An Assignment Report

On

Model Evaluation, Training & Testing (Supervised & Unsupervised Learning Model)

by

Aditya Sharma
AU18B1009

Under the guidance of

Satyajit Pangaonkar

School of Engineering

Avantika University, Ujjain

2020-2021

Content: **Page**

Assignment No-02 (CS 6301- Machine Learning)

Title : Model Evaluation, Training & Testing

Statement :

Based on the available dataset, perform the prediction analysis. Based on the method described in the class, model the data class. Use Supervised Learning algorithm (like Support Vector Machine) and Unsupervised Learning Algorithm (Hierarchical Clustering) for the operation. Check the performance based on the analysis of algorithm used.

Supervised Learning

a. Identification of the Dataset :

I. Type of the Dataset : Multivariate, Structured dataset.

The dataset is related to red variant of the Portuguese "Vinho Verde" wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

II. Data Quality and Analysis :

Input Variables:

- ✓ **Fixed acidity:** most acids involved with wine or fixed or non-volatile
- ✓ **Volatile acidity:** the amount of acetic acid in wine
- ✓ **Citric acid:** found in small quantities, citric acid can add 'freshness' and flavour to wines
- ✓ **Residual sugar:** the amount of sugar remaining after fermentation stops
- ✓ **Chlorides:** the amount of salt in the wine
- ✓ **Free sulphur dioxide**: the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfide ion
- ✓ **Total sulphur dioxide:** amount of free and bound forms of S02
- ✓ **Density:** the density of water is close to that of water depending on the percent alcohol and sugar content

- ✓ **pH:** describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic)
- ✓ **Sulphates:** a wine additive which can contribute to sulfur dioxide gas (S02) levels
- ✓ **Alcohol:** the percent alcohol content of the wine

Output Variable:

- ✓ **Quality:** target variable (score between 0 and 10, expect output 'good' / 'bad')

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

Fig. 1 Red wine data

```
In [594]: wine.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Fig. 2 Concise summary of the dataframe red wine data

```
In [598]: wine.isnull().sum()

Out[598]: fixed acidity           0
          volatile acidity        0
          citric acid             0
          residual sugar          0
          chlorides               0
          free sulfur dioxide     0
          total sulfur dioxide    0
          density                 0
          pH                      0
          sulphates               0
          alcohol                 0
          quality                 0
          dtype: int64
```

Fig. 3 Number of missing values in the data set

```
In [602]: sn.displot(wine['quality'])

Out[602]: <seaborn.axisgrid.FacetGrid at 0x2ad46a99e50>
```

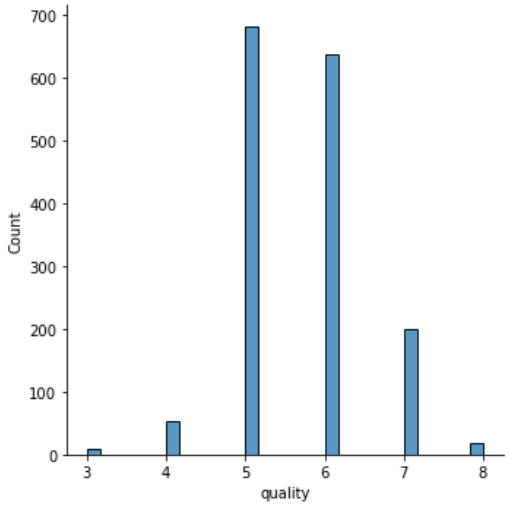


Fig. 4 Countplot for quality of wine

```
In [599]: wine['quality'].value_counts()

Out[599]: 5    681
          6    638
          7    199
          4     53
          8     18
          3     10
          Name: quality, dtype: int64
```

Fig. 5 Count of quality of wine

```
In [593]: wine.shape

Out[593]: (1599, 12)
```

Fig. 6 Number of rows and columns

```
In [664]: wine.tail()

Out[664]:
```

	fixed acidity	volatile acidity	citric acid	chlorides	free sulfur dioxide	total sulfur dioxide	sulphates	alcohol	quality
1594	6.2	0.600	0.08	0.090	32.0	44.0	0.58	10.5	0
1595	5.9	0.550	0.10	0.062	39.0	51.0	0.76	11.2	0
1596	6.3	0.510	0.13	0.076	29.0	40.0	0.75	11.0	0
1597	5.9	0.645	0.12	0.075	32.0	44.0	0.71	10.2	0
1598	6.0	0.310	0.47	0.067	18.0	42.0	0.66	11.0	0

Fig. 4 Last 5 values in dataset

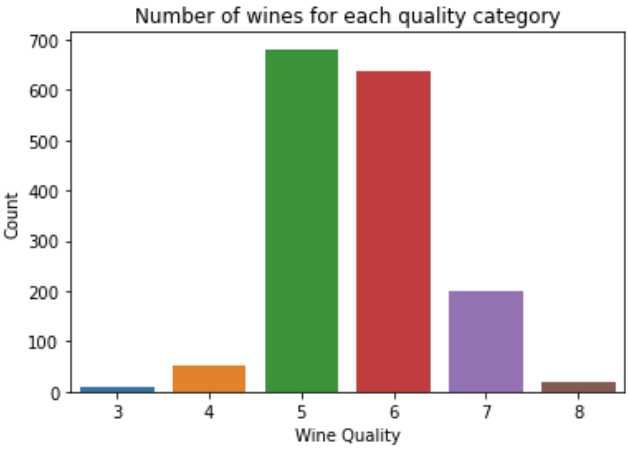


Fig. 7 Count plot for quality of wine

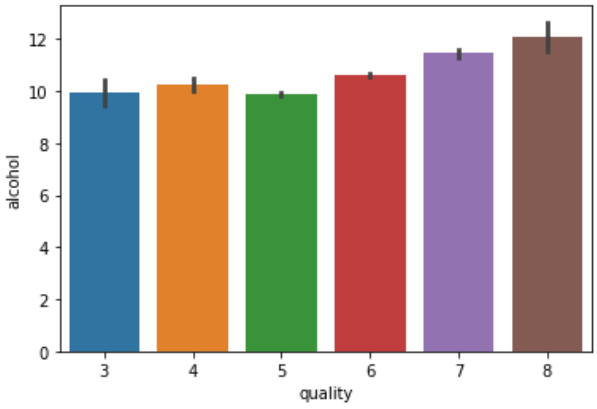


Fig. 8 Quality vs alcohol

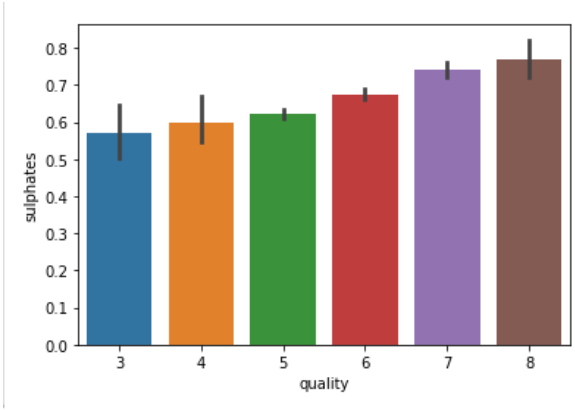


Fig. 9 Quality vs sulphates

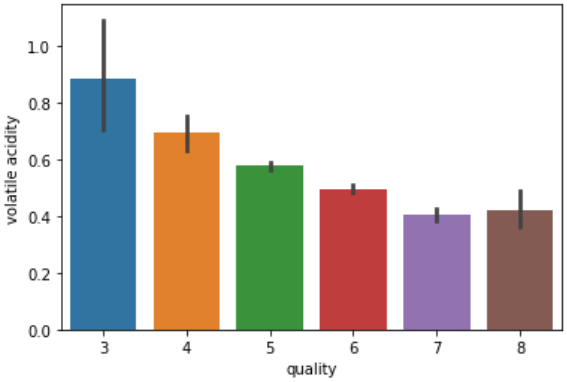


Fig. 10 Quality vs volatile acidity

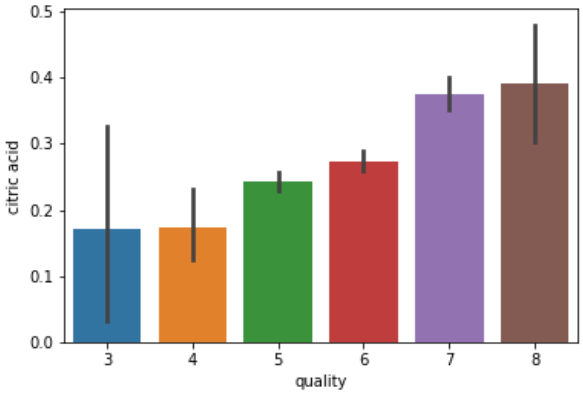


Fig. 11 Quality vs citric acid

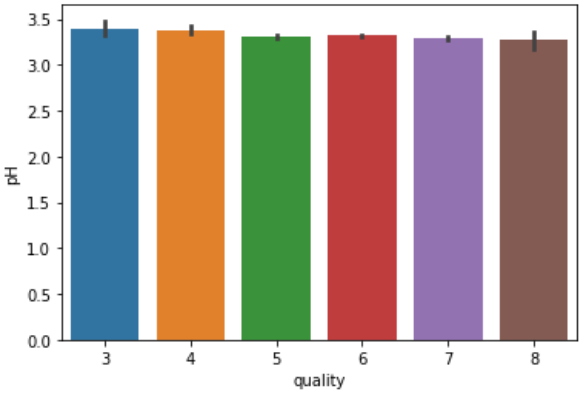


Fig. 12 Quality vs pH

From the above inference it is clear that pH, density, residual sugar, fixed acidity does not affect the quality of wine.

- ✓ **Composition of citric acid go higher as we go higher in the quality of the wine**
- ✓ **Composition of chloride go down as we go higher in the quality of the wine**
- ✓ **Sulphates level goes higher with the quality of wine**
- ✓ **Alcohol level also goes higher as the quality of wine increases**
- ✓ **The volatile acidity decreases as we go higher the quality**

III. Features Pre-Processing :

```
In [619]: wine['quality'].value_counts()
Out[619]: 5    681
          6    638
          7    199
          4     53
          8     18
          3     10
          Name: quality, dtype: int64
```

Assigning good and bad to quality of wine

bad: quality < 6

good: quality > 6

```
In [620]: bins = (2, 6.0, 8)
          group_names = ['bad', 'good']
          wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
```

Fig. 13 Making binary classification for the response variable. Dividing wine as good and bad by giving the limit for the quality

```
wine.drop('pH', axis=1, inplace = True)
wine.drop('density', axis=1, inplace = True)
wine.drop('residual sugar', axis=1, inplace = True)
```

Fig. 14 Dropping of columns

```
In [623]: wine.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   chlorides             1599 non-null   float64
 4   free sulfur dioxide   1599 non-null   float64
 5   total sulfur dioxide  1599 non-null   float64
 6   sulphates             1599 non-null   float64
 7   alcohol               1599 non-null   float64
 8   quality               1599 non-null   category
dtypes: category(1), float64(8)
memory usage: 101.7 KB
```

Fig. 15 Quality converted to categorical variable.

```
In [624]: from sklearn.preprocessing import StandardScaler, LabelEncoder
          label_quality = LabelEncoder()

In [625]: wine['quality'] = label_quality.fit_transform(wine['quality'])

In [626]: wine['quality'].value_counts()

Out[626]: 0    1382
          1     217
          Name: quality, dtype: int64
```

Fig. 16 Assigning 0's to 'bad' and 1's to 'good'

IV. Format of the Dataset : CSV

b. Identification of Learning Model (Supervised Learning)

I. Algorithm used: Support Vector Machine & Random Forest Classifier

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

II. Methodology used :

For feature pre-processing, here I had used label encoding, converted the 'quality' to categorical variable and assigned it the label 'good' & 'bad', and again converted 'good' & 'bad' to 0's and 1's and performed standard scaling on both the train and test datasets

III. Model building, Training & Testing :

```
In [629]: X = wine.drop('quality', axis = 1)
          y = wine['quality']

          rain and Test splitting of data

In [630]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 30)

In [631]: y_test

Out[631]: 1147    1
          659     0
          871     0
          1333    0
          1411    0
                 ..
          72      0
          754     0
```

Fig. 17 Separating the dataset as response variable and feature variables and running train_test_split

```
In [633]: sc = StandardScaler()

In [634]: X_train = sc.fit_transform(X_train)
          X_test = sc.fit_transform(X_test)

In [635]: print ('Train set:', X_train.shape,  y_train.shape)
          print ('Test set:', X_test.shape,  y_test.shape)

          Train set: (1119, 8) (1119,)
          Test set: (480, 8) (480,)
```

Fig. 18 Applying Standard scaling to get optimized result

IV. Model Accuracy, Prediction & Precession :

```
In [636]: #construct a dict to store accuracies of different each model
          svm_acc = {}

In [637]: svc = SVC()
          svc.fit(X_train, y_train)
          pred_svc = svc.predict(X_test)

In [638]: print(classification_report(y_test, pred_svc))

                        precision    recall  f1-score   support

                     0       0.89      0.99      0.94       416
                     1       0.79      0.23      0.36        64

              accuracy                           0.89       480
             macro avg       0.84      0.61      0.65       480
          weighted avg       0.88      0.89      0.86       480
```

Fig. 19 Running SVM with default hyperparameter

```
In [640]: from sklearn import metrics
          y_pred=svc.predict(X_test)
          print('Accuracy Score:')
          print(metrics.accuracy_score(y_test,y_pred)*100,"%")

          Accuracy Score:
          88.95833333333333 %
```

Fig. 20 Accuracy of the model

```
In [641]: #making confusing matrix
          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(y_test,pred_svc)
          sn.heatmap(cm,annot=True,fmt='1.0f')

Out[641]: <AxesSubplot:>
```

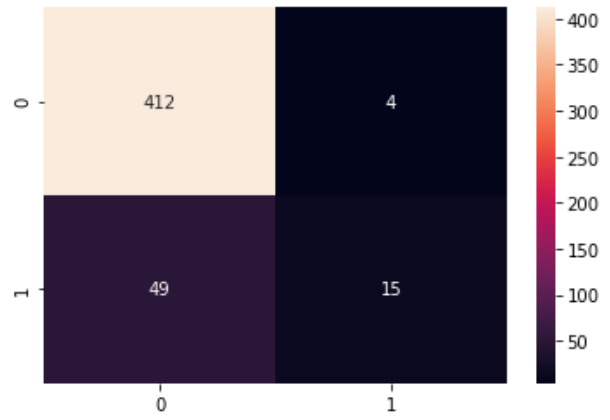


Fig. 21 Accuracy confusion matrix

Out[651]:

	kernel	acc_score
0	svm with Linear kernel	(86.66666666666667, %)
1	svm with RBF kernel	(88.95833333333333, %)
2	svm with Polynomial kernel	(88.125, %)
3	svm with Sigmoid kernel	(88.125, %)

Fig. 22 Accuracy score for all the kernel

```
In [ ]: param = {
            'C': [0.1,0.8,0.9,1,1.1,1.2,1.3,1.4],
            'kernel':['linear', 'rbf'],
            'gamma' :[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]
        }
        grid_svc = GridSearchCV(svc, param_grid=param, scoring='accuracy', cv=10)

In [ ]: grid_svc.fit(X_train, y_train)

In [ ]: #Best parameters for our svc model
        grid_svc.best_params_
```

Fig. 23 Improving the accuracy

```
In [652]: #Let's run our SVC again with the best parameters.
          svc2 = SVC(C = 1.2, gamma =  1.2, kernel= 'rbf')
          svc2.fit(X_train, y_train)
          pred_svc2 = svc2.predict(X_test)
          print(classification_report(y_test, pred_svc2))
          print(metrics.accuracy_score(y_test,pred_svc2)*100,"%")

                        precision    recall  f1-score   support

                     0       0.91      0.98      0.94       416
                     1       0.71      0.34      0.46        64

              accuracy                           0.89       480
             macro avg       0.81      0.66      0.70       480
          weighted avg       0.88      0.89      0.88       480

          89.375 %
```

Fig. 24 Improving the accuracy: 85 to 89 %

c. Key Learning Outcomes :

The red wine data contains various columns that are hard to understand at the first go that which feature are affecting the quality of wine. It took time to completely understand the dataset and then applying supervised algorithm (Support Vector Machine). In the above example the key learning are converting the quality of wine to categorical variable and applying all the kernel hyper parameters and analysing the accuracy results of all four kernels.

Unsupervised Learning

a. Identification of the Dataset :

I. Type of the Dataset : Multivariate, Structured dataset

The data is hosted on the UCI Machine Learning repository. There are multiple product categories – Fresh, Milk, Grocery, etc. The values represent the number of units purchased by each client for each product.

II. Data Quality and Analysis :

```
'Channel'
'Region'
'Fresh'
'Milk'
'Grocery'
'Frozen',
'Detergents_Paper'
'Delicassen'
```

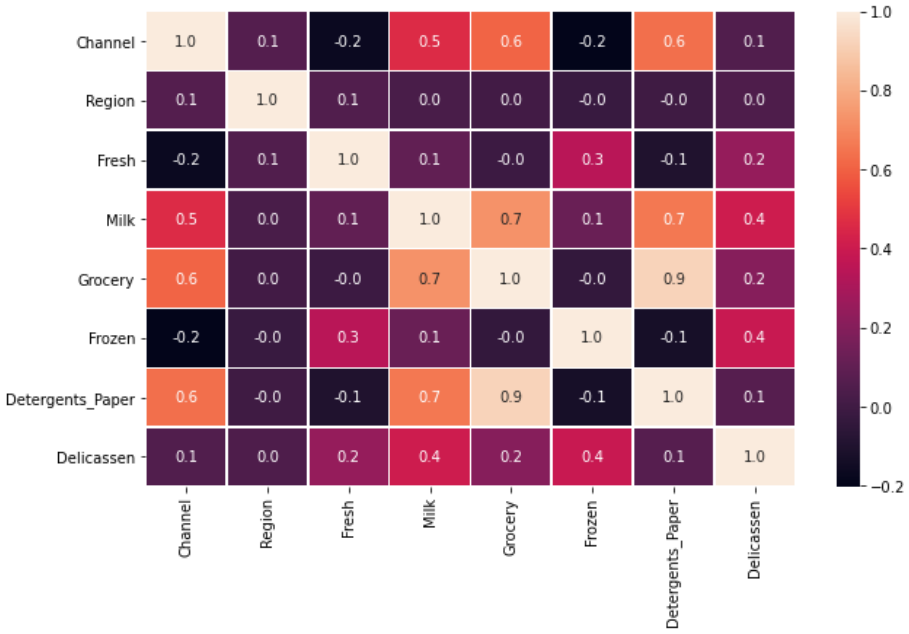
Out[13]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185
...	...	...	...	...	...	...	...	...
435	1	3	29703	12051	16027	13135	182	2204
436	1	3	39228	1431	764	4510	93	2346
437	2	3	14531	15488	30243	437	14841	1867
438	1	3	10290	1981	2232	1038	168	2125
439	1	3	2787	1698	2510	65	477	52

440 rows × 8 columns

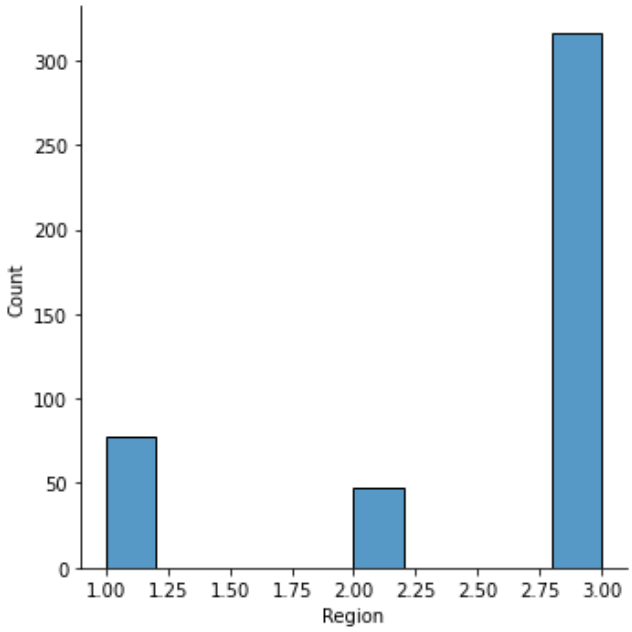
```
In [30]: data.columns
Out[30]: Index(['Channel', 'Region', 'Fresh', 'Milk', 'Grocery', 'Frozen',
                'Detergents_Paper', 'Delicassen'],
               dtype='object')
```

```
In [31]: f,ax = plt.subplots(figsize=(10, 6))
         sn.heatmap(data.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
Out[31]: <AxesSubplot:>
```

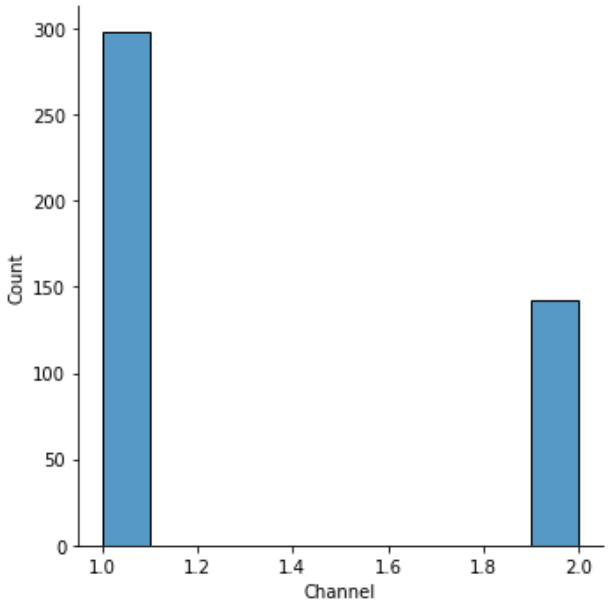



```
In [20]: data.isnull().sum()
Out[20]: Channel            0
         Region             0
         Fresh              0
         Milk               0
         Grocery            0
         Frozen             0
         Detergents_Paper   0
         Delicassen         0
         dtype: int64
```

```
In [23]: sn.displot(data['Region'])
Out[23]: <seaborn.axisgrid.FacetGrid at 0x222a26063a0>
```




```
In [22]: sn.displot(data['Channel'])
Out[22]: <seaborn.axisgrid.FacetGrid at 0x222a7932fa0>
```



III. Features Pre-Processing :

Normalizing the data so that the scale of each variable is the same. if the scale of the variables is not the same, the model might become biased towards the variables with a higher magnitude like Fresh or Milk

In [24]:
```
from sklearn.preprocessing import normalize
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
data_scaled.head()
```

Out[24]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286
2	0.000125	0.000187	0.396552	0.549792	0.479632	0.150119	0.219467	0.489619
3	0.000065	0.000194	0.856837	0.077254	0.272650	0.413659	0.032749	0.115494
4	0.000079	0.000119	0.895416	0.214203	0.284997	0.155010	0.070358	0.205294

IV. Format of the Dataset : CSV

b. Identification of Learning Model (Unsupervised Learning)

i. Algorithm used : Hierarchical clustering

Hierarchical clustering algorithms group similar objects into groups called **clusters**. There are two types of hierarchical clustering algorithms:

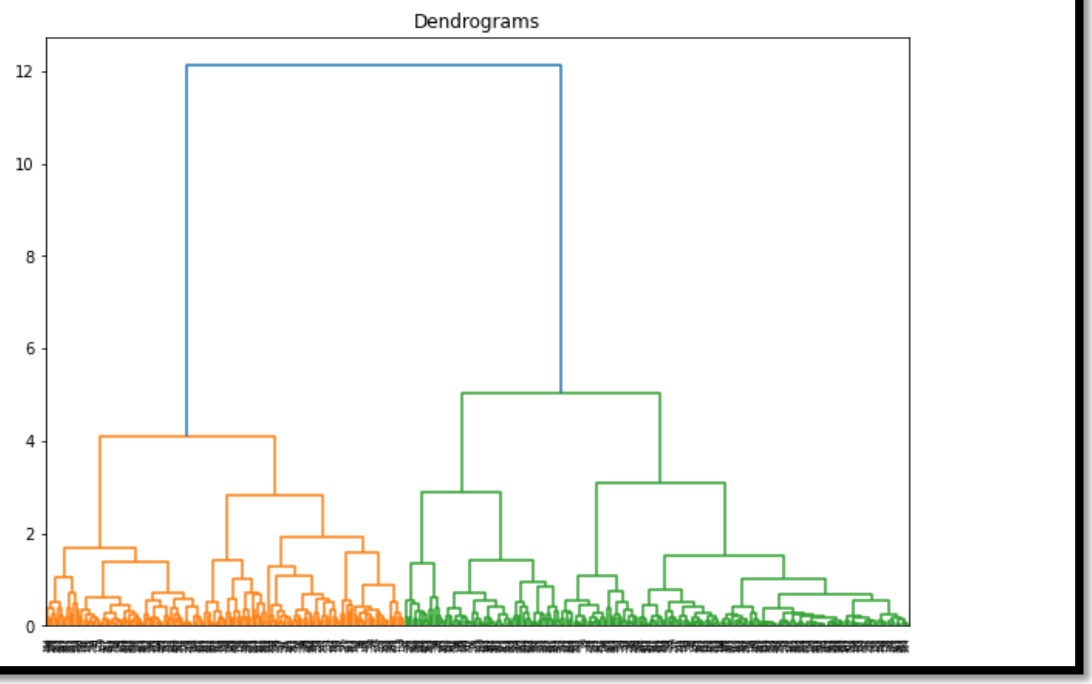
- Agglomerative — Bottom up approach. Start with many small clusters and merge them together to create bigger clusters.
- Divisive — Top down approach. Start with a single cluster than break it up into smaller clusters.

ii. Model building, Training & Testing :

we can see that the scale of all the variables is almost similar. Now, we are good to go.

Let's first draw the dendrogram to help us decide the number of clusters.

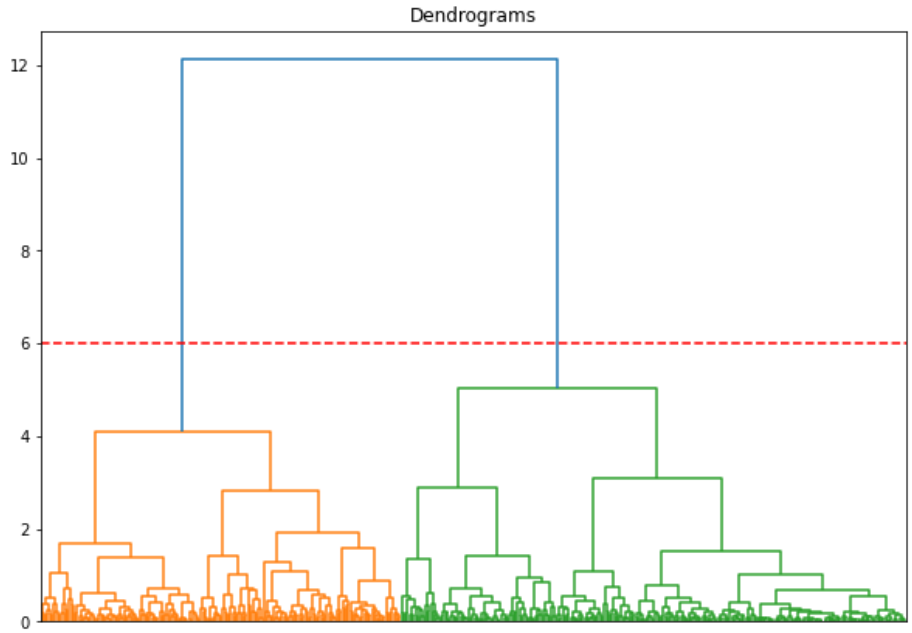
```
In [25]: import scipy.cluster.hierarchy as shc
         plt.figure(figsize=(10, 7))
         plt.title("Dendrograms")
         dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
```



The x-axis contains the samples and y-axis represents the distance between these samples. The vertical line with maximum distance is the blue line and hence we can decide a threshold of 6 and cut the dendrogram.

```
In [33]: plt.figure(figsize=(10, 7))
         plt.title("Dendrograms")
         dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
         plt.axhline(y=6, color='r', linestyle='--')
```

```
Out[33]: <matplotlib.lines.Line2D at 0x222a9f9b850>
```



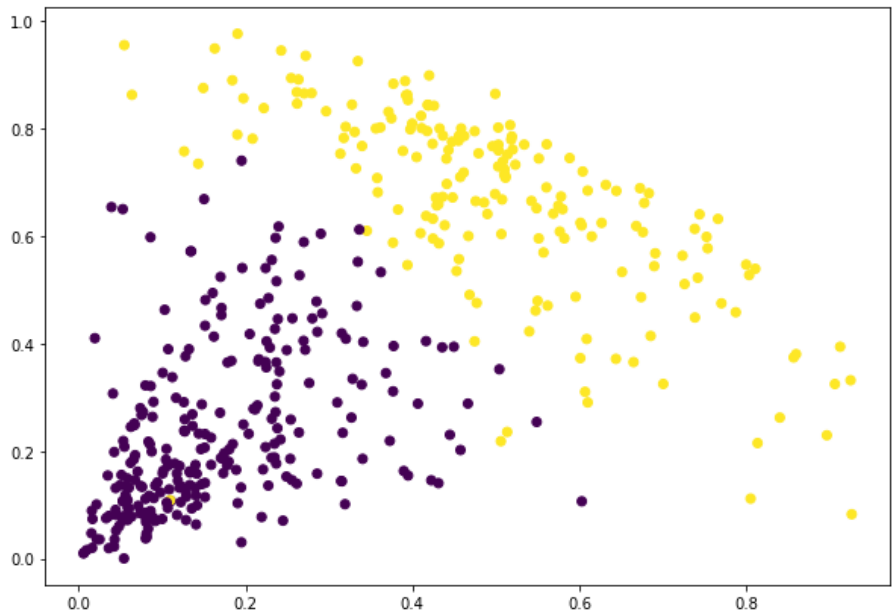
We have two clusters as this line cuts the dendrogram at two points. Let's now apply hierarchical clustering for 2 clusters

```
n [26]: from sklearn.cluster import AgglomerativeClustering
        cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
        cluster.fit_predict(data_scaled)
```

```
ut[26]: array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
               0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
               1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
               1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
               0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
               0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
               0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
               0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
               0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
               0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
               0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
               0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
               1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
               0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
```

The values of 0s and 1s in the output since we defined 2 clusters. 0 represents the points that belong to the first cluster and 1 represents points in the second cluster.

```
In [27]: plt.figure(figsize=(10, 7))
         plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
Out[27]: <matplotlib.collections.PathCollection at 0x222a89db700>
```



Visualization of clusters

c. Key Learning Outcomes :

Hierarchical clustering is a super useful way of segmenting observations. The advantage of not having to pre-define the number of clusters gives it quite an edge over k-Means.

Submitted By:

Aditya Sharma

AU18B1009