)
```

[14]:
```
train= pd.concat([train, sex, embark],axis=1)
```

[15]:
```
train.head()
```

[15]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	True	False	True
1	2	1	1	38.0	1	0	71.2833	False	False	False
2	3	1	3	26.0	0	0	7.9250	False	False	True
3	4	1	1	35.0	1	0	53.1000	False	False	True
4	5	0	3	35.0	0	0	8.0500	True	False	True

[16]:
```
from sklearn.model_selection import train_test_split
```

[17]:
```
x_train,x_test,y_train,y_test=train_test_split(train.
↪drop('Survived',axis=1),train['Survived'],test_size=0.30,random_state=101)
```

[18]:
```
from sklearn.linear_model import LogisticRegression
```

[19]:
```
logmodel =LogisticRegression()
```

[20]:
```
logmodel.fit(x_train,y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[20]:
```
LogisticRegression()
```

[21]:
```
predict=logmodel.predict(x_test)
predict
```

[21]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1], dtype=int64)
```

1 Decision Trees

We'll start just by training a single decision tree

[22]:
```
from sklearn.tree import DecisionTreeClassifier
```

[23]:
```
dtree=DecisionTreeClassifier()
```

[24]:
```
dtree.fit(x_train,y_train)
```

[24]:
```
DecisionTreeClassifier()
```

2 Training and Predicting

[25]:
```
predict_tree=dtree.predict(x_test)
```

[26]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[27]:
```
print(classification_report(y_test,predict_tree))
```

```
              precision    recall  f1-score   support

           0       0.76      0.79      0.78       154
           1       0.70      0.67      0.68       114

    accuracy                           0.74       268
   macro avg       0.73      0.73      0.73       268
weighted avg       0.74      0.74      0.74       268
```

[28]:
```
print(confusion_matrix(y_test,predict_tree))
```

```
[[122  32]
 [ 38  76]]
```

3 Random Forests

Now let's compare the decision tree model to a random forest

[29]:
```
from sklearn.ensemble import RandomForestClassifier
```

[30]:
```
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(x_train,y_train)
```

[30]:
```
RandomForestClassifier()
```

[31]:
```
rfc_pred=rfc.predict(x_test)
```

[32]:
```
print(confusion_matrix(y_test,rfc_pred))
```

```
[[141  13]
 [ 34  80]]
```

[33]:
```
print(classification_report(y_test,rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86       154
           1       0.86      0.70      0.77       114

    accuracy                           0.82       268
   macro avg       0.83      0.81      0.82       268
weighted avg       0.83      0.82      0.82       268
```

[34]:
```
from sklearn.naive_bayes import GaussianNB
```

[35]:
```
naive_model=GaussianNB()
```

[36]:
```
naive_model.fit(x_train,y_train)
```

[36]:
```
GaussianNB()
```

[37]:
```
from sklearn.linear_model import LogisticRegression
```

[38]:
```
logmodel =LogisticRegression()
```

[39]:
```
logmodel.fit(x_train,y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[39]:
```
LogisticRegression()
```

[40]:
```
predictions=logmodel.predict(x_test)
```

[41]:
```
predictions
```

[41]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1], dtype=int64)
```

[42]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[43]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       154
           1       0.79      0.63      0.70       114

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

[44]:
```
train.head()
```

[44]:
```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare   male      Q  \
0            1         0       3  22.0      1      0   7.2500   True  False
1            2         1       1  38.0      1      0  71.2833  False  False
2            3         1       3  26.0      0      0   7.9250  False  False
3            4         1       1  35.0      1      0  53.1000  False  False
4            5         0       3  35.0      0      0   8.0500   True  False

       S
0   True
1  False
2   True
3   True
4   True
```

[45]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Age          891 non-null    float64
 4   SibSp        891 non-null    int64
 5   Parch        891 non-null    int64
 6   Fare         891 non-null    float64
 7   male         891 non-null    bool
 8   Q            891 non-null    bool
 9   S            891 non-null    bool
dtypes: bool(3), float64(2), int64(5)
memory usage: 51.5 KB
```

[46]:
```
train_new=train.copy()
```

[47]:
```
train_new=train_new.astype({'male':int,'Q':int,'S':int})
```

[48]:
```
train_new.columns
```

[48]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'male', 'Q', 'S'],
      dtype='object')
```

[49]:
```
train_new.drop(['PassengerId'],inplace=True,axis=1)
```

[50]:
```
train_new
```

[50]:

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1
..	…	…	…	…	…	…	…	..	..
886	0	2	27.0	0	0	13.0000	1	0	1
887	1	1	19.0	0	0	30.0000	0	0	1
888	0	3	19.0	1	2	23.4500	0	0	1
889	1	1	26.0	0	0	30.0000	1	0	0
890	0	3	32.0	0	0	7.7500	1	1	0

[891 rows x 9 columns]

[51]:
```
from sklearn.model_selection import train_test_split
```

[52]:
```
x_train,x_test,y_train,y_test=train_test_split(train_new.
↪drop('Survived',axis=1),train_new['Survived'],test_size=0.
↪30,random_state=101)
```

[53]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[54]:
```
Knmodel= KNeighborsClassifier(n_neighbors=3)
```

[55]:
```
Knmodel.fit(x_train,y_train)
```

[55]:
```
KNeighborsClassifier(n_neighbors=3)
```

[56]:
```
predict=Knmodel.predict(x_test)
```

[57]:
```
confusion_matrix(y_test,predict)
```

[57]:
```
array([[127,  27],
       [ 58,  56]], dtype=int64)
```

[58]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.69      0.82      0.75       154
           1       0.67      0.49      0.57       114

    accuracy                           0.68       268
   macro avg       0.68      0.66      0.66       268
weighted avg       0.68      0.68      0.67       268
```

[59]:
```
#repeat with value 6
```

[60]:
```
Knmodel= KNeighborsClassifier(n_neighbors=40)
```

[61]:
```
Knmodel.fit(x_train,y_train)
```

[61]:
```
KNeighborsClassifier(n_neighbors=40)
```

[62]:
```
predict=Knmodel.predict(x_test)
```

[63]:
```
confusion_matrix(y_test,predict)
```

[63]:
```
array([[131,  23],
       [ 65,  49]], dtype=int64)
```

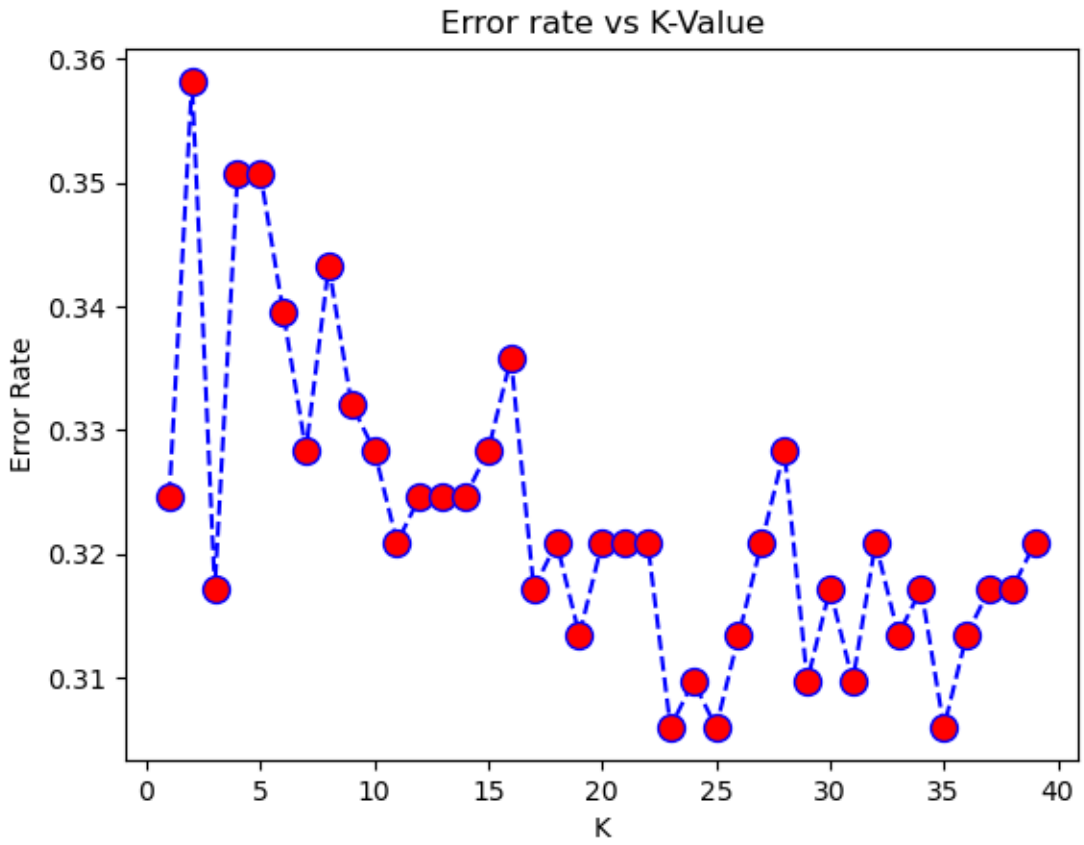
[64]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.67      0.85      0.75       154
           1       0.68      0.43      0.53       114

    accuracy                           0.67       268
   macro avg       0.67      0.64      0.64       268
weighted avg       0.67      0.67      0.65       268
```

[65]:
```
error_rate=[]
#will take some time
for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i=knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

[66]:
```
plt.
  ↪plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',markerfacecolor='red
plt.title('Error rate vs K-Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[66]:
```
Text(0, 0.5, 'Error Rate')
```



[67]:
```
from sklearn.preprocessing import StandardScaler
```

[68]:
```
scaler= StandardScaler()
scaler.fit(train_new.drop('Survived',axis=1))
scaled_features=scaler.transform(train_new.drop('Survived',axis=1))
```

[69]:
```
scaled_features
```

[69]:
```
array([[ 0.82737724, -0.52119766,  0.43279337, …,  0.73769513,
        -0.30756234,  0.61930636],
       [-1.56610693,  0.57872934,  0.43279337, …, -1.35557354,
        -0.30756234, -1.61470971],
       [ 0.82737724, -0.24621591, -0.4745452 , …, -1.35557354,
        -0.30756234,  0.61930636],
       …,
       [ 0.82737724, -0.72743397,  0.43279337, …, -1.35557354,
        -0.30756234,  0.61930636],
       [-1.56610693, -0.24621591, -0.4745452 , …,  0.73769513,
        -0.30756234, -1.61470971],
       [ 0.82737724,  0.16625671, -0.4745452 , …,  0.73769513,
```

```
        3.25137334, -1.61470971]])
```

[70]:
```
train_new.drop('Survived',axis=1).columns
```

[70]:
```
Index(['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'male', 'Q', 'S'],
dtype='object')
```

[71]:
```
train1=pd.DataFrame(scaled_features,columns=['Pclass', 'Age', 'SibSp', 'Parch',␣
↪'Fare', 'male', 'Q', 'S'])
```

[72]:
```
train1
```

[72]:

	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0.827377	-0.521198	0.432793	-0.473674	-0.502445	0.737695	-0.307562	0.619306
1	-1.566107	0.578729	0.432793	-0.473674	0.786845	-1.355574	-0.307562	-1.614710
2	0.827377	-0.246216	-0.474545	-0.473674	-0.488854	-1.355574	-0.307562	0.619306
3	-1.566107	0.372493	0.432793	-0.473674	0.420730	-1.355574	-0.307562	0.619306
4	0.827377	0.372493	-0.474545	-0.473674	-0.486337	0.737695	-0.307562	0.619306
..	…	…	…	…	…	…	…	…
886	-0.369365	-0.177470	-0.474545	-0.473674	-0.386671	0.737695	-0.307562	0.619306
887	-1.566107	-0.727434	-0.474545	-0.473674	-0.044381	-1.355574	-0.307562	0.619306
888	0.827377	-0.727434	0.432793	2.008933	-0.176263	-1.355574	-0.307562	0.619306
889	-1.566107	-0.246216	-0.474545	-0.473674	-0.044381	0.737695	-0.307562	-1.614710
890	0.827377	0.166257	-0.474545	-0.473674	-0.492378	0.737695	3.251373	-1.614710

[891 rows x 8 columns]