PROJECT REPORT
ON
WINE QUALITY CLASSIFICATION

Programming Language: Python

1. Separating raw data into Training data and Testing Data (three times each Wine).

Read data:

```
red_wine = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\winequality-red.csv',header = 0)
white_wine = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\winequality-white.csv',header = 0)
```

Separating Training (90%) and Testing data three time for each:

export csv file

```
X = red_wine.drop('quality',axis = 1)
y = red_wine['quality']
for i in range(3):
    i = i + 1
    X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0.9)
    train = pd.concat([X_train,y_train],axis=1)
    test = pd.concat([X_test,y_test],axis=1)
    train.to_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\train\red_train_{}.csv'.format(i), index = Fals
    test.to_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\test\red_test_{}.csv'.format(i), index = False)
```

```
X = white_wine.drop('quality',axis = 1)
y = white_wine['quality']
for i in range(3):
    i = i + 1
    X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0.9)
    train = pd.concat([X_train,y_train],axis=1)
    test = pd.concat([X_test,y_test],axis=1)
    train.to_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\train\white_train_{}.csv'.format(i), index = Fa
    test.to_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\test\white_test_{}.csv'.format(i), index = False
```

Determine Independent and Dependent variable:

```
red_train_1 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\train\red_train_1.csv')
red_train_2 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\train\red_train_2.csv')
red_train_3 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\train\red_train_3.csv')
red_test_1 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\test\red_test_1.csv')
red_test_2 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\test\red_test_2.csv')
red_test_3 = pd.read_csv(r'C:\Users\Do Anh Luyen\Creative Cloud Files\Desktop\Task 2\test\red_test_3.csv')
```

```
X_red_train_1=red_train_1.drop(['quality'],axis=1)
X_red_train_2=red_train_2.drop(['quality'],axis=1)
X_red_train_3=red_train_3.drop(['quality'],axis=1)
X_red_test_1=red_test_1.drop(['quality'],axis=1)
X_red_test_2=red_test_2.drop(['quality'],axis=1)
X_red_test_3=red_test_3.drop(['quality'],axis=1)
y_red_train_1=red_train_1['quality']
y_red_train_2=red_train_2['quality']
y_red_train_3=red_train_3['quality']
y_red_test_1=red_test_1['quality']
y_red_test_2=red_test_2['quality']
y_red_test_3=red_test_3['quality']
```

2. KMeans Model

- Data preprocessing:

There is no missing value or noise data.

Using StandardScaler for scaling data:

```
#scaler
std = StandardScaler()
X_red_train_1 = std.fit_transform(X_red_train_1)
X_red_test_1 = std.transform(X_red_test_1)
```

- Determine number of clusters for Red and White wine:

- Group Dependent variable:

0 will be represented for low quality (quality from 0 to 7).

1 will be represented for high quality (quality from 7 to 10).

```
y_red_train_1= [ 0 if ( i < 7 ) else 1 for i in y_red_train_1]
y_red_train_2= [ 0 if ( i < 7 ) else 1 for i in y_red_train_2]
y_red_train_3= [ 0 if ( i < 7 ) else 1 for i in y_red_train_3]
y_red_test_1= [ 0 if ( i < 7 ) else 1 for i in y_red_test_1]
y_red_test_2= [ 0 if ( i < 7 ) else 1 for i in y_red_test_2]
y_red_test_3= [ 0 if ( i < 7 ) else 1 for i in y_red_test_3]
```

```
y_white_train_1 = [0 if ( i < 7) else 1 for i in y_white_train_1]
y_white_train_2 = [0 if ( i < 7) else 1 for i in y_white_train_2]
y_white_train_3 = [0 if ( i < 7) else 1 for i in y_white_train_3]
y_white_test_1 = [0 if ( i < 7) else 1 for i in y_white_test_1]
y_white_test_2 = [0 if ( i < 7) else 1 for i in y_white_test_2]
y_white_test_3 = [0 if ( i < 7) else 1 for i in y_white_test_3]
```

⇨ For Red_Wine we will have n_clusters = 2.

⇨ For White_Wine we will have n_clusters = 2.

- Using Kmeans:

Red_Wine:

- Training data:

```
#Kmeans
km = KMeans(n_clusters = 2,random_state = 42)
km.fit(X_red_train_1)
clusters = km.predict(X_red_train_1)
```

+ Using PCA to 2-Dimensinal for Visualization:

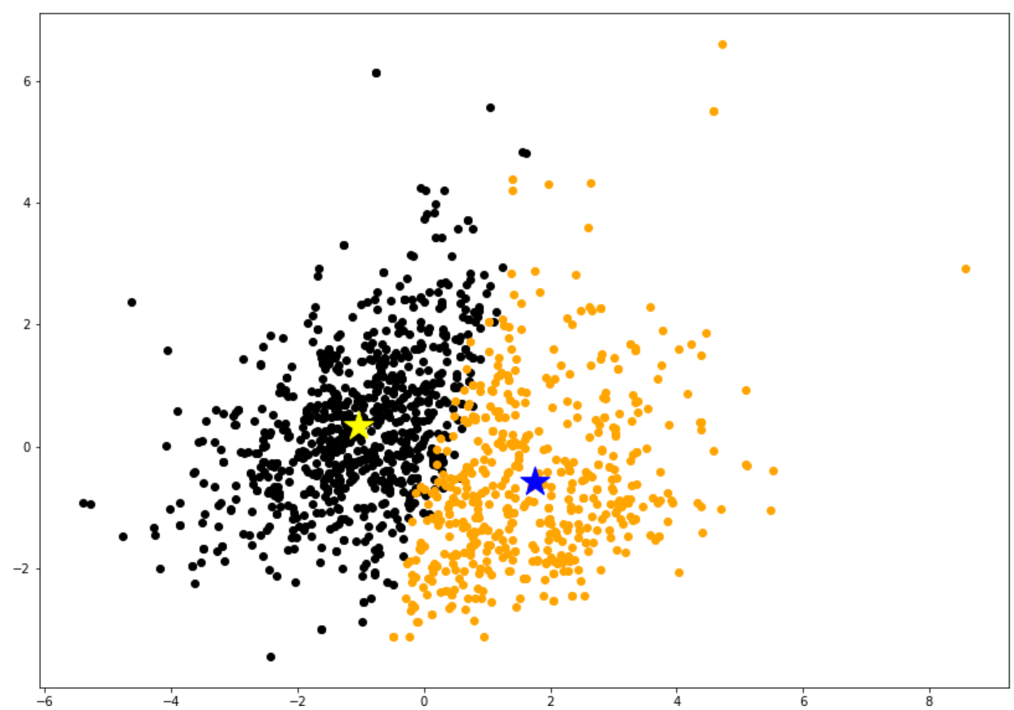
```
X_centre = km.cluster_centers_
```

```
pca = PCA(n_components = 2, random_state = 42)
X_red_train_1 = pd.DataFrame(pca.fit_transform(X_red_train_1),columns = ['PCA1','PCA2'])
X_red_train_1['cluster'] = clusters
X_centre = pd.DataFrame(pca.transform(X_centre),columns = ['PCA1','PCA2'])
```

+ Visualization:

```
plt.figure(figsize = (14,10))
plt.scatter(X_red_train_1[X_red_train_1['cluster'] == 0].loc[:,'PCA1'],X_red_train_1[X_red_train_1['cluster'] == 0].loc[:,'P
plt.scatter(X_red_train_1[X_red_train_1['cluster'] == 1].loc[:,'PCA1'],X_red_train_1[X_red_train_1['cluster'] == 1].loc[:,'P
plt.scatter(X_centre['PCA1'].loc[0],X_centre['PCA2'].loc[0],color = 'yellow',marker = '*',s = 600)
plt.scatter(X_centre['PCA1'].loc[1],X_centre['PCA2'].loc[1],color = 'blue',marker = '*',s = 600)
```

```
<matplotlib.collections.PathCollection at 0x245871c9940>
```



Yellow star is the central of 0 group.

Blue star is the central of 1 group.

- Tesing Data:

```
#for testing
clusters = km.predict(X_red_test_1)
X_centre = km.cluster_centers_

X_red_test_1 = pd.DataFrame(pca.transform(X_red_test_1),columns = ['PCA1','PCA2'])
X_red_test_1['cluster'] = clusters
X_centre = pd.DataFrame(pca.transform(X_centre),columns = ['PCA1','PCA2'])
```

+ Confusion matrix:

```
classificationSummary(y_red_test_1,clusters)
```

```
Confusion Matrix (Accuracy 0.6687)

       Prediction
Actual  0  1
     0 98 43
     1 10  9
```

```
precision = (9/(9+43))
print("Precision : {}".format(precision))
```

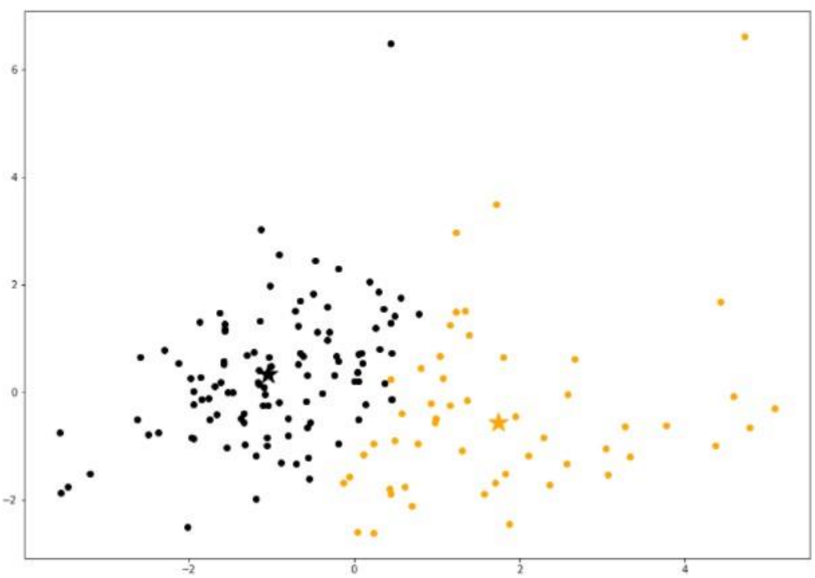
```
Precision : 0.17307692307692307
```

Accuracy = 0.6687,

True positive = 9, False positive = 43, TN = 98, FN = 10,

Precision = 0.173.

+ Visualization:



- For another testing set, we will have score below:

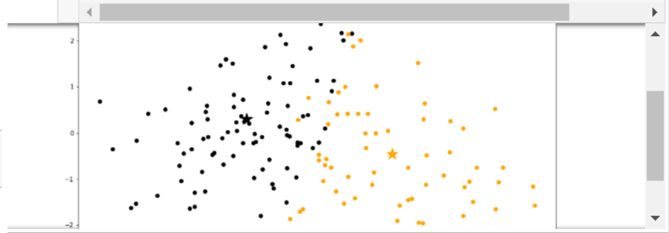
```
classificationSummary(y_red_test_2,clusters)
```

```
Confusion Matrix (Accuracy 0.6375)

       Prediction
Actual  0  1
     0 87 53
     1  5 15
```

```
precision = (15/(15+53))
print("Precision : {}".format(precision))
```

```
Precision : 0.22058823529411764
```

```
In [23]:
plt.figure(figsize = (14,10))
plt.scatter(X_red_test_2[X_red_test_2['cluster'] == 0].loc[:,'PCA1'],X_red_test_2[X_red_test_2['cluster'] == 0].loc[:,'PCA2'
plt.scatter(X_red_test_2[X_red_test_2['cluster'] == 1].loc[:,'PCA1'],X_red_test_2[X_red_test_2['cluster'] == 1].loc[:,'PCA2'

plt.scatter(X_centre['PCA1'].loc[0],X_centre['PCA2'].loc[0],color = 'black',marker = '*',s = 400)
plt.scatter(X_centre['PCA1'].loc[1],X_centre['PCA2'].loc[1],color = 'orange',marker = '*',s = 400)
```

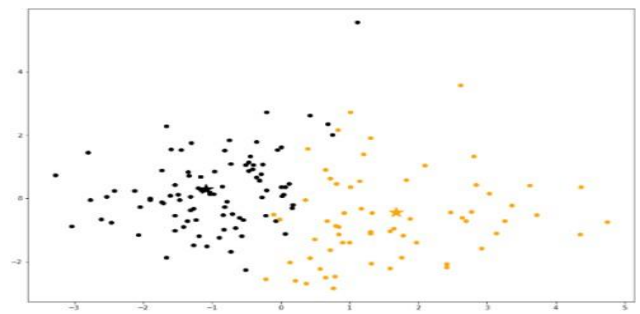


```
classificationSummary(y_red_test_3,clusters)
```

```
Confusion Matrix (Accuracy 0.6188)

       Prediction
Actual  0  1
     0 85 52
     1  9 14
```

```
precision = (14/(14+52))
print("Precision : {}".format(precision))
```

```
Precision : 0.21212121212121213
```



- Calculate Mean of 3 testing:

```
Mean = (a+b+c)/3
print("Mean_Confusion_matrix_Red_wine:\n {}".format(Mean))
mean = (a_1 + b_1 + c_1)/3
print("Mean_accuracy_Red_wine: {}".format(mean))
```

```
Mean_Confusion_matrix_Red_wine:
 [[90.         49.33333333]
 [ 8.         12.66666667]]
Mean_accuracy_Red_wine: 0.6416666666666666
```

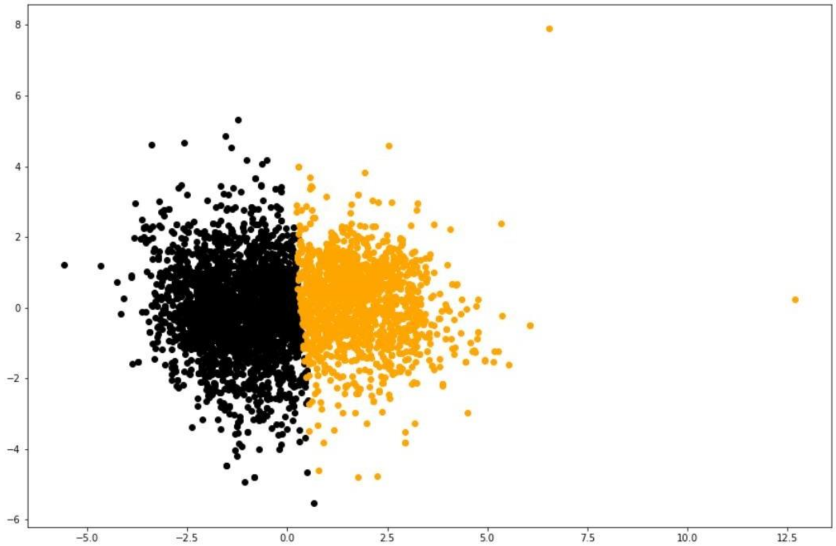
⇨ In conclusion, we have mean_accuracy equals 0.64

White_Wine:

- Training data set:

```
#train_1
std_1 = StandardScaler()
X_white_train_1 = std_1.fit_transform(X_white_train_1)
X_white_test_1 = std_1.transform(X_white_test_1)
km = KMeans(n_clusters = 2,random_state=42)
km.fit(X_white_train_1)
clusters = km.predict(X_white_train_1)
X_centre = km.cluster_centers_
pca_1 = PCA(n_components = 2,random_state=42)
X_white_train_1 = pd.DataFrame(pca_1.fit_transform(X_white_train_1),columns = ['PCA1','PCA2'])
X_white_train_1['cluster'] = clusters
X_centre = pd.DataFrame(pca_1.transform(X_centre),columns = ['PCA1','PCA2'])
```

+ Visualization:



- Testing data set:

```
#test1
clusters = km.predict(X_white_test_1)
X_white_test_1 = pd.DataFrame(pca_1.transform(X_white_test_1),columns = ['PCA1','PCA2'])
X_white_test_1['cluster'] = clusters
```

+ Confusion Matrix:

```
classificationSummary(y_white_test_1,clusters)
```

```
Confusion Matrix (Accuracy 0.4551)

       Prediction
Actual   0   1
     0 206 192
     1  75  17
```

```
precision = (17/(17+192))
print("Precision : {}".format(precision))
```

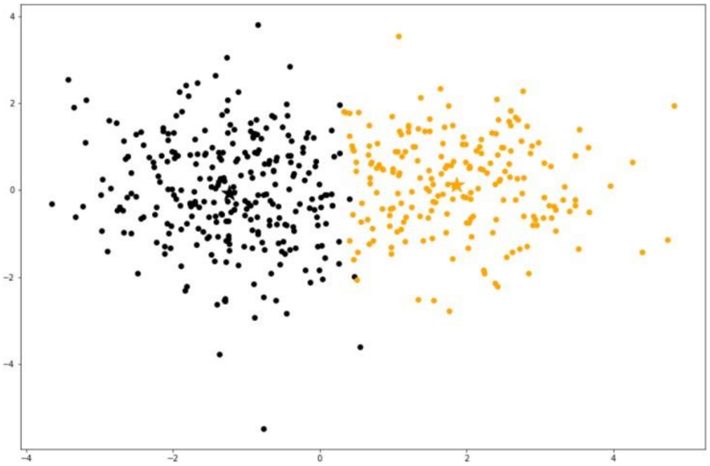
```
Precision : 0.08133971291866028
```

Accuracy = 0.455,

TP = 17, FP = 192, TN = 206, FN = 75,

Precision = 0.08.

+ Visualization:



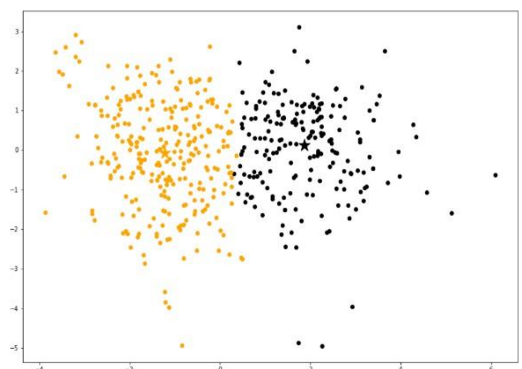
- For another testing sets:

```
1  classificationSummary(y_white_test_2,clusters)
```

```
Confusion Matrix (Accuracy 0.6020)

       Prediction
Actual   0   1
     0 197 187
     1   8  98
```

```
1  precision = (98/(98+187))
2  print("Precision : {}".format(precision))
```

```
Precision : 0.34385964912280703
```

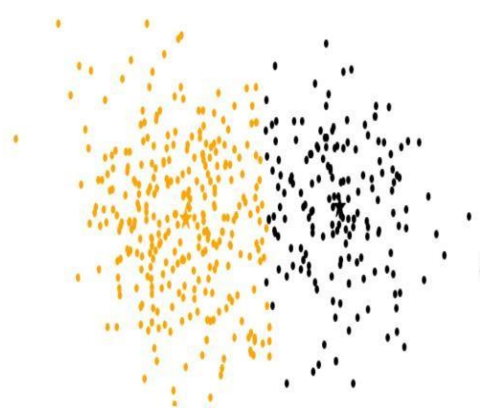


```
1  classificationSummary(y_white_test_3,clusters)
```

```
Confusion Matrix (Accuracy 0.5327)

       Prediction
Actual   0   1
     0 163 217
     1  12  98
```

```
1  precision = (98/(98+217))
2  print("Precision : {}".format(precision))
```

```
Precision : 0.3111111111111111
```



- Calculate Mean of 3 testing:

```
Mean1 = (d+e+f)/3
print("Mean_Confusion_matrix_White_wine:\n {}".format(Mean1))
mean1 = (d_1 + e_1 + f_1)/3
print("Mean_accuracy_White_wine: {}".format(mean1))
```

```
Mean_Confusion_matrix_White_wine:
 [[188.66666667 198.66666667]
 [ 31.66666667  71.        ]]
Mean_accuracy_White_wine: 0.5299319727891156
```

⇨ In conclusion, Mean_accuracy for White_wine is 0.53.

3. Regression model:

- Preprocessing data (StandardScaler):

```
mm = MinMaxScaler()
X_red_train_1 = mm.fit_transform(X_red_train_1)
X_red_test_1 = mm.transform(X_red_test_1)
```

```
X_white_train_1 = mm.fit_transform(X_white_train_1)
X_white_test_1 = mm.transform(X_white_test_1)
```

- Linear Regression:

Red_Wine:

```
lr = LinearRegression().fit(X_red_train_1,y_red_train_1)
```

- Confusion matrix:

```
confusion_matrix = confusion_matrix(y_red_test_1,y_prediction )
print(confusion_matrix)
```

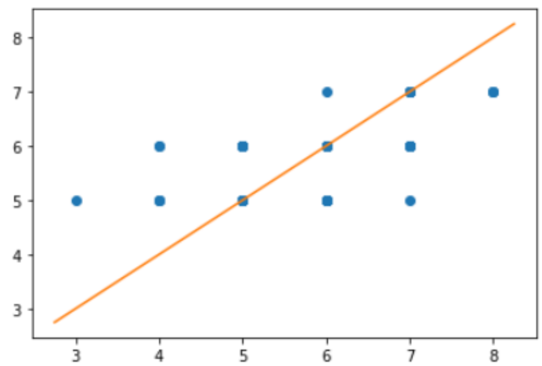
```
[[ 0  0  1  0  0  0]
 [ 0  0  3  2  0  0]
 [ 0  0 41 18  0  0]
 [ 0  0 32 43  1  0]
 [ 0  0  1  8  7  0]
 [ 0  0  0  0  3  0]]
```

- False positive, false negative, true positive, true negative, precision, accuracy.

	3	4	5	6	7	8
FP	0.0	0.0	37.000000	28.000000	4.000000	0.0
FN	1.0	5.0	18.000000	33.000000	9.000000	3.0
TP	0.0	0.0	41.000000	43.000000	7.000000	0.0
TN	159.0	155.0	64.000000	56.000000	140.000000	157.0
Pecision	NaN	NaN	0.525641	0.605634	0.636364	NaN

Accuracy: `Accurancy: 0.2905520217191234`

- Visualization:



White_Wine:

```
lr = LinearRegression().fit(X_white_train_1,y_white_train_1)
```

- Confusion matrix:

```
confusion_matrix = confusion_matrix(y_white_test_1,y_prediction )
print(confusion_matrix)
```

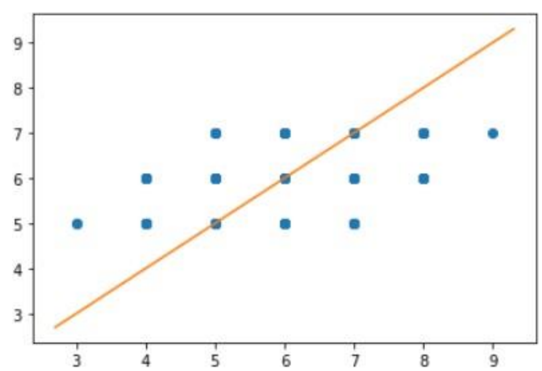
```
[[  0   0   2   0   0   0   0]
 [  0   0  12  10   0   0   0]
 [  0   0  55  89   4   0   0]
 [  0   0  31 172  23   0   0]
 [  0   0   5  48  21   0   0]
 [  0   0   0  10   7   0   0]
 [  0   0   0   0   1   0   0]]
```

- False positive, false negative, true positive, true negative, precision, accuracy.

	3	4	5	6	7	8	9
FP	0.0	0.0	50.00000	157.000000	35.000	0.0	0.0
FN	2.0	22.0	93.00000	54.000000	53.000	17.0	1.0
TP	0.0	0.0	55.00000	172.000000	21.000	0.0	0.0
TN	488.0	468.0	292.00000	107.000000	381.000	473.0	489.0
Precision	NaN	NaN	0.52381	0.522796	0.375	NaN	NaN

Accuracy: `0.2655541307739012`

- Visualization:



- Logistic Regression:

Red_Wine:

```
In [68]:
lg = LogisticRegression(max_iter = 10000).fit(X_red_train_1,y_red_train_1)

print("Accurancy: {}".format(lg.score(X_red_test_1,y_red_test_1)))
y_prediction = lg.predict(X_red_test_1)
y_true = y_red_test_1
```

```
Accurancy: 0.4875
```

```
In [69]:
confusion_matrix = confusion_matrix(y_red_test_1,y_prediction )
print(confusion_matrix)
```

```
[[ 0  0  1  0  0  0]
 [ 0  0  2  3  0  0]
 [ 0  0 44 15  0  0]
 [ 0  0 43 32  1  0]
 [ 0  0  1 13  2  0]
 [ 0  0  0  1  2  0]]
```

```
In [70]:
X = a(y_true,y_prediction)
pd.DataFrame(X,columns = sorted(y_red_test_1.unique()), index = ['FP','FN','TP','TN','Prcision'])
```

```
<ipython-input-58-87a397bf5c65>:8: RuntimeWarning: invalid value encountered in true_divide
  Precision = TP/(TP+FP)
```

Out[70]:

	3	4	5	6	7	8
FP	0.0	0.0	47.000000	32.0	3.0	0.0
FN	1.0	5.0	15.000000	44.0	14.0	3.0
TP	0.0	0.0	44.000000	32.0	2.0	0.0
TN	159.0	155.0	54.000000	52.0	141.0	157.0
Prcision	NaN	NaN	0.483516	0.5	0.4	NaN

In [69] is the confusion matrix.

In [70] is the False positive, false negative, true positive, true negative, precision, accuracy.

In [68] is the logistic model application and Accuracy = 0.4875.

White_Wine:

In [79]:
```python
lg = LogisticRegression(max_iter = 10000).fit(X_white_train_1,y_white_train_1)
print("Accuray: {}".format(lg.score(X_white_test_1,y_white_test_1)))
y_prediction = lg.predict(X_white_test_1)
y_true = y_white_test_1
```

```
Accuray: 0.5204081632653061
```

In [80]:
```python
confusion_matrix = confusion_matrix(y_white_test_1,lg.predict(X_white_test_1))
print(confusion_matrix)
```

```
[[  0   0   2   0   0   0   0]
 [  0   0  15   7   0   0   0]
 [  0   0  69  78   1   0   0]
 [  0   0  42 173  11   0   0]
 [  0   0   5  56  13   0   0]
 [  0   0   1  12   4   0   0]
 [  0   0   0   0   1   0   0]]
```

In [81]:
```python
X = a(y_true,y_prediction)
pd.DataFrame(X,columns = sorted(y_white_test_1.unique()), index = ['FP','FN','TP','TN','Precision'])
```

```
<ipython-input-58-87a397bf5c65>:8: RuntimeWarning: invalid value encountered in true_divide
  Precision = TP/(TP+FP)
```

Out[81]:

	3	4	5	6	7	8	9
FP	0.0	0.0	65.000000	153.000000	17.000000	0.0	0.0
FN	2.0	22.0	79.000000	53.000000	61.000000	17.0	1.0
TP	0.0	0.0	69.000000	173.000000	13.000000	0.0	0.0
TN	488.0	468.0	277.000000	111.000000	399.000000	473.0	489.0
Precision	NaN	NaN	0.514925	0.530675	0.433333	NaN	NaN

In [80] is confusion matrix.

In [81] is the False positive, false negative, true positive, true negative, precision, accuracy.

In [79] is the logistic model application and Accuracy is 0.52.

4. Comparing:

Before separating Dependent variable (quality) into two group 0 and 1 (low and high quality) and run Kmeans with n_clusters = 2 for both

Red_wine and White_wine, my team has already run Kmeans without separating Dependent variable which means n_clusters = 6 for Red_wine and n_clusters = 7 for White_wine, we saw that out accuracy very bad just 20% or less. This score is lower than LinearRegression model (around 30%) because Kmeans is unsupervised learning model. This means Kmeans didn't learn from the dependent variable, it just clusters based on the distance between each point, higher clusters will lead to lower accuracy, while Linear Regression learning from both Independent and Dependent variable. Otherwise, Logistic Regression has higher score than Linear Regression because the dependent variable in this exercise is categorical variable so classification model will be the best.

Eg: Logistic Regression, Decision Tree, Random Forest, Support Vector Machine.