Exam 2

1. What exactly is the business problem you are trying to solve? Summarize this in the form of a meaningful problem statement?

We need to design a machine learning solution for determining the category or variety of grape used in making wines based on several chemical characteristics of individual wines.

In our problem we design a machine learning solution for predicting the different types of target variable cultivar based on the characteristics of remaining 13 variables which represnt the chemical constitunets of wines.

2. What are some of preliminary decisions you may need to make based on your problem statement? Your answer should include identification of an initial machine learning algorithm you will apply with respect to the problem statement in (1). Justification should be based on identification of the category of machine learning (supervised, unsupervised, etc.) as well as suggested machine learning algorithm from within the identified machine learning category.

Based on my analysis we can consider two approaches as the data for target variable cultivar is categorical and it is multiclass which is it has 3 types of cultivar categories cultivar1,cultivar2,cultivar3 which are labelled.

Unsupervised learning clustering using K-means algorithm In clustering similar data can be clustered into groups based on centroids for k-means ,hence clustering based on cultivar variable can be experimented.Hence all records with cultivar0 can be grouped into one cluster,records with cultivar1 into another cluster and cultivar2 into another cluster.

Logistic regression can be applied when the target variable is categorical and multiclass, Hence it can be used in this case as the data is categorized into 3types cultivar0,cultivar1,cultivar2.We can apply logistic regression to predict the output of the categorical varible by converting it into numeric form using LabelEncoder.

In [1]:
```
import pandas  as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics

pd.set_option('display.max_rows', 200)
pd.set_option('display.max_columns', 120)
```

In [2]:
```
WineDF = pd.read_csv('F://wine.csv', header=0, sep=',')
```

In [3]:
```
WineDF.head()
```

Out[3]:

	Cultivar	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids
0	cultivar1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	cultivar1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	cultivar1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	cultivar1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	cultivar1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39

3. Keeping your preliminary decisions from (2) in mind, peruse the dataset to:

a. Display the datatype of each of the 14 columns to determine if any of the columns need to be transformed to comply with the requirements of your chosen algorithm. Specify the names of columns that require transformation along with the transformation that need to be performed. Include a reasonable explanation as to why the columns need to be transformed as well as what appropriate transformation will be necessary to make the feature algorithm-compliant.

In [4]:
```
print('The shape of the data is:',WineDF.shape)
```

```
The shape of the data is: (178, 14)
```

In [5]:
```
WineDF.columns
```

Out[5]:
```
Index(['Cultivar', 'Alcohol', 'MalicAcid', 'Ash', 'Alkalinity', 'Magnesium',
       'Phenols', 'Flavanoids', 'NonFlavanoids', 'Pcyanins', 'ColorIntensit
y',
       'Hue', 'OD280', 'Proline'],
      dtype='object')
```

Above are the chemical characteristics and the cultivar attribute names which are present in our dataset.

In [6]:
```
WineDF.dtypes
```

Out[6]:
```
Cultivar          object
Alcohol          float64
MalicAcid        float64
Ash              float64
Alkalinity       float64
Magnesium          int64
Phenols          float64
Flavanoids       float64
NonFlavanoids    float64
Pcyanins         float64
ColorIntensity   float64
Hue              float64
OD280            float64
Proline            int64
dtype: object
```

We need to do transofrmation on target variable attribute Cultivar using LabelEncoder, since the data is categorical we need to convert it to numerical for implementing in our machine learning algorithm K-Means

b. Identify any other data cleanup and pre-processing that may be required to get the data ready for your chosen machine learning algorithm. This may include handling missing values. Missing values for any feature are to be replaced with a median value for that feature. State so if missing values are not indicated.

In [7]:
```
WineDF.groupby('Cultivar').count()
```

Out[7]:

Cultivar	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids	F
cultivar1	59	59	59	59	59	59	59	59	
cultivar2	71	71	71	71	71	71	71	71	
cultivar3	48	48	48	48	48	48	48	48	

Dataset is unbalanced .Here we can see that the target variable cultivar is unevenly distributed across the dataset.

Cultivar1 produces 34% of total wines ,where as cultivar3 produces 26% and the hughest is produced by cultivar2 with 40% of the total wines.

In [8]:
```
WineDF.isnull().any()
```

Out[8]:
```
Cultivar          False
Alcohol           False
MalicAcid         False
Ash               False
Alkalinity        False
Magnesium         False
Phenols           False
Flavanoids        False
NonFlavanoids     False
Pcyanins          False
ColorIntensity    False
Hue               False
OD280             False
Proline           False
dtype: bool
```

There are no nulls or missing data for any attributes hence we can no more transofrmations are required

In [9]:
```
WineDF.head()
```

Out[9]:

	Cultivar	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids
0	cultivar1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.2
1	cultivar1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.2
2	cultivar1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.3
3	cultivar1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.2
4	cultivar1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.3

In [10]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
WineDF[['Alcohol', 'MalicAcid', 'Ash', 'Alkalinity','Magnesium','Phenols','Fla
vanoids','NonFlavanoids','Pcyanins','ColorIntensity','Hue','OD280','Proline']]
= scaler.fit_transform(WineDF[['Alcohol', 'MalicAcid', 'Ash', 'Alkalinity','Ma
gnesium','Phenols','Flavanoids','NonFlavanoids','Pcyanins','ColorIntensity','H
ue','OD280','Proline']])
Wine_DF = WineDF
Wine_DF.head()
```

Out[10]:

	Cultivar	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlav
0	cultivar1	0.842105	0.191700	0.572193	0.257732	0.619565	0.627586	0.573840	0
1	cultivar1	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	0.510549	0
2	cultivar1	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	0.611814	0
3	cultivar1	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	0.664557	0
4	cultivar1	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	0.495781	0

Scaling is applied to normalize the data and standardize it across all the 13 variables

4. Perform preliminary exploratory data analysis (EDA) pertinent to the problem statement and your chosen machine learning algorithm in (2). This may include basic statistics, data shape, grouping on the outcome variable, generating scatter plots or line plots, etc. as appropriate based on your chosen algorithm. Anything that can give you further insight into your dataset vis-à-vis the machine learning algorithm you have selected should be included with an explanation/conclusion of the output.

In [11]:
```
Wine_DF.describe()
```

Out[11]:

	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	Nor
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
mean	0.518584	0.315484	0.538244	0.458502	0.323278	0.453487	0.356386	
std	0.213639	0.220780	0.146708	0.172142	0.155244	0.215811	0.210730	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.350658	0.170455	0.454545	0.340206	0.195652	0.262931	0.182489	
50%	0.531579	0.222332	0.534759	0.458763	0.304348	0.474138	0.378692	
75%	0.696711	0.462945	0.640374	0.561856	0.402174	0.627586	0.534810	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

The data is described for statistical mean and deviation values of ecah attribute in the dataframe.

In [13]:
```
Wine_DF.groupby('Cultivar').mean()
```

Out[13]:

Cultivar	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavan
cultivar1	0.714407	0.251122	0.585879	0.331819	0.394989	0.641438	0.557463	0.301
cultivar2	0.328614	0.235707	0.473149	0.496806	0.266840	0.440991	0.367267	0.440
cultivar3	0.558882	0.512599	0.575980	0.557560	0.318614	0.240948	0.093135	0.599

None of the chemicals make a significant difference on Cultivar when distibution of data compared with mean value

Data visualization

The statistical significance of data is as below using data visualization:

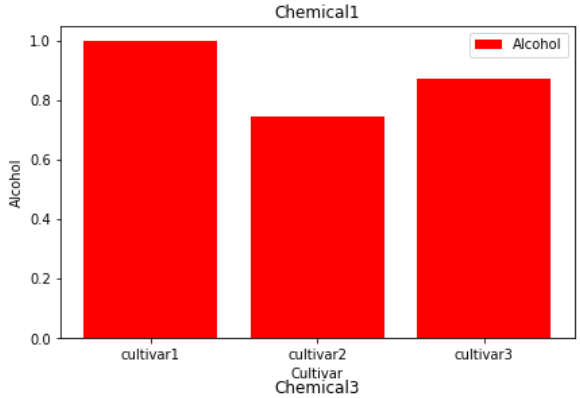
In [14]:
```
import matplotlib.pyplot as plt
```

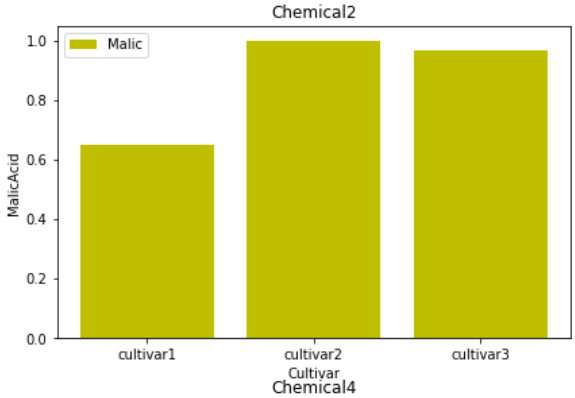
In [15]:
```
Cultivar_list=Wine_DF['Cultivar'].values.tolist()
Alcohol_list=Wine_DF['Alcohol'].values.tolist()
Mal_list=Wine_DF['MalicAcid'].values.tolist()
Ash_list=Wine_DF['Ash'].values.tolist()
Alk_list=Wine_DF['Alkalinity'].values.tolist()
Mag_list=Wine_DF['Magnesium'].values.tolist()
Phen_list=Wine_DF['Phenols'].values.tolist()
Flav_list=Wine_DF['Flavanoids'].values.tolist()
NonFlav_list=Wine_DF['NonFlavanoids'].values.tolist()
Pcy_list=Wine_DF['Pcyanins'].values.tolist()
Color_list=Wine_DF['ColorIntensity'].values.tolist()
Hue_list=Wine_DF['Hue'].values.tolist()
OD_list=Wine_DF['OD280'].values.tolist()
Pro_list=Wine_DF['Proline'].values.tolist()
```

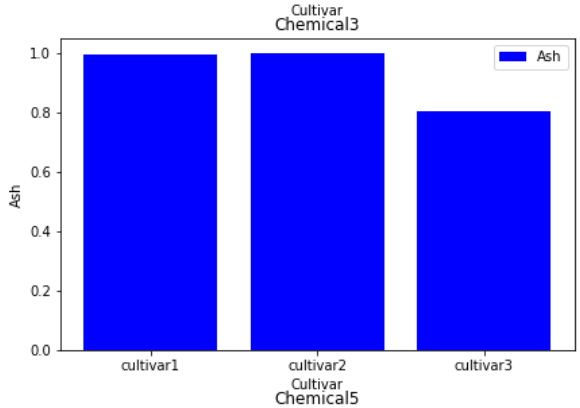
I am converting the chemical attributes to list to see a data visualization on the distribution of each chemical on the Cultivar target variable.

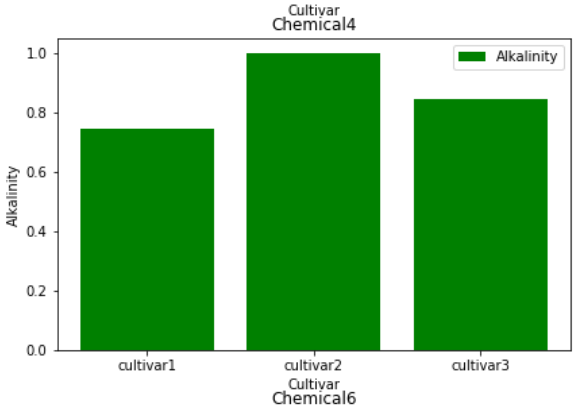
In [16]:
```
plt.figure(figsize=(15,40))
plt.subplot(8,2,1) #where 3 is the number of rows and 2 is columns for the par
tition in final image
plt.bar(Cultivar_list,Alcohol_list,label='Alcohol',color='r')
plt.title('Chemical1')
plt.ylabel('Alcohol')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,2)
plt.bar(Cultivar_list,Mal_list,label='Malic',color='y')
plt.title('Chemical2')
plt.ylabel('MalicAcid')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,3)
plt.bar(Cultivar_list,Ash_list,label='Ash',color='b')
plt.title('Chemical3')
plt.ylabel('Ash')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,4)
plt.bar(Cultivar_list,Alk_list,label='Alkalinity',color='g')
plt.title('Chemical4')
plt.ylabel('Alkalinity')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,5)
plt.bar(Cultivar_list,Mag_list,label='Magnesium',color='m')
plt.title('Chemical5')
plt.ylabel('Magnesium')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,6)
plt.bar(Cultivar_list,Phen_list,label='Phenols',color='r')
plt.title('Chemical6')
plt.ylabel('Phenols')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,7)
plt.bar(Cultivar_list,Flav_list,label='Flavanoids',color='y')
plt.title('Chemical7')
plt.ylabel('Flavanoids')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,8)
plt.bar(Cultivar_list,NonFlav_list,label='NonFlavanoids',color='b')
plt.title('Chemical8')
plt.ylabel('NonFlavanoids')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,9)
plt.bar(Cultivar_list,Pcy_list,label='Pcyanins',color='g')
plt.title('Chemical9')
plt.ylabel('Pcyanins')
plt.xlabel('Cultivar')
plt.legend(loc='best')
```

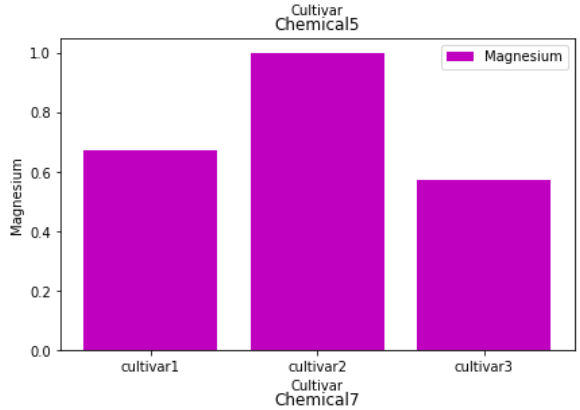
```
plt.subplot(8,2,10)
plt.bar(Cultivar_list,Color_list,label='ColorIntensity',color='m')
plt.title('Chemical10')
plt.ylabel('ColorIntensity')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,11)
plt.bar(Cultivar_list,Hue_list,label='Hue',color='r')
plt.title('Chemical11')
plt.ylabel('Hue')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,12)
plt.bar(Cultivar_list,OD_list,label='OD280',color='y')
plt.title('Chemical12')
plt.ylabel('OD280')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.subplot(8,2,13)
plt.bar(Cultivar_list,Pro_list,label='Proline',color='b')
plt.title('Chemical13')
plt.ylabel('Proline')
plt.xlabel('Cultivar')
plt.legend(loc='best')
plt.show()
```

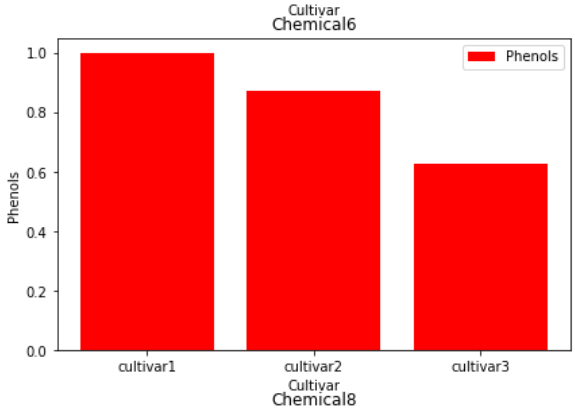

Chemical1
Alcohol
cultivar1
cultivar2
cultivar3
Cultivar

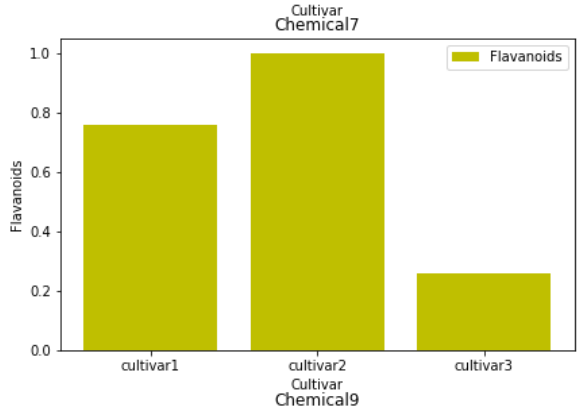

Chemical2
Malic
MalicAcid
cultivar1
cultivar2
cultivar3
Cultivar

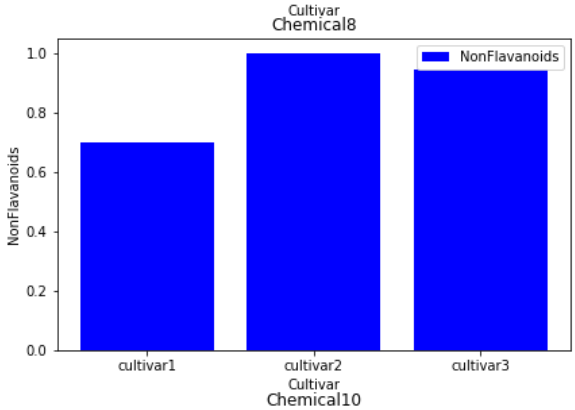

Chemical3
Ash
cultivar1
cultivar2
cultivar3
Cultivar

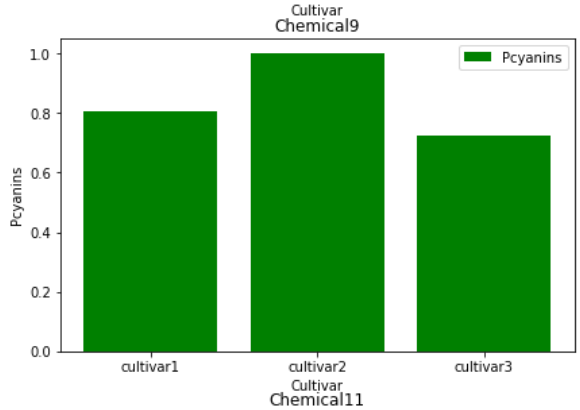

Chemical4
Alkalinity
cultivar1
cultivar2
cultivar3
Cultivar

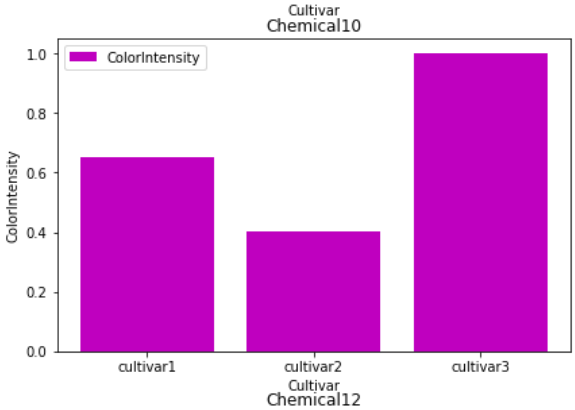

Chemical5
Magnesium
cultivar1
cultivar2
cultivar3
Cultivar

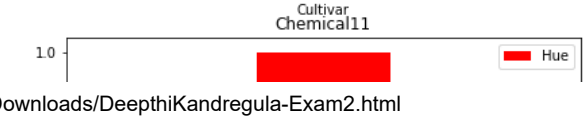

Chemical6
Phenols
cultivar1
cultivar2
cultivar3
Cultivar

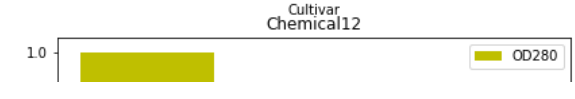

Chemical7
Flavanoids
cultivar1
cultivar2
cultivar3
Cultivar

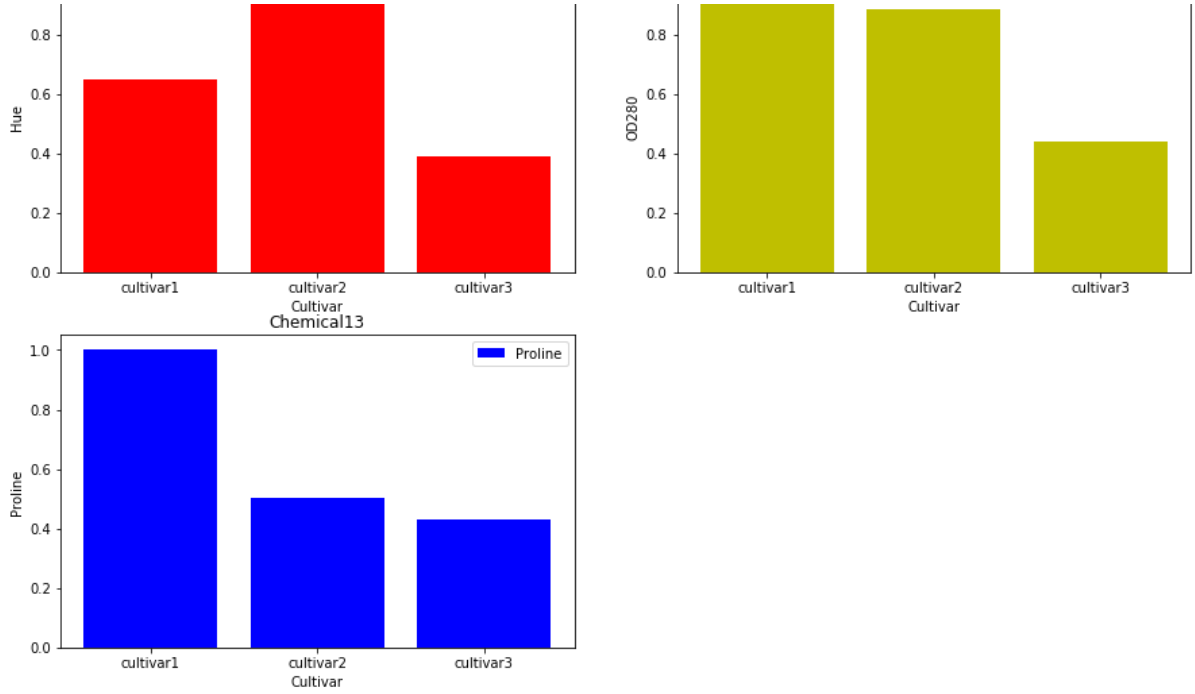

Chemical8
NonFlavanoids
cultivar1
cultivar2
cultivar3
Cultivar


Chemical9
Pcyanins
cultivar1
cultivar2
cultivar3
Cultivar


Chemical10
ColorIntensity
cultivar1
cultivar2
cultivar3
Cultivar


Chemical11
Hue


Chemical12
OD280



Hence we can see that the cultivars are unevenly distributed for each chemical.

Transformation

In [17]:
```
WineNumericDF=Wine_DF.drop('Cultivar',axis=1)
WineNumericDF.head()
```

Out[17]:

	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids	I
0	0.842105	0.191700	0.572193	0.257732	0.619565	0.627586	0.573840	0.283019	
1	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	0.510549	0.245283	
2	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	0.611814	0.320755	
3	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	0.664557	0.207547	
4	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	0.495781	0.490566	

In [18]:
```
from sklearn.preprocessing import LabelEncoder

lbl_encoder = LabelEncoder()
WineNumericDF['Cultivar_code'] = lbl_encoder.fit_transform(WineDF['Cultivar'])
```

LabelEncoder converts categorical variable to numeric data and classifies it into 3 groups 0,1,2

In [19]:
```
WineFeaturesDF=WineNumericDF.drop('Cultivar_code',axis=1)
WineTargetDF=WineNumericDF[['Cultivar_code']]
```

We convert Features and Target Dataframes s that we use the features dataframe for predicting the variable and Target dataframe for performance of target variable

In [20]:
```
WineFeaturesDF.shape
```

Out[20]:
```
(178, 13)
```

In [21]:
```
WineTargetDF.shape
```

Out[21]:
```
(178, 1)
```

Verified shape to not miss any data

In [22]:
```
WineTargetDF.groupby('Cultivar_code').size()
```

Out[22]:
```
Cultivar_code
0    59
1    71
2    48
dtype: int64
```

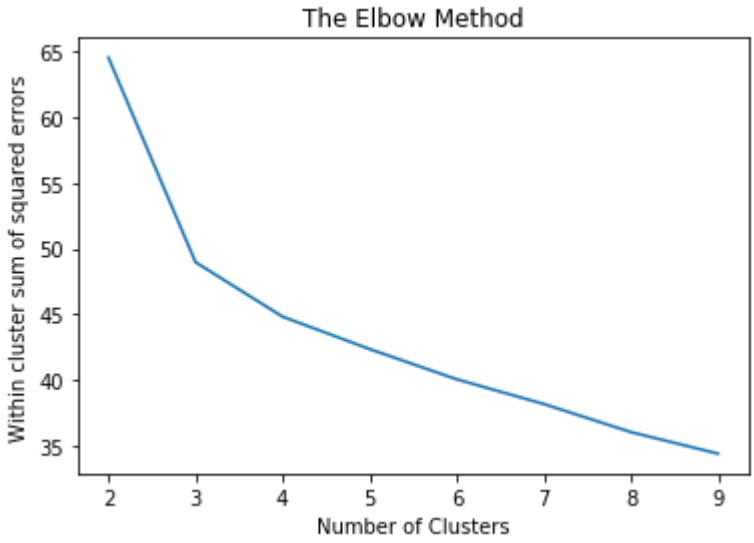
Data is similar to original data after transforation with Cultivar2 having highest produce of 71records out of 178 that is 40%

In [23]:
```
from sklearn.metrics import silhouette_score

WCSSE = []
silhouette=[]
for k in range(2,10):
    km = KMeans(n_clusters = k, random_state=20)
    clusters = km.fit(WineFeaturesDF)
    print(km.inertia_)
    WCSSE.append(km.inertia_)
    silhouette.append(silhouette_score(WineFeaturesDF, clusters.labels_))
    print("with k = {}".format(k))
    print("within Cluster SSE = " +  str(WCSSE[k-2])  + '\t' + "Silhouette Sco
re = " + str(silhouette[k-2]))
    print("---"*30)
```

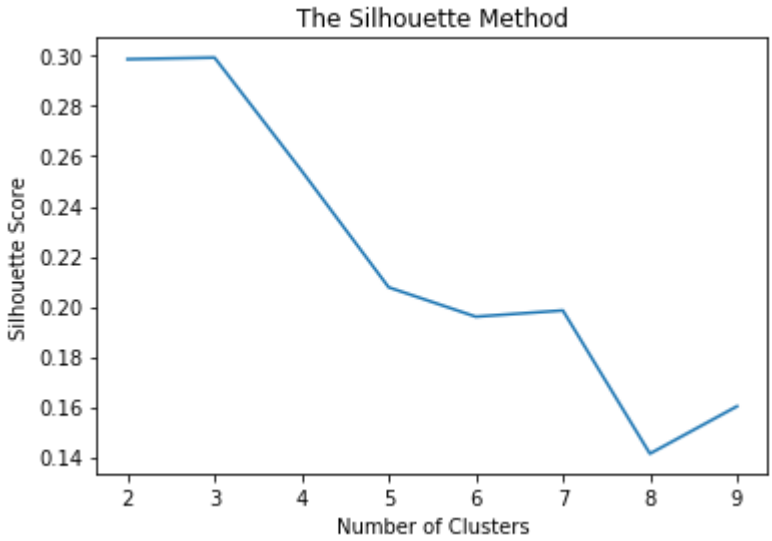
```
64.53766702389431
with k = 2
within Cluster SSE = 64.53766702389431  Silhouette Score = 0.2987221815974774
3
-------------------------------------------------------------------------------
-------------
48.97029115513917
with k = 3
within Cluster SSE = 48.97029115513917  Silhouette Score = 0.2993667406489506
4
-------------------------------------------------------------------------------
-------------
44.82503028071058
with k = 4
within Cluster SSE = 44.82503028071058  Silhouette Score = 0.2542051106557118
3
-------------------------------------------------------------------------------
-------------
42.357923449556
with k = 5
within Cluster SSE = 42.357923449556    Silhouette Score = 0.2076880964107259
7
-------------------------------------------------------------------------------
-------------
40.05896406239196
with k = 6
within Cluster SSE = 40.05896406239196  Silhouette Score = 0.1959834629055433
6
-------------------------------------------------------------------------------
-------------
38.18015131891916
with k = 7
within Cluster SSE = 38.18015131891916  Silhouette Score = 0.1984677968609780
7
-------------------------------------------------------------------------------
-------------
36.04174614917319
with k = 8
within Cluster SSE = 36.04174614917319  Silhouette Score = 0.1413321753522848
2
-------------------------------------------------------------------------------
-------------
34.40317499831163
with k = 9
within Cluster SSE = 34.40317499831163  Silhouette Score = 0.1602280556203290
7
-------------------------------------------------------------------------------
-------------
```

In [24]:
```
plt.plot(range(2,10), WCSSE)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Within cluster sum of squared errors')
plt.show()
```



The optimal point is k=3

In [25]:
```
plt.plot(range(2,10), silhouette)
plt.title('The Silhouette Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



The optimal point is k=3 as there is a large downfall from 3 to 5.

In [26]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=3, random_state=20)
predictions = km.fit_predict(WineFeaturesDF)

target = WineTargetDF['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(WineTargetDF['Cultivar_code'], predictio
ns)
print(homogeneity_coeff)

completeness_coeff = completeness_score(WineTargetDF['Cultivar_code'], predict
ions)
print(completeness_coeff)


pd.crosstab(WineTargetDF['Cultivar_code'],predictions)
```

```
0.8366403391266617
0.8196302687581578
0.8113987824934515
```

Out[26]:

col_0	0	1	2
Cultivar_code			
0	0	1	58
1	7	62	2
2	48	0	0

Cultivar1 is distributed 98% to cluster2 and 2% to cluster1.

Cultivar2 is distributed 9% to cluster0,87% to cluster1 and 3% to cluster2

Cultivar3 is distributed 100% to cluster0

5. If your chosen algorithm demands training and test datasets, split your wine dataset using an 80/20 split. If dataset is split, evaluate your training and test datasets to ensure they are representative of your full data set.

In [27]:
```
from sklearn.model_selection import train_test_split
X1_train, X1_test, Y1_train, Y1_test = train_test_split(WineFeaturesDF, WineTargetDF, test_size=0.2, random_state=42)
X1_train.shape, Y1_train.shape, X1_test.shape, Y1_test.shape
```

Out[27]: ((142, 13), (142, 1), (36, 13), (36, 1))

In [28]:
```
X1_train.head()
```

Out[28]:

	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids
158	0.871053	0.185771	0.716578	0.742268	0.304348	0.627586	0.204641	0.754717
137	0.394737	0.942688	0.684492	0.742268	0.282609	0.279310	0.054852	0.943396
98	0.352632	0.065217	0.395722	0.407216	0.195652	0.875862	0.719409	0.207547
159	0.644737	0.183794	0.684492	0.613402	0.206522	0.558621	0.160338	0.735849
38	0.536842	0.150198	0.395722	0.252577	0.304348	0.489655	0.485232	0.283019

In [29]:
```
Y1_train.head()
```

Out[29]:

	Cultivar_code
158	2
137	2
98	1
159	2
38	0

In [30]:
```
X1_test.head()
```

Out[30]:

	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	NonFlavanoids
19	0.686842	0.466403	0.641711	0.237113	0.500000	0.593103	0.567511	0.075472
45	0.836842	0.652174	0.577540	0.427835	0.445652	0.644828	0.487342	0.320755
140	0.500000	0.409091	0.716578	0.536082	0.282609	0.193103	0.033755	0.754717
30	0.710526	0.150198	0.716578	0.613402	0.336957	0.696552	0.613924	0.301887
67	0.352632	0.084980	0.299465	0.463918	0.086957	0.389655	0.350211	0.264151

In [31]:
```
Y1_test.head()
```

Out[31]:

	Cultivar_code
19	0
45	0
140	2
30	0
67	1

In [32]:
```
Y1_train.groupby('Cultivar_code').size()
```

Out[32]:
```
Cultivar_code
0    45
1    57
2    40
dtype: int64
```

Cultivar1 is representing 76% of original cultivar1 original data , Cultivar2 is represnting 80% of the cultivar2 original data, Cultivar3 is representing 83% of Cultivar3 original data

In [33]:
```
Y1_test.groupby('Cultivar_code').size()
```

Out[33]:
```
Cultivar_code
0    14
1    14
2     8
dtype: int64
```

Cultivar1 is representing 24% of original cultivar1 original data , Cultivar2 is represnting 20% of the cultivar2 original data, Cultivar3 is representing 17% of Cultivar3 original data

6. Use the relevant portion of your dataset to train the model of your selected machine learning algorithm. Do all the necessary preprocessing to determine the parameters for your selected algorithm. For example, you will need to specify (and justify) the number of clusters if you choose to use KMeans clustering algorithm via the Elbow curve, Silhouette analysis, etc.

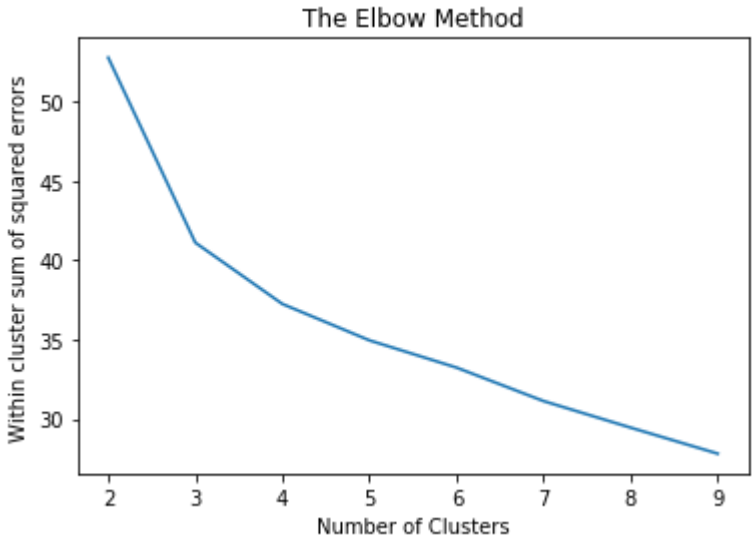
Train Model

In [34]:
```
from sklearn.metrics import silhouette_score

WCSSE = []
silhouette=[]
for k in range(2,10):
    km = KMeans(n_clusters = k, random_state=20)
    clusters = km.fit(X1_train)
    print(km.inertia_)
    WCSSE.append(km.inertia_)
    silhouette.append(silhouette_score(X1_train, clusters.labels_))
    print("with k = {}".format(k))
    print("within Cluster SSE = " +  str(WCSSE[k-2])  + '\t' + "Silhouette Sco
re = " + str(silhouette[k-2]))
    print("---"*30)
```

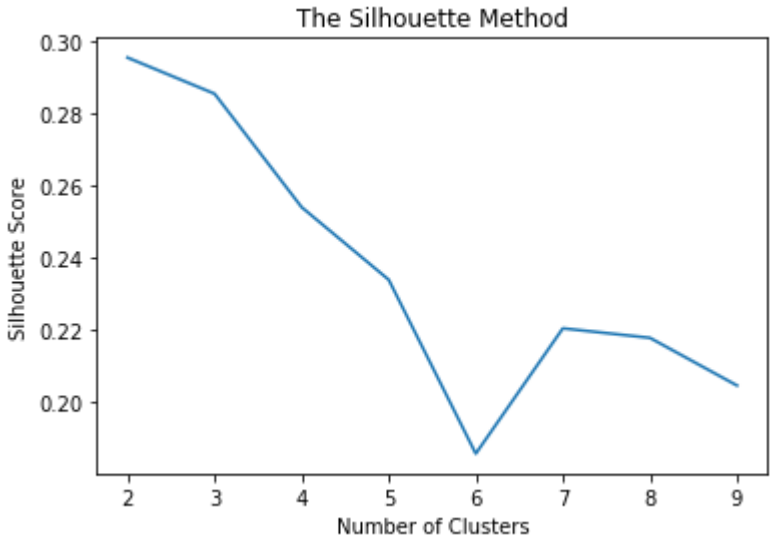
```
52.7515980317784
with k = 2
within Cluster SSE = 52.7515980317784   Silhouette Score = 0.2956852267976977
6
-------------------------------------------------------------------------------
-------------
41.09920607005725
with k = 3
within Cluster SSE = 41.09920607005725  Silhouette Score = 0.2856456827976892
-------------------------------------------------------------------------------
-------------
37.23675794052124
with k = 4
within Cluster SSE = 37.23675794052124  Silhouette Score = 0.2541300059940051
-------------------------------------------------------------------------------
-------------
34.944191014883245
with k = 5
within Cluster SSE = 34.944191014883245 Silhouette Score = 0.2339817626118071
1
-------------------------------------------------------------------------------
-------------
33.2289677354748
with k = 6
within Cluster SSE = 33.2289677354748   Silhouette Score = 0.1856584105513493
8
-------------------------------------------------------------------------------
-------------
31.120759184936603
with k = 7
within Cluster SSE = 31.120759184936603 Silhouette Score = 0.2204691213562588
7
-------------------------------------------------------------------------------
-------------
29.44752122276158
with k = 8
within Cluster SSE = 29.44752122276158  Silhouette Score = 0.2178279821893587
5
-------------------------------------------------------------------------------
-------------
27.809921556022537
with k = 9
within Cluster SSE = 27.809921556022537 Silhouette Score = 0.2045868027193378
5
-------------------------------------------------------------------------------
-------------
```

In [35]:
```
plt.plot(range(2,10), WCSSE)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Within cluster sum of squared errors')
plt.show()
```



The optimal point is k=3

In [36]:
```
plt.plot(range(2,10), silhouette)
plt.title('The Silhouette Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



The optimal point is k=3

7. Using appropriate metrics for your chosen algorithm, evaluate the trained model. Explain and justify the worthiness of your trained model.

In [37]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=3, random_state=20)
predictions = km.fit_predict(X1_train)

target = Y1_train['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_train['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_train['Cultivar_code'], predictions
)
print(completeness_coeff)


pd.crosstab(Y1_train['Cultivar_code'],predictions)
```

```
0.8353078853893485
0.8131942626970855
0.8060508732730011
```

Out[37]:

col_0	0	1	2
Cultivar_code			
0	44	0	1
1	2	5	50
2	0	40	0

Cultivar1 is distributed 98% to cluster0 and 2% to cluster2.

Cultivar2 is distributed 4% to cluster0,9% to cluster1 and 87% to cluster2

Cultivar3 is distributed 100% to cluster1

In [38]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=2, random_state=20)
predictions = km.fit_predict(X1_train)

target = Y1_train['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_train['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_train['Cultivar_code'], predictions
)
print(completeness_coeff)


pd.crosstab(Y1_train['Cultivar_code'],predictions)
```

```
0.4736119400659598
0.389251568852859
0.6235947909797085
```

Out[38]:

col_0	0	1
Cultivar_code		
0	45	0
1	38	19
2	0	40

Cultivar1 is distributed 100% to cluster0

Cultivar2 is distributed 67% to cluster0,33% to cluster1

Cultivar3 is distributed 100% to cluster1.

In [39]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=4, random_state=20)
predictions = km.fit_predict(X1_train)

target = Y1_train['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_train['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_train['Cultivar_code'], predictions
)
print(completeness_coeff)


pd.crosstab(Y1_train['Cultivar_code'],predictions)
```

```
0.6126760563380281
0.8972683189611219
0.7119698160583537
```

Out[39]:

col_0	0	1	2	3
Cultivar_code				
0	41	0	0	4
1	0	1	34	22
2	0	40	0	0

Cultivar1 is distributed 91% to cluster0,9% to cluster3

Cultivar2 is distributed 2% to cluster1,60% to cluster2 and 39% to cluster3

Cultivar3 is distributed 100% to cluster1

Worthiness of Train Model predictive performance metrics for different numer of clusters

In [40]:
```
--------------------------------------------------------------------------------
--------------
Metric                        k=2                     k=4                     k=3
--------------------------------------------------------------------------------
---------------
SSE                           52.7515980317784        37.23675794052124       41.0
9920607005725
Silhouette score              0.29568522679769776     0.2541300059940051      0.28
56456827976892
Silhouette coefficient        0.4736119400659598      0.6126760563380281      0.83
53078853893485
Homogeneity coefficient       0.389251568852859       0.897268318961121       0.81
31942626970855
Completeness coefficient      0.6235947909797085      0.7119698160583537      0.80
60508732730011
```

```
  File "<ipython-input-40-361edd469314>", line 1
    --------------------------------------------------------------------------
-------------------

    ^
SyntaxError: invalid syntax
```

comparing the predictive performance of all three models - with K=2, K=3, and K=4, The model with K=3 is the best among the three for the following reasons.

The above model is less than a perfect model beacuse all cultivars are not assigned to each cluster.it less than perfect model, hence misclassification or misaasignments would take place.

Comparing the silhouette scores, model K=2 and K=3 are about the same. K=4 has a much lower silhouette score indicating that there are quite a few missclassifcations .Cross tabulations of actual outcomes with predicted ones for K=4, there is only one cluster having complete cluster with cultivar2 ,cultivar0 is distrubted 91% to cluster0,9% to cluster3.cultivar1 is distributed 2% to clusters 1,60% to cluster2 and 39% to cluster3.Cross-tabulation of actual vs predicted for K=2 has simiar issues even though its silhouette score is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 67% and 33%.Cross-tabulation of actual vs predicted for K=3 . We have a very good distribution of cultivar0,2 completely assigned to 1,2 clusters.But cultivar1 is assigned partially between clusters 0,1,2 by 4%,9%,87%. Hence since K=3 has better distribution it is best for Silhouette Score.

Comparing only homogeneity score, the model with K=3 and K=4 are having less difference, but K=2 has the lowest homogeneity score.Cross tabulations of actual outcomes with predicted ones for K=4, there is only one cluster having complete cluster with cultivar2 ,cultivar0 is distrubted 91% to cluster0,9% to cluster3.cultivar1 is distributed 2% to clusters 1,60% to cluster2 and 39% to cluster3.Cross-tabulation of actual vs predicted for K=2 has simiar issues even though its silhouette score is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 67% and 33%.Cross-tabulation of actual vs predicted for K=3 . We have a very good distribution of cultivar0,2 completely assigned to 1,2 clusters.But cultivar1 is assigned partially between clusters 0,1,2 by 4%,9%,87%. Hence since K=3 has better distribution it is best for Homeogeneity Score as most of the clusters are assigned from single cultivar compared to number of clusters 2,4

Comparing the completeness score, K=4 model has the lowest score of three. so, it is definitely better than K=3 or K=2 models. The score for k=3 is the highest and higher of K=2 and K=3. Since the completeness score is based on all data points belonging to one class being assigned to the same cluster, this makes sense.Cross-tabulation of actual vs predicted for k=2 is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 67% and 33%.So the completeness of K=2 is higher than the compleness of K=3. Hence K=2 model is better distributed with a higher completeness score.But the distribution of records is very largely uneven,so K=2 cannot be used.

Thus we come to the conclusion that K=3 is the best model

For optimal clusters k=3

---------- Metric OriginalModel TrainModel ---------- SSE 48.97029115513917 41.09920607005725 Silhouette score 0.29936674064895064 0.2856456827976892 Silhouette coefficient 0.8366403391266617 0.8353078853893485 Homogeneity coefficient 0.8196302687581578 0.8131942626970855 Completeness coefficient 0.8113987824934515 0.8060508732730011

Hence we see that the TrainModel shows a very good performance Sihouette,Homegeneity and completeness coefficients in predicting the cultivar vavriables ditribution across the 13 constituent chemicals. The trained model performance is very similar to the Original model before test and train split.

8. Next, use the relevant portion of your dataset (as dictated by the chosen algorithm) to evaluate the performance of your model. Again, use all relevant metrics for your algorithm to discuss the outcome in terms of model's accuracy and usefulness in generating predictions. These may include such metrics as SSE, MSSE, Silhouette scores, completeness scores, confusion matrix, AOC curve, etc. as dictated by and available for your chosen machine language algorithm.

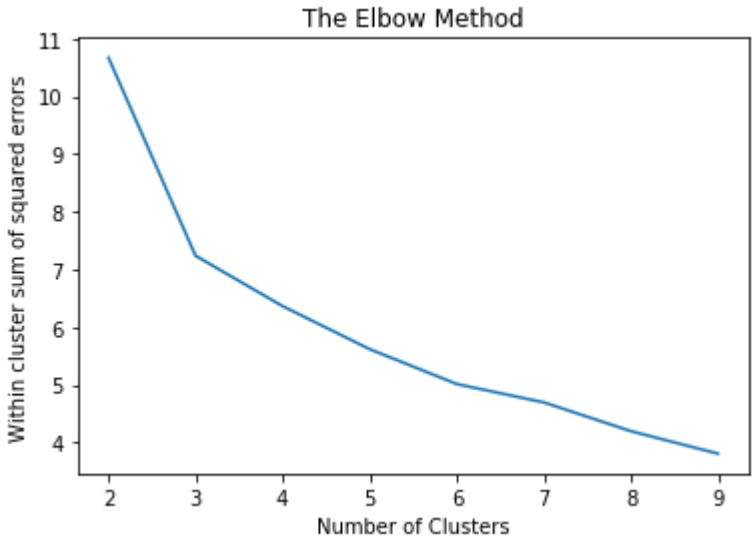
Test Model

In [41]:
```
from sklearn.metrics import silhouette_score

WCSSE = []
silhouette=[]
for k in range(2,10):
    km = KMeans(n_clusters = k, random_state=20)
    clusters = km.fit(X1_test)
    print(km.inertia_)
    WCSSE.append(km.inertia_)
    silhouette.append(silhouette_score(X1_test, clusters.labels_))
    print("with k = {}".format(k))
    print("within Cluster SSE = " +  str(WCSSE[k-2])  + '\t' + "Silhouette Sco
re = " + str(silhouette[k-2]))
    print("---"*30)
```

```
10.672389455779703
with k = 2
within Cluster SSE = 10.672389455779703 Silhouette Score = 0.3142734784737372
-------------------------------------------------------------------------------
-------------
7.237255374076926
with k = 3
within Cluster SSE = 7.237255374076926  Silhouette Score = 0.3573639864638751
-------------------------------------------------------------------------------
-------------
6.363069679410656
with k = 4
within Cluster SSE = 6.363069679410656  Silhouette Score = 0.2734934187216911
-------------------------------------------------------------------------------
-------------
5.617463175581721
with k = 5
within Cluster SSE = 5.617463175581721  Silhouette Score = 0.2206342514561685
2
-------------------------------------------------------------------------------
-------------
5.012721066373907
with k = 6
within Cluster SSE = 5.012721066373907  Silhouette Score = 0.2243987414039295
7
-------------------------------------------------------------------------------
-------------
4.693989074185378
with k = 7
within Cluster SSE = 4.693989074185378  Silhouette Score = 0.1816845403467402
5
-------------------------------------------------------------------------------
-------------
4.19556946523607
with k = 8
within Cluster SSE = 4.19556946523607   Silhouette Score = 0.1991410778264795
5
-------------------------------------------------------------------------------
-------------
3.8059051270021724
with k = 9
within Cluster SSE = 3.8059051270021724 Silhouette Score = 0.1806906716208057
4
-------------------------------------------------------------------------------
-------------
```

In [42]:
```
plt.plot(range(2,10), WCSSE)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Within cluster sum of squared errors')
plt.show()
```



The optimal point is k=3

In [43]:
```
plt.plot(range(2,10), silhouette)
plt.title('The Silhouette Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```

The optimal point is k=3

In [44]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=3, random_state=20)
predictions = km.fit_predict(X1_test)

target = Y1_test['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_test['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_test['Cultivar_code'], predictions)
print(completeness_coeff)


pd.crosstab(Y1_test['Cultivar_code'],predictions)
```

```
0.9166666666666666
0.9184072443788571
0.9074881047446167
```

Out[44]:

col_0	0	1	2
Cultivar_code			
0	0	0	14
1	13	1	0
2	0	8	0

Cultivar1 is completely assigned to cluster2 100%.

Cultivar2 is distributed 7% to clusters 1 and 93% to cluster0

Cultivar3 is completely assigned to cluster1 100%.

In [45]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=4, random_state=20)
predictions = km.fit_predict(X1_test)

target = Y1_test['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_test['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_test['Cultivar_code'], predictions)
print(completeness_coeff)


pd.crosstab(Y1_test['Cultivar_code'],predictions)
```

```
0.4722222222222222
0.8400716501766841
0.6575794344347402
```

Out[45]:

col_0	0	1	2	3
Cultivar_code				
0	7	7	0	0
1	0	1	1	12
2	0	0	8	0

Cultivar1 is distrubted 50% each between clusters 0,1.

Cultivar2 is distributed 7% each to clusters 1,2 and 86% to cluster3

Cultivar3 is completely assigned to cluster2 100%.

Metrics

In [46]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score

km = KMeans(n_clusters=2, random_state=20)
predictions = km.fit_predict(X1_test)

target = Y1_test['Cultivar_code'].to_numpy(dtype='Int64').reshape(-1,1)
silhouette_coeff = silhouette_score(target, predictions)
print(silhouette_coeff)

homogeneity_coeff = homogeneity_score(Y1_test['Cultivar_code'], predictions)
print(homogeneity_coeff)

completeness_coeff = completeness_score(Y1_test['Cultivar_code'], predictions)
print(completeness_coeff)


pd.crosstab(Y1_test['Cultivar_code'],predictions)
```

```
0.6404065719855194
0.4935083224653958
0.7678331878117991
```

Out[46]:

col_0	0	1
Cultivar_code		
0	14	0
1	2	12
2	0	8

Cultivar1 is completely assigned to cluster0 100%.

Cultivar2 is distributed 14% each to clusters 0 and 86% to cluster1

Cultivar3 is completely assigned to cluster1 100%.

Comparition of Test Model predictive performance metrics for different numer of clusters

In [47]:
```
------------------------------------------------------------------------------
--------------
Metric                      k=2                  k=4
k=3
------------------------------------------------------------------------------
--------------
SSE                         10.672389455779703   6.363069679410656        7.2
37255374076926
Silhouette score            0.3142734784737372   0.2734934187216911       0.3
573639864638751
Silhouette coefficient      0.6404065719855194   0.4722222222222222       0.9
166666666666666
Homogeneity coefficient     0.4935083224653958   0.8400716501766841       0.9
184072443788571
Completeness coefficient    0.7678331878117991   0.6575794344347402       0.9
074881047446167
```

```
  File "<ipython-input-47-509f9c1eea10>", line 1
    ------------------------------------------------------------------------
------------------

^
SyntaxError: invalid syntax
```

comparing the predictive performance of all three models - with K=2, K=3, and K=4, The model with K=3 is the best among the three for the following reasons.

The above model is less than a perfect model beacuse all cultivars are not assigned to each cluster.it less than perfect model, hence misclassification or misaasignments would take place.

Comparing the silhouette scores, model K=2 and K=3 are about the same. K=4 has a much lower silhouette score indicating that there are quite a few missclassifcations .Cross tabulations of actual outcomes with predicted ones for K=4, there is only one cluster having complete cluster with cultivar2 ,cultivar0 is distrubted 50% each between clusters 0,1.cultivar1 is distributed 7% each to clusters 1,2 and 86% to cluster3.Cross-tabulation of actual vs predicted for K=2 has simiar issues even though its silhouette score is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 14% and 86%.Cross-tabulation of actual vs predicted for K=3 . We have a very good distribution of cultivar0,2 completely assigned to 1,2 clusters.But cultivar1 is assigned partially between 1,0 by 7% and 93%. Hence since K=3 has better distribution it is best for Silhouette Score

Comparing only homogeneity score, the model with K=3 and K=4 are having less difference, but K=2 has the lowest homogeneity score.Cross tabulations of actual outcomes with predicted ones for K=4, there is only one cluster having complete cluster with cultivar2 ,cultivar0 is distrubted 50% each between clusters 0,1.cultivar1 is distributed 7% each to clusters 1,2 and 86% to cluster3.Cross-tabulation of actual vs predicted for K=2 is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 14% and 86%.Cross-tabulation of actual vs predicted for K=3 .We have a very good distribution of cultivar0,2 completely assigned to 1,2 clusters.But cultivar1 is assigned partially between 1,0 by 7% and 93%. Hence since K=3 has better distribution it is best for Homeogeneity Score as most of the clusters are assigned from single cultivar compared to number of clusters 2,4

Comparing the completeness score, K=4 model has the lowest score of three. so, it is definitely better than K=3 or K=2 models. The score for k=3 is the highest and higher of K=2 and K=3. Since the completeness score is based on all data points belonging to one class being assigned to the same cluster, this makes sense.Cross-tabulation of actual vs predicted for k=2 is similar to that of K= 3. We have a good distribution of cultivar0,2 completely assigned to 0,1 clusters.But cultivar1 is assigned partially between 0,1 by 14% and 86%.So the complteness of K=2 is lower than the compleness of K=4 which is better distributed with a higher completeness score. Therefore, K=2 model is not very good at clustering.And K=3 is the best model.

Thus we come to the conclusion that K=3 is the best model

For optimal clusters k=3

In [48]:
```
--------------------------------------------------------------------------------
------------
Metric                      OriginalModel          TrainModel
TestModel
--------------------------------------------------------------------------------
-------------
SSE                         48.97029115513917      41.09920607005725
7.237255374076926
Silhouette score            0.29936674064895064    0.2856456827976892
0.3573639864638751
Silhouette coefficient      0.8366403391266617     0.8353078853893485
0.9166666666666666
Homogeneity coefficient     0.8196302687581578     0.8131942626970855
0.9184072443788571
Completeness coefficient    0.8113987824934515     0.8060508732730011
0.9074881047446167
```

```
  File "<ipython-input-48-16afd51a7a77>", line 1
    ----------------------------------------------------------------------------
----------------

^
SyntaxError: invalid syntax
```

Hence we see that the TestModel shows a very good performance Sihouette,Homegeneity and completeness coefficients. Where are the predictions in the Trained model are a little lower compared to the actual performance of test model. The trained model performance is very similar to the Original model before test and train split.

Logistic Regression

In [49]:
```
WineCorr=WineNumericDF.corr()
WineCorr
```

Out[49]:

	Alcohol	MalicAcid	Ash	Alkalinity	Magnesium	Phenols	Flavanoids	Nc
Alcohol	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	
MalicAcid	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	
Ash	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	
Alkalinity	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	
Magnesium	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	
Phenols	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	
Flavanoids	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	
NonFlavanoids	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	
Pcyanins	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	
ColorIntensity	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	
Hue	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	
OD280	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	
Proline	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	
Cultivar_code	-0.328222	0.437776	-0.049643	0.517859	-0.209179	-0.719163	-0.847498	

Correlation of all the variables this output is similar to that of heatmap where we can see the correlations of each attribute

In [50]:
```
corr_target = abs(WineCorr['Cultivar_code'])
relevant_features = corr_target[corr_target>0.5]
relevant_features
```

Out[50]:
```
Alkalinity      0.517859
Phenols         0.719163
Flavanoids      0.847498
Hue             0.617369
OD280           0.788230
Proline         0.633717
Cultivar_code   1.000000
Name: Cultivar_code, dtype: float64
```

Correlation with output variable Cultivar_code is seen,the above variable are the most highly correlated variables.We filter these for those variables with correlation value greater than 0.5 so the values closer to 1 are highly correlated.

In [51]:
```
print(Wine_DF[["Alkalinity","Phenols"]].corr())
```

```
            Alkalinity   Phenols
Alkalinity    1.000000 -0.321113
Phenols      -0.321113  1.000000
```

Chemicals Alkalinity and Phenols are ver lowly correlated with negative values

In [52]:
```
print(Wine_DF[["Flavanoids","NonFlavanoids"]].corr())
```

```
                Flavanoids  NonFlavanoids
Flavanoids          1.0000        -0.5379
NonFlavanoids      -0.5379         1.0000
```

Chemicals Flavanoids and Non Flavanoids are ver lowly correlated with negative values

In [53]:
```
print(Wine_DF[["Pcyanins","Hue"]].corr())
```

```
           Pcyanins       Hue
Pcyanins   1.000000  0.295544
Hue        0.295544  1.000000
```

Chemicals Pcyanins and Hue are lowly correlated with values very close to 0

In [54]:
```
print(Wine_DF[["OD280","Proline"]].corr())
```

```
             OD280   Proline
OD280    1.000000  0.312761
Proline  0.312761  1.000000
```

Chemicals 0280 and Proline are lowly correlated with values very close to 0

In [55]:
```
FeaturesDF=WineNumericDF[["Alkalinity","Phenols","Flavanoids","NonFlavanoids",
"Pcyanins","Hue","OD280","Proline"]]
FeaturesDF.head()
```

Out[55]:

	Alkalinity	Phenols	Flavanoids	NonFlavanoids	Pcyanins	Hue	OD280	Proline
0	0.257732	0.627586	0.573840	0.283019	0.593060	0.455285	0.970696	0.561341
1	0.030928	0.575862	0.510549	0.245283	0.274448	0.463415	0.780220	0.550642
2	0.412371	0.627586	0.611814	0.320755	0.757098	0.447154	0.695971	0.646933
3	0.319588	0.989655	0.664557	0.207547	0.558360	0.308943	0.798535	0.857347
4	0.536082	0.627586	0.495781	0.490566	0.444795	0.455285	0.608059	0.325963

For train and test split we divide the input data for prediction into Features dataframe

In [56]:
```
TargetDF=WineNumericDF[['Cultivar_code']]
TargetDF.head()
```

Out[56]:

	Cultivar_code
0	0
1	0
2	0
3	0
4	0

For train and test split we divide the target data for prediction into Target dataframe

In [57]:
```
X_train, X_test, Y_train, Y_test = train_test_split(FeaturesDF,TargetDF, test_
size=0.2, random_state=42)
X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
```

Out[57]: ((142, 8), (142, 1), (36, 8), (36, 1))

we split the data in to 80,20 ratio

In [58]:
```
from sklearn.linear_model import LogisticRegression
```

In [59]:
```
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)
C:\Users\Deepthi\Anaconda3-3.7\lib\site-packages\sklearn\linear_model\logisti
c.py:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.2
2. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

Out[59]:
```
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 [60]:
```
Y_predict = logmodel.predict(X_test)
```

predicting the target variable

In [61]:
```
predict_y = Y_predict.reshape(-1,1)
print(predict_y.shape)
```

```
(36, 1)
```

predicted data count of records

In [62]:
```
test_y = (Y_test.values).reshape(-1,1)
print(test_y.shape)
print(test_y.size)
```

```
(36, 1)
36
```

actual data count of records

In [63]:
```
predicted_probs = logmodel.predict_proba(X_test)
print(predicted_probs)
```

```
[[0.63140156 0.35001121 0.01858724]
 [0.7204612  0.22125765 0.05828115]
 [0.06135956 0.33883236 0.59980808]
 [0.74242    0.20988357 0.04769643]
 [0.24974651 0.70617216 0.04408132]
 [0.77274257 0.15608961 0.07116782]
 [0.22089182 0.64403173 0.13507644]
 [0.07085264 0.20100392 0.72814344]
 [0.41517643 0.54925174 0.03557184]
 [0.15337141 0.27672712 0.56990147]
 [0.48426271 0.49691096 0.01882633]
 [0.05105542 0.31289395 0.63605063]
 [0.66400896 0.25915818 0.07683287]
 [0.0979555  0.43656841 0.46547609]
 [0.79357125 0.17339402 0.03303473]
 [0.16037803 0.72536502 0.11425695]
 [0.12037008 0.6801415  0.19948843]
 [0.11977708 0.72902496 0.15119796]
 [0.7201213  0.26563973 0.01423897]
 [0.14133147 0.74114452 0.117524  ]
 [0.92387101 0.05536325 0.02076573]
 [0.4239555  0.56226568 0.01377882]
 [0.13293972 0.35726298 0.5097973 ]
 [0.07701541 0.32707487 0.59590972]
 [0.04024206 0.25134527 0.70841266]
 [0.05187975 0.27081233 0.67730792]
 [0.10232581 0.79290278 0.10477141]
 [0.33551552 0.56318316 0.10130132]
 [0.09178199 0.6047716  0.30344641]
 [0.7063739  0.25198175 0.04164435]
 [0.67521531 0.30209428 0.02269042]
 [0.08376302 0.74956358 0.16667341]
 [0.09680927 0.14773476 0.75545598]
 [0.86982683 0.09308602 0.03708715]
 [0.8107897  0.14883765 0.04037265]
 [0.79117046 0.19661685 0.01221269]]
```

predicted probabilites of the target variable

In [175]:
```
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[175]:

	0	1	2	Predicted	Actual
0	0.631402	0.350011	0.018587	0	0
1	0.720461	0.221258	0.058281	0	0
2	0.061360	0.338832	0.599808	2	2
3	0.742420	0.209884	0.047696	0	0
4	0.249747	0.706172	0.044081	1	1
5	0.772743	0.156090	0.071168	0	0
6	0.220892	0.644032	0.135076	1	1
7	0.070853	0.201004	0.728143	2	2
8	0.415176	0.549252	0.035572	1	1
9	0.153371	0.276727	0.569901	2	2
10	0.484263	0.496911	0.018826	1	0
11	0.051055	0.312894	0.636051	2	2
12	0.664009	0.259158	0.076833	0	0
13	0.097956	0.436568	0.465476	2	1
14	0.793571	0.173394	0.033035	0	0

In [176]:
```
prob_results_df.groupby("Predicted").size()
```

Out[176]:
```
Predicted
0    13
1    13
2    10
dtype: int64
```

In [189]:
```
prob_results_df.groupby("Actual").size()
```

Out[189]:
```
Actual
0    14
1    14
2     8
dtype: int64
```

We see that Predicted values are cultivar0 is 93% of actual test count 14,cultivar1 is 93% of actual test count 14,cultivar2 is 80% of actual test count 8

In [178]:
```
from sklearn.metrics import classification_report
from sklearn import metrics

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

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        14
           1       0.92      0.86      0.89        14
           2       0.80      1.00      0.89         8

    accuracy                           0.92        36
   macro avg       0.91      0.93      0.91        36
weighted avg       0.93      0.92      0.92        36
```

In [179]:
```
conf_matrix = metrics.confusion_matrix(Y_test, Y_predict)
conf_matrix
```

Out[179]:
```
array([[13,  1,  0],
       [ 0, 12,  2],
       [ 0,  0,  8]], dtype=int64)
```

In [180]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_precision_score(Y_test, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y_test)
    test = lb.transform(Y_test)
    predict = lb.transform(Y_predict)
    return precision_score(test, predict, average=average)
precision=multiclass_precision_score(Y_test, Y_predict, average="macro")
print("Precision score: ", precision)
```

```
Precision score:  0.9076923076923077
```

In [181]:
```
from sklearn.metrics import recall_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_recall_score(Y_test, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y1_test)
    test = lb.transform(Y1_test)
    pred = lb.transform(Y_predict)
    return recall_score(test, pred, average=average)
recall=multiclass_recall_score(Y_test, Y_predict, average="macro")
print("Recall score: ", recall)
```

```
Recall score:  0.9285714285714285
```

In [182]:
```
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_roc_auc_score(Y_test, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y1_test)
    test = lb.transform(Y1_test)
    pred = lb.transform(Y_predict)
    return roc_auc_score(test, pred, average=average)
auc = multiclass_roc_auc_score(Y1_test, Y_predict, average="macro")
print("Area under curve : ", auc)
```

```
Area under curve :  0.9448051948051949
```

In [183]:
```
print("**********************************************************************")
print("Logistic Regression Results")
print("**********************************************************************")
print("Accuracy:",metrics.accuracy_score(Y_test, Y_predict))
print("Precision:",precision)
print("Recall:",recall)
print("AUC:",auc)
print("**********************************************************************")
```

```
**********************************************************************
Logistic Regression Results
**********************************************************************
Accuracy: 0.9166666666666666
Precision: 0.9076923076923077
Recall: 0.9285714285714285
AUC: 0.9448051948051949
**********************************************************************
```

Logistic Regression trained model with test data is very good in prediction the cultivar variable distribution against 8 highly correlated variables with target cultivar variable.Model has high accuarcy Accuracy of 91.6% accuracy,high precision rate of 90.7,high recall rate of 92.8%.Higher the AUC better the model thus the model is also verified with high auc os 0.944

In [194]:
```
logmodel.fit(X_test,Y_test)
Y_predict = logmodel.predict(X_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)
C:\Users\Deepthi\Anaconda3-3.7\lib\site-packages\sklearn\linear_model\logisti
c.py:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.2
2. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [195]:
```
predict_y = Y_predict.reshape(-1,1)
print(predict_y.shape)
```

```
(142, 1)
```

In [196]:
```
train_y = (Y_train.values).reshape(-1,1)
print(train_y.shape)
print(train_y.size)
```

```
(142, 1)
142
```

In [197]:
```
predicted_probs = logmodel.predict_proba(X_train)
print(predicted_probs)
```

```
[[0.20258951 0.44513374 0.35227675]
 [0.07782709 0.46866571 0.45350719]
 [0.53840168 0.39717975 0.06441857]
 [0.18842751 0.43416042 0.37741206]
 [0.56455781 0.32455449 0.1108877 ]
 [0.26311726 0.57577649 0.16110625]
 [0.33963926 0.53982041 0.12054033]
 [0.2467311  0.46055783 0.29271107]
 [0.17724696 0.4754905  0.34726254]
 [0.65159173 0.27671184 0.07169642]
 [0.44556276 0.43936671 0.11507053]
 [0.19663727 0.59137018 0.21199254]
 [0.13066482 0.38359123 0.48574395]
 [0.68356378 0.25384942 0.06258681]
 [0.33045521 0.51683942 0.15270537]
 [0.62712231 0.30488131 0.06799638]
 [0.64229851 0.26327992 0.09442157]
 [0.14072965 0.40920464 0.45006571]
 [0.11847788 0.45169876 0.42982337]
 [0.39949545 0.46451222 0.13599233]
 [0.39781752 0.43691839 0.16526409]
 [0.5087589  0.37871672 0.11252438]
 [0.34746482 0.55957066 0.09296451]
 [0.62356131 0.31861657 0.05782212]
 [0.10118219 0.37833452 0.52048329]
 [0.37392739 0.52755953 0.09851308]
 [0.479866   0.40627522 0.11385878]
 [0.19443027 0.36479063 0.4407791 ]
 [0.63399114 0.2474985  0.11851037]
 [0.67824424 0.24499109 0.07676467]
 [0.62916225 0.22877392 0.14206383]
 [0.30584484 0.2868706  0.40728456]
 [0.4141015  0.46008543 0.12581307]
 [0.49106196 0.40386897 0.10506907]
 [0.50527957 0.38572043 0.109     ]
 [0.14823409 0.39532551 0.45644041]
 [0.24507556 0.53445936 0.22046508]
 [0.75660181 0.18977499 0.0536232 ]
 [0.15209764 0.37119173 0.47671063]
 [0.34165475 0.50852754 0.14981771]
 [0.64660887 0.30340994 0.04998118]
 [0.11342532 0.45787772 0.42869695]
 [0.29757838 0.54038611 0.1620355 ]
 [0.18159337 0.62559467 0.19281196]
 [0.58568182 0.33529364 0.07902454]
 [0.33418259 0.37181373 0.29400368]
 [0.52233294 0.40780555 0.06986151]
 [0.54844634 0.38965208 0.06190159]
 [0.12939692 0.58737932 0.28322376]
 [0.37135696 0.49285222 0.13579082]
 [0.58921307 0.32134344 0.08944349]
 [0.16365367 0.33607559 0.50027074]
 [0.45937177 0.43188997 0.10873826]
 [0.29260942 0.59545539 0.11193519]
 [0.42219656 0.43625572 0.14154772]
 [0.60337267 0.3398901  0.05673723]
 [0.29836153 0.47021638 0.23142209]
```

```
[0.19225942 0.57749788 0.2302427 ]
[0.45339968 0.43662195 0.10997836]
[0.16115583 0.36924668 0.46959749]
[0.11090017 0.48181018 0.40728965]
[0.65687702 0.28229444 0.06082854]
[0.41122737 0.49622836 0.09254427]
[0.12460868 0.41687824 0.45851309]
[0.17628426 0.39587336 0.42784238]
[0.15473056 0.51554404 0.32972539]
[0.56821604 0.35338822 0.07839574]
[0.5458024  0.34935165 0.10484594]
[0.30910764 0.54893933 0.14195303]
[0.26058012 0.32322113 0.41619875]
[0.14614569 0.47329882 0.38055549]
[0.33745398 0.47280024 0.18974578]
[0.18105295 0.34181643 0.47713062]
[0.18380694 0.58618563 0.23000743]
[0.20957542 0.69581474 0.09460985]
[0.11608646 0.64355892 0.24035461]
[0.65951232 0.24941217 0.0910755 ]
[0.72764203 0.20456768 0.06779029]
[0.14764051 0.41531658 0.43704291]
[0.78979494 0.16236305 0.04784201]
[0.09969264 0.43684609 0.46346127]
[0.67011826 0.26312973 0.06675201]
[0.5678839  0.34733591 0.08478019]
[0.29724383 0.60579328 0.09696289]
[0.21811353 0.50911826 0.27276821]
[0.56483013 0.30512522 0.13004465]
[0.65556173 0.29118299 0.05325528]
[0.65935005 0.24849389 0.09215605]
[0.44269204 0.47771443 0.07959352]
[0.45513955 0.41246444 0.13239601]
[0.25013898 0.34588274 0.40397828]
[0.12124317 0.36459216 0.51416467]
[0.47843579 0.4223047  0.0992595 ]
[0.15096289 0.56813837 0.28089874]
[0.11862051 0.52133853 0.36004096]
[0.14230158 0.42782601 0.42987241]
[0.16950962 0.43605317 0.3944372 ]
[0.27812594 0.52414351 0.19773055]
[0.61539452 0.30594437 0.07866111]
[0.62252377 0.33144128 0.04603495]
[0.30413343 0.42812259 0.26774398]
[0.18383    0.3853072  0.4308628 ]
[0.21179464 0.3583645  0.42984086]
[0.5644634  0.32868572 0.10685089]
[0.26945557 0.48723248 0.24331195]
[0.11911306 0.38447681 0.49641013]
[0.14391003 0.43474152 0.42134845]
[0.17114224 0.3633356  0.46552217]
[0.1958846  0.3611132  0.4430022 ]
[0.46032739 0.4782893  0.06138332]
[0.64072131 0.26553651 0.09374218]
[0.16262392 0.52179482 0.31558125]
[0.69739647 0.25040654 0.05219699]
[0.21238376 0.31087419 0.47674205]
```

```
[0.74088489 0.19306522 0.06604989]
[0.57707861 0.32220201 0.10071938]
[0.25180181 0.50151954 0.24667866]
[0.51042506 0.39453873 0.09503621]
[0.66962664 0.24631791 0.08405546]
[0.1798552  0.37109213 0.44905267]
[0.24825866 0.47190569 0.27983565]
[0.56040329 0.29677766 0.14281904]
[0.11933196 0.35562615 0.52504189]
[0.19202105 0.35690867 0.45107028]
[0.64120559 0.29016849 0.06862592]
[0.70361824 0.25473923 0.04164252]
[0.17363506 0.35576325 0.47060169]
[0.27374334 0.32554561 0.40071105]
[0.20000546 0.34101899 0.45897555]
[0.25989624 0.52509794 0.21500582]
[0.3108126  0.62327095 0.06591645]
[0.31807617 0.50173895 0.18018487]
[0.23539472 0.58267607 0.18192921]
[0.6072593  0.29620963 0.09653108]
[0.34403894 0.58521886 0.0707422 ]
[0.10847385 0.42003118 0.47149497]
[0.57730295 0.37429566 0.04840139]
[0.37853087 0.54152514 0.07994399]
[0.29471316 0.52796654 0.1773203 ]
[0.77375592 0.18924774 0.03699634]
[0.1250458  0.53767019 0.33728401]
[0.32731297 0.51105552 0.16163151]]
```

In [198]:
```
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"] = train_y
prob_results_df.head(15)
```

Out[198]:

	0	1	2	Predicted	Actual
0	0.202590	0.445134	0.352277	1	2
1	0.077827	0.468666	0.453507	1	2
2	0.538402	0.397180	0.064419	0	1
3	0.188428	0.434160	0.377412	1	2
4	0.564558	0.324554	0.110888	0	0
5	0.263117	0.575776	0.161106	1	1
6	0.339639	0.539820	0.120540	1	1
7	0.246731	0.460558	0.292711	1	1
8	0.177247	0.475491	0.347263	1	2
9	0.651592	0.276712	0.071696	0	0
10	0.445563	0.439367	0.115071	0	1
11	0.196637	0.591370	0.211993	1	1
12	0.130665	0.383591	0.485744	2	2
13	0.683564	0.253849	0.062587	0	0
14	0.330455	0.516839	0.152705	1	1

In [199]:
```
prob_results_df.groupby("Predicted").size()
```

Out[199]:
```
Predicted
0    52
1    59
2    31
dtype: int64
```

In [200]:
```
prob_results_df.groupby("Actual").size()
```

Out[200]:
```
Actual
0    45
1    57
2    40
dtype: int64
```

We see that Predicted values are cultivar0 is 87% of actual train count 14,cultivar1 is 97% of actual test count 14,cultivar2 is 78% of actual train count 8

In [201]:
```
from sklearn.metrics import classification_report
from sklearn import metrics

print(classification_report(Y_train,Y_predict))
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89        45
           1       0.80      0.82      0.81        57
           2       0.97      0.75      0.85        40

    accuracy                           0.85       142
   macro avg       0.86      0.84      0.85       142
weighted avg       0.85      0.85      0.84       142
```

In [203]:
```
conf_matrix = metrics.confusion_matrix(Y_train, Y_predict)
conf_matrix
```

Out[203]:
```
array([[43,  2,  0],
       [ 9, 47,  1],
       [ 0, 10, 30]], dtype=int64)
```

In [ ]:
```

```

In [205]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_precision_score(Y_train, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y_train)
    train = lb.transform(Y_train)
    predict = lb.transform(Y_predict)
    return precision_score(train, predict, average=average)
precision=multiclass_precision_score(Y_train, Y_predict, average="macro")
print("Precision score: ", precision)
```

```
Precision score:  0.8637583939661577
```

In [206]:
```
from sklearn.metrics import recall_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_recall_score(Y_train, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y1_test)
    train = lb.transform(Y1_train)
    pred = lb.transform(Y_predict)
    return recall_score(train, pred, average=average)
recall=multiclass_recall_score(Y_train, Y_predict, average="macro")
print("Recall score: ", recall)
```

```
Recall score:  0.8433723196881092
```

In [207]:
```
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_roc_auc_score(Y_train, Y_predict, average="macro"):
    lb = LabelBinarizer()
    lb.fit(Y1_train)
    train = lb.transform(Y1_train)
    pred = lb.transform(Y_predict)
    return roc_auc_score(train, pred, average=average)
auc = multiclass_roc_auc_score(Y1_train, Y_predict, average="macro")
print("Area under curve : ", auc)
```

```
Area under curve :  0.881058843625471
```

In [208]:
```
print("**********************************************************************")
print("Logistic Regression Results")
print("**********************************************************************")
print("Accuracy:",metrics.accuracy_score(Y_train, Y_predict))
print("Precision:",precision)
print("Recall:",recall)
print("AUC:",auc)
print("**********************************************************************")
```

```
**********************************************************************
Logistic Regression Results
**********************************************************************
Accuracy: 0.8450704225352113
Precision: 0.8637583939661577
Recall: 0.8433723196881092
AUC: 0.881058843625471
**********************************************************************
```

TrainedModel with train data is predicting the results good with a high accuracy of 84.5,high Precision rate 86.3%,high recall rate 84.3% .This model also has ahigh auc of 0.88 closer to 1 which tells that this test model is verified as good model.

In [212]:
```
-----------------------------------------------------------------------------
---------------------------------
Metric            TestModel                                 TrainModel
-----------------------------------------------------------------------------
---------------------------------
Accuracy          0.9166666666666666                         0.845070422535211
3
Precision         0.9076923076923077                         0.863758393966157
7
Recall            0.9285714285714285                         0.843372319688109
2
AUC               0.9448051948051949                         0.881058843625471
-----------------------------------------------------------------------------
---------------------------------
```

```
  File "<ipython-input-212-a213b3dc0e7a>", line 1
    ------------------------------------------------------------------------
----------------------------------------

^
SyntaxError: invalid syntax
```

As per the comparision above both the models have good results,however the trained model with test data performs better than the trained model train data with higher metrics of accuarcy,precision rate,recall rate and auc.

In [ ]: