Module 8 exercises: Unsupervised Learning

The data

- We will be using the well explored Wine Data from SciKit-Learn. There are 13 features and a 'real_class' column.
- Please remember that in the real world you will not have a real class column, as the goal of clustering is to find out how many distinct classes there are and then analyse the properties of each, in order to derive some value.

Exercise 1

1. Import the following packages:
    a. pandas
    b. matplotlib
    c. numpy
    d. seaborn
2. Run the following code to obtain the wine dataset from `sklearn`:

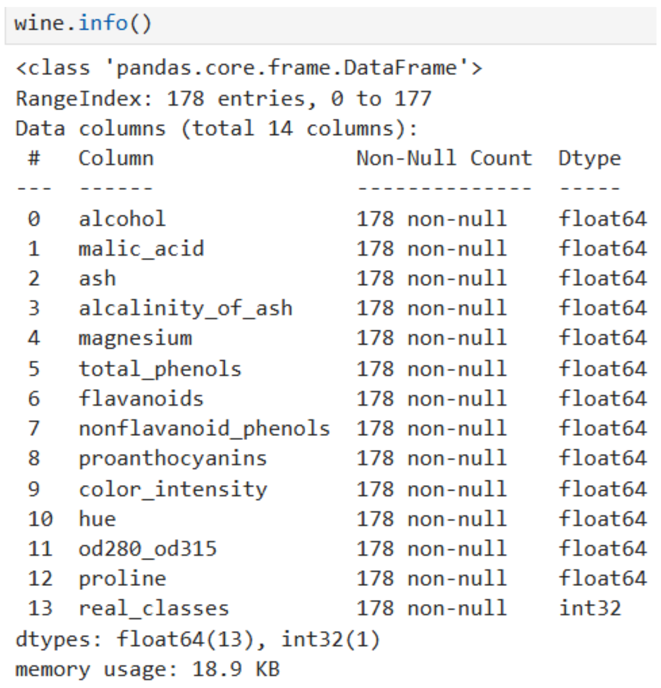
```
from sklearn.datasets import load_wine

## Import the Wine dataset from SciKit-Learn
wine_bunch = load_wine()

# Allocating this data to a Data Frame
wine = pd.DataFrame(wine_bunch.data)

# Use the feature_names attribute to give sensible column headings
# Assigning wine.columns to col names for use later
col_names = list(wine_bunch.feature_names)

# Once colum has a lenghty name, we rename this for ease of display
# od280/od315_of_diluted_wines   -> od280_od315
col_names[11] = 'od280_od315'
wine.columns = col_names

# I am adding a 1 to the 'real classes' columns to avoid having to talk
about 'cluster 0'
# We will start the count from 1
wine['real_classes'] = wine_bunch.target + 1
```

```
wine.head()
```

Exercise 2 – EDA

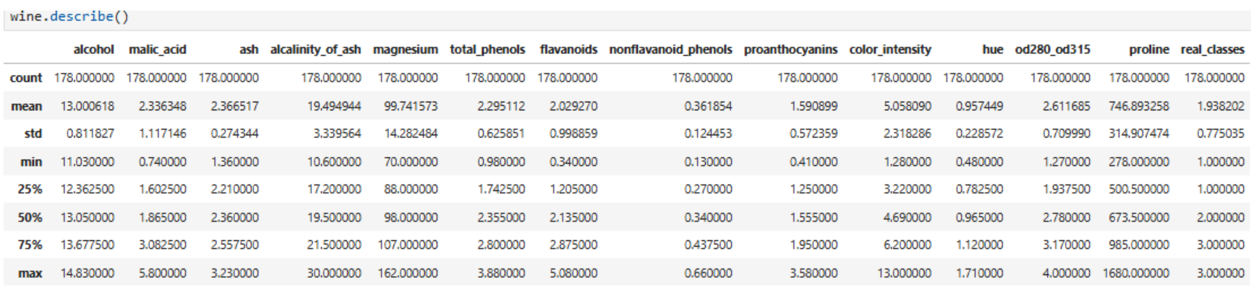
Familiarise yourself with the dataset. Seek to understand:

- the types of data.
- distributions of features.
- obvious patterns.
- errors, nulls, outliers.



```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   alcohol               178 non-null    float64
 1   malic_acid            178 non-null    float64
 2   ash                   178 non-null    float64
 3   alcalinity_of_ash     178 non-null    float64
 4   magnesium             178 non-null    float64
 5   total_phenols         178 non-null    float64
 6   flavanoids            178 non-null    float64
 7   nonflavanoid_phenols  178 non-null    float64
 8   proanthocyanins       178 non-null    float64
 9   color_intensity       178 non-null    float64
 10  hue                   178 non-null    float64
 11  od280_od315           178 non-null    float64
 12  proline               178 non-null    float64
 13  real_classes          178 non-null    int32  
dtypes: float64(13), int32(1)
memory usage: 18.9 KB
```



```
wine.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280_od315	proline	real_classes
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	1.938202
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	0.775035
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	1.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	1.000000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	2.000000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	3.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	3.000000

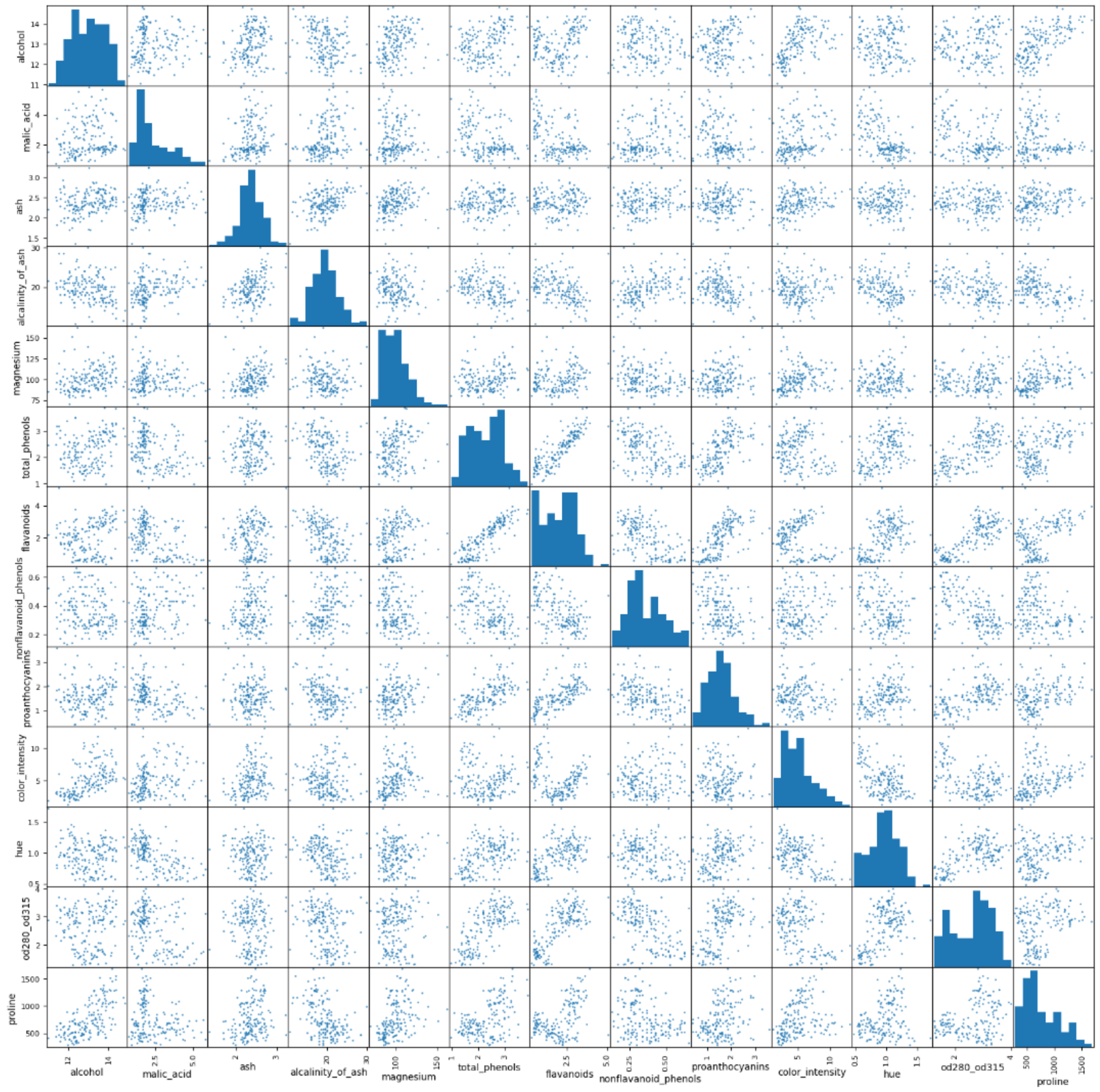
```
wine['real_classes'].unique()
```

```
array([1, 2, 3])
```

```
wine['real_classes'].value_counts()
```

```
real_classes
2    71
1    59
3    48
Name: count, dtype: int64
```

```
# Plotting all features against eachother on acatter plots and
# Visualising their distributions

g = pd.plotting.scatter_matrix(wine[col_names],     # The col_names list excludes the real_classes column
                               figsize=(20,20),     # Need a large plot as we have many variables,
                                                    # experiment with this
                               marker = '.',        # Specifying marker, this is the default anyway
                               s = 15,              # Size of each marker
                               alpha = 0.8)         # Opacity of each marker
```

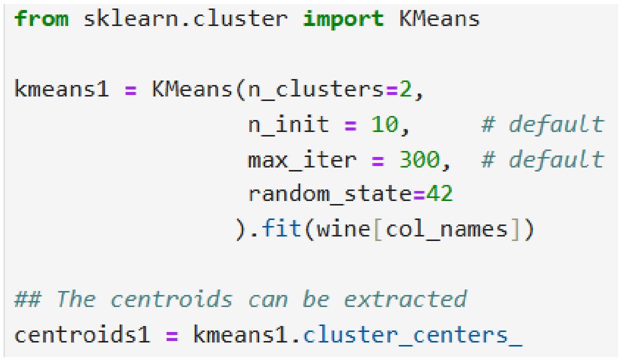


Exercise 3 – Building your first model

After exploring the dataset, we can apply our clustering algorithm. We start by using the raw data as-is, though down the line we may make some changes.

1. Import the `KMeans` model from scikit-learn's `cluster` module.

2. Create a `KMeans` model, setting `n_clusters=2`, and calling it kmeans1. Then use the `fit` method to train the model on `wine[col_names]`.

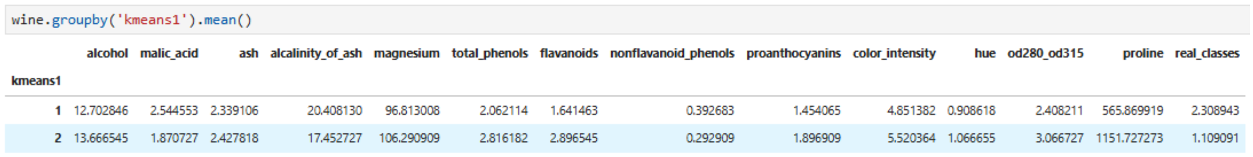


```
from sklearn.cluster import KMeans

kmeans1 = KMeans(n_clusters=2,
                 n_init = 10,      # default
                 max_iter = 300,   # default
                 random_state=42
                ).fit(wine[col_names])

## The centroids can be extracted
centroids1 = kmeans1.cluster_centers_
```

3. Store the centroids and add a column giving each point it's label. The code below can be used:

```
## The centroids can be extracted
centroids1 = kmeans1.cluster_centers_

## This is the list of allocated classes
labels1 = kmeans1.labels_

## Appending these labels to our original dataframe
# I added 1 again to avoid talking about class 0
wine['kmeans1'] = labels1 + 1
```

4. Compute the mean value of each feature per group label given by `KMeans`. What are the key differences between the groups?



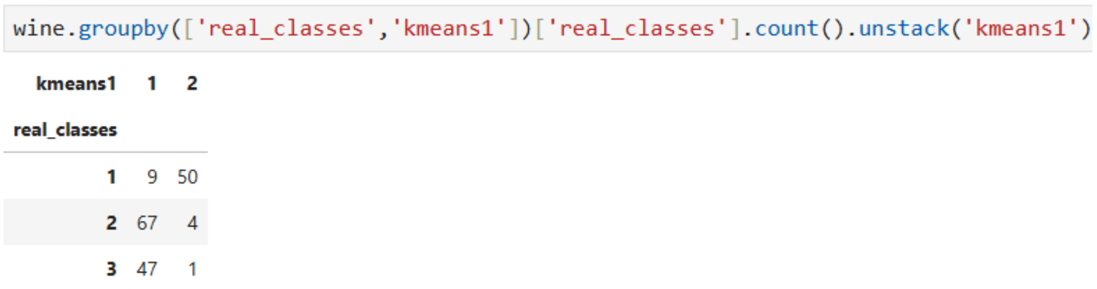
```
wine.groupby('kmeans1').mean()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280_od315	proline	real_classes
kmeans1														
1	12.702846	2.544553	2.339106	20.408130	96.813008	2.062114	1.641463	0.392683	1.454065	4.851382	0.908618	2.408211	565.869919	2.308943
2	13.666545	1.870727	2.427818	17.452727	106.290909	2.816182	2.896545	0.292909	1.896909	5.520364	1.066655	3.066727	1151.727273	1.109091

5. Use the below code to plot the clustered data:

```
import matplotlib as mpl
mpl.rcParams['axes.prop_cycle'] = mpl.cycler(color=["b", "r", "g"])

plot = pd.plotting.scatter_matrix(wine[col_names],
                                  figsize=(20,20),
                                  marker = '.',
                                  s = 15,
                                  alpha = 0.8,
                                  c = wine['kmeans1'])
```

6. If you have time, try to display the count of actual label vs. predicted label for each group.



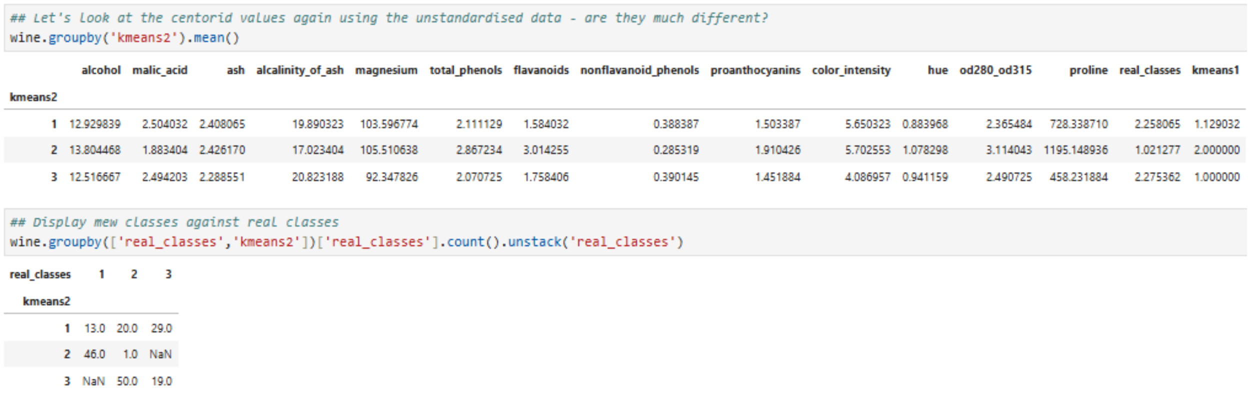
```python
wine.groupby(['real_classes','kmeans1'])['real_classes'].count().unstack('kmeans1')
```

kmeans1 real_classes	1	2
1	9	50
2	67	4
3	47	1

Exercise 4 – Changing clusters and features:

1. Repeat Exercise 3 but using three clusters. Compare this to the original classes as we did above

```python
# Importing k-means from SciKIt-Learn
from sklearn.cluster import KMeans

### Fitting k-means clusters is simple - we only need one line of code

kmeans2 = KMeans(n_clusters=3,
                 n_init = 10,     # default
                 max_iter = 300,  # default
                 random_state=42
                ).fit(wine[col_names])

## The centroids can be extracted
centroids2 = kmeans2.cluster_centers_

## This is the list of allocated classes
labels2 = kmeans2.labels_

## Appending these labels to our original dataframe
# I added 1 again to avoid talking about class 0
wine['kmeans2'] = labels2 + 1
```



```python
## Let's look at the centorid values again using the unstandardised data - are they much different?
wine.groupby('kmeans2').mean()
```

kmeans2	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280_od315	proline	real_classes	kmeans1
1	12.929839	2.504032	2.408065	19.890323	103.596774	2.111129	1.584032	0.388387	1.503387	5.650323	0.883968	2.365484	728.338710	2.258065	1.129032
2	13.804468	1.883404	2.426170	17.023404	105.510638	2.867234	3.014255	0.285319	1.910426	5.702553	1.078298	3.114043	1195.148936	1.021277	2.000000
3	12.516667	2.494203	2.288551	20.823188	92.347826	2.070725	1.758406	0.390145	1.451884	4.086957	0.941159	2.490725	458.231884	2.275362	1.000000

```python
## Display mew classes against real classes
wine.groupby(['real_classes','kmeans2'])['real_classes'].count().unstack('real_classes')
```

real_classes kmeans2	1	2	3
1	13.0	20.0	29.0
2	46.0	1.0	NaN
3	NaN	50.0	19.0

2. Repeat it again but this time using only the below five features:

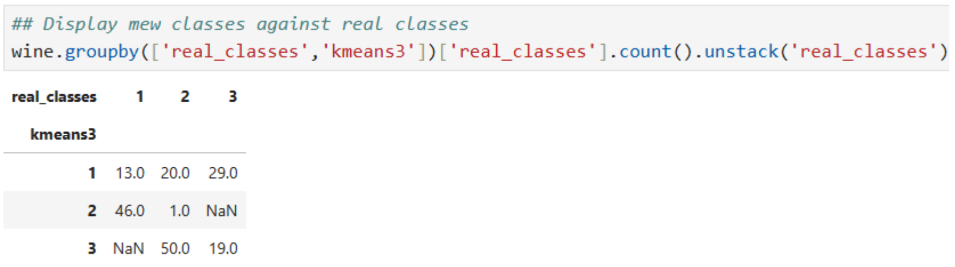
- Alcohol
- Alcalinity_of_ash
- Magnesium
- Color_intensity
- Proline

```
# Create a list of reduced columns
reduced_cols = ['alcohol','alcalinity_of_ash','magnesium','color_intensity','proline']
reduced_cols
```

```
['alcohol', 'alcalinity_of_ash', 'magnesium', 'color_intensity', 'proline']
```

```
# fit the model, 3 clusters, reduced features

#### REMEMBER TO RENAME EVERYTHING SO NOT TO OVERWRITE ####

## Fitting k-means clusters is simple - we only need one line of code
kmeans3 = KMeans(n_clusters=3,
                 random_state=3).fit(wine[reduced_cols])

## The resulting kmeans1 object has centroid values...
centroids3 = kmeans3.cluster_centers_

## And it has an array of labels, one for each datapoint.
labels3 = kmeans3.labels_

## We can use these labels to create a new column in our original dataset:
wine['kmeans3'] = labels3 +1
```

3. How does this model with five features compare to the model with all? Why do you think this is the case? What would you do to resolve the issue?



```
## Display mew classes against real classes
wine.groupby(['real_classes','kmeans3'])['real_classes'].count().unstack('real_classes')
```

real_classes	1	2	3
kmeans3			
1	13.0	20.0	29.0
2	46.0	1.0	NaN
3	NaN	50.0	19.0

4. Re-run once more with only one feature (you should see which) * Examine the output of this

```python
# fit the model, 3 clusters, reduced features

#### REMEMBER TO RENAME EVERYTHING SO NOT TO OVERWRITE ####

## Fitting k-means clusters is simple - we only need one line of code
kmeans4 = KMeans(n_clusters=3,
                 random_state=3).fit(wine['proline'].values.reshape(-1, 1))

## The resulting kmeans1 object has centroid values...
centroids4 = kmeans4.cluster_centers_

## And it has an array of labels, one for each datapoint.
labels4 = kmeans4.labels_

## We can use these labels to create a new column in our original dataset:
wine['kmeans4'] = labels4 +1
```

```python
## Display mew classes against real classes
wine.groupby(['real_classes','kmeans4'])['real_classes'].count().unstack('real_classes')
```

real_classes	1	2	3
kmeans4			
1	13.0	20.0	29.0
2	46.0	1.0	NaN
3	NaN	50.0	19.0

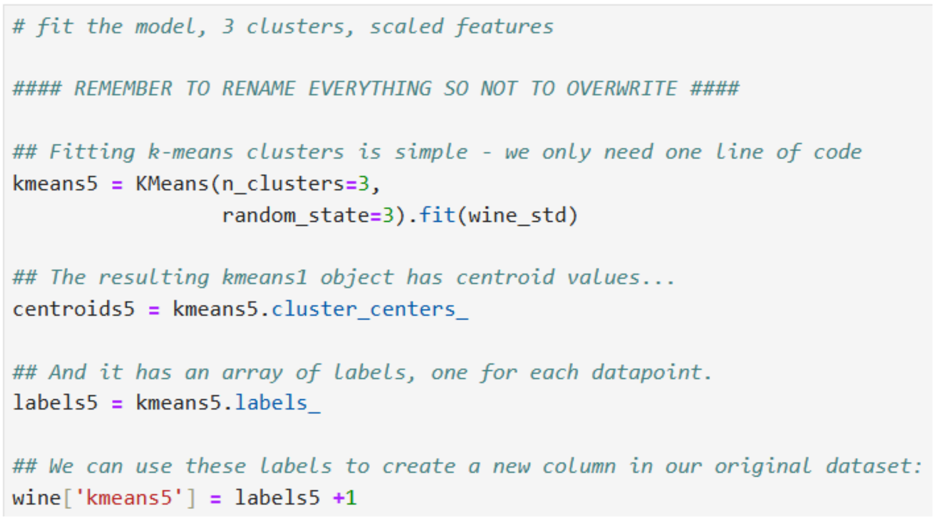
Note: For each model create new columns in the DataFrame to store the results, e.g., kmeans2 and kmeans3.

Exercise 5 – Scaling

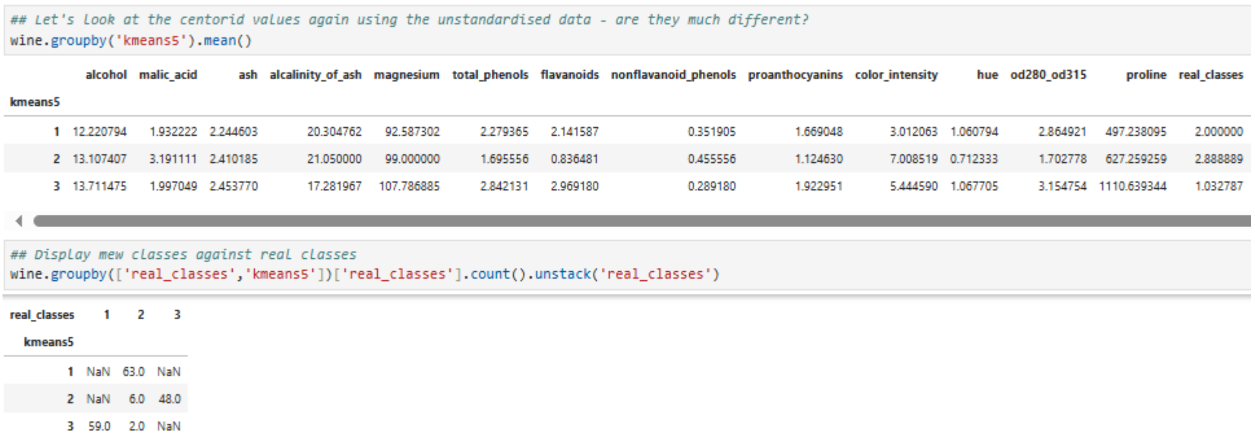
1. Our model was being affected by the large scale of some of our features. Apply the `MinMaxScaler` which can be imported from sklearn's preprocessing module to each feature.

```python
from sklearn import preprocessing
# defining the columns to scale
columns_to_scale = col_names

# calling the MinMaxScalar with default settings
min_max_scaler = preprocessing.MinMaxScaler()

# FItting the scalar to the two named columns, creating a new DataFrame from this array
# Assigning the column names to the new dataframe
wine_std = pd.DataFrame(min_max_scaler.fit_transform(wine[columns_to_scale]),
                                      columns = min_max_scaler.get_feature_names_out() )

# Viewing the new DataFrame
wine_std
```

2. Repeat Exercise 3, fitting `KMeans` to the scaled dataset.



```
# fit the model, 3 clusters, scaled features

#### REMEMBER TO RENAME EVERYTHING SO NOT TO OVERWRITE ####

## Fitting k-means clusters is simple - we only need one line of code
kmeans5 = KMeans(n_clusters=3,
                 random_state=3).fit(wine_std)

## The resulting kmeans1 object has centroid values...
centroids5 = kmeans5.cluster_centers_

## And it has an array of labels, one for each datapoint.
labels5 = kmeans5.labels_

## We can use these labels to create a new column in our original dataset:
wine['kmeans5'] = labels5 +1
```



```
## Let's look at the centorid values again using the unstandardised data - are they much different?
wine.groupby('kmeans5').mean()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280_od315	proline	real_classes
kmeans5														
1	12.220794	1.932222	2.244603	20.304762	92.587302	2.279365	2.141587	0.351905	1.669048	3.012063	1.060794	2.864921	497.238095	2.000000
2	13.107407	3.191111	2.410185	21.050000	99.000000	1.695556	0.836481	0.455556	1.124630	7.008519	0.712333	1.702778	627.259259	2.888889
3	13.711475	1.997049	2.453770	17.281967	107.786885	2.842131	2.969180	0.289180	1.922951	5.444590	1.067705	3.154754	1110.639344	1.032787

```
## Display mew classes against real classes
wine.groupby(['real_classes','kmeans5'])['real_classes'].count().unstack('real_classes')
```

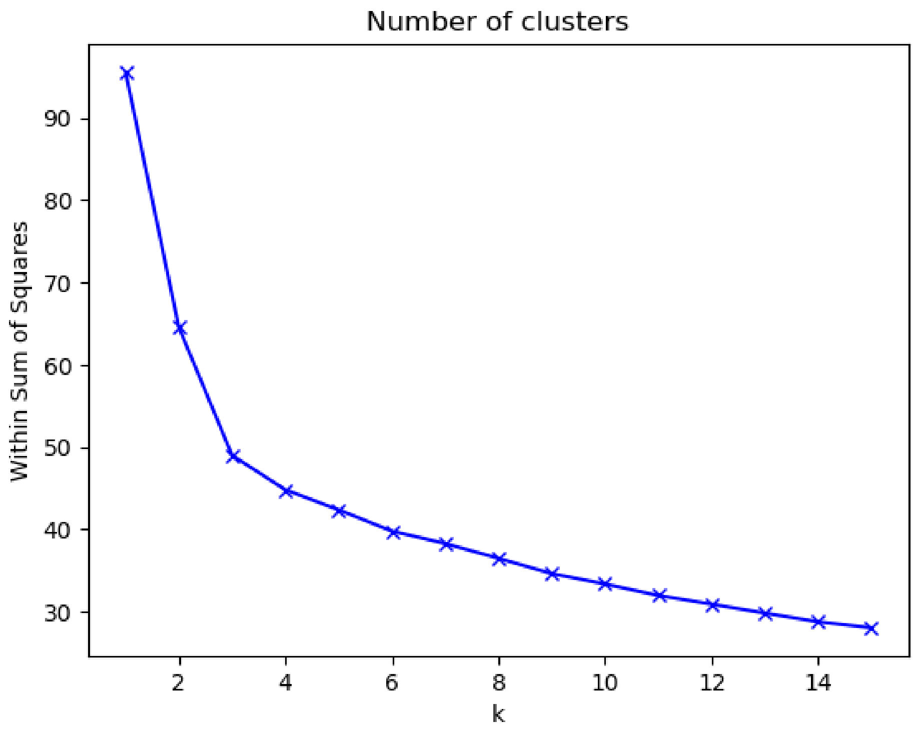
real_classes	1	2	3
kmeans5			
1	NaN	63.0	NaN
2	NaN	6.0	48.0
3	59.0	2.0	NaN

3. What difference do you see?

Exercise 6 – Choosing the optimal number of clusters

1. Run the below code and examine the output. What does it show you?

```
## We use a for loop to try different numbers of clusters
## For each k, we calculate the 'inertia' and record these values
WCSS = {}
K = range(1,16)
for k in K:
    km = KMeans(n_clusters=k, random_state=42).fit(wine_std)
    WCSS[f'{k}'] = km.inertia_

## Now we can plot to help us identify the 'optimal' value for k
plt.plot(K, WCSS.values(), 'bx-')
plt.xlabel('k')
plt.ylabel('Within Sum of Squares')
plt.title('Number of clusters')
plt.show()
```



Exercise 7 – Extension

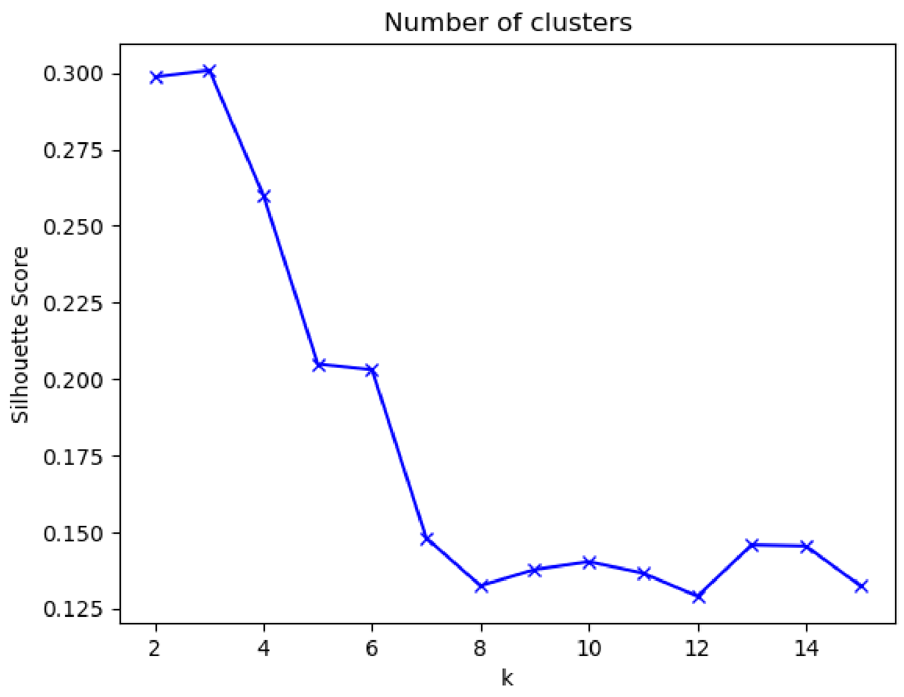
1. Examine the output of the below code which computes the silhouette coefficient and visualises it. Explore the metric and interpret the outputs.

```
from sklearn.metrics import silhouette_score, silhouette_samples

silhouettes = silhouette_samples(X=wine[col_names],
                                 labels=wine['kmeans5'])
wine["silhouette"] = silhouettes

g = sns.FacetGrid(wine, col="kmeans5")
g.map(sns.histplot, "silhouette")
```

In a new cell

```
## We use a for loop to try different numbers of clusters
silhouettes = {}
K = range(2,16)
for k in K:
    km = KMeans(n_clusters=k, random_state=42).fit(wine_std)
    silhouettes[f'{k}'] = silhouette_score(X=wine_std,
                                           labels=km.labels_)
```

```python
## Now we can plot to help us identify the 'optimal' value for k
plt.plot(K, silhouettes.values(), 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette Score')
plt.title('Number of clusters')
plt.show()
```



Exercise 8 – Other clustering approaches

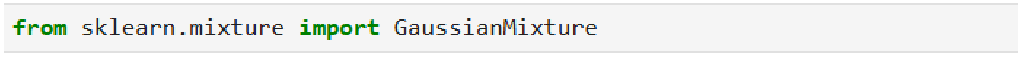
If you have time, explore the following other clustering approaches:

- DBScan

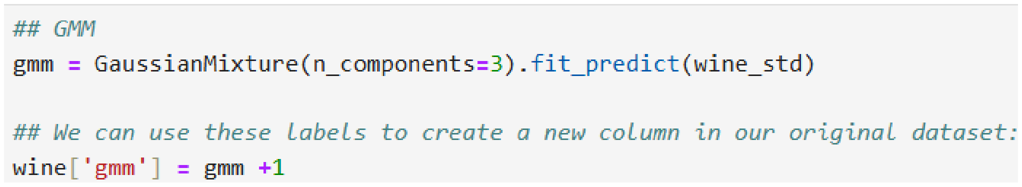
```python
from sklearn.cluster import DBSCAN
```

```python
## DBScan
dbscan = DBSCAN(eps=0.6,
                min_samples=5).fit(wine_std)

## We can use these labels to create a new column in our original dataset:
wine['dbscan'] = dbscan.labels_ +1
```

- Gaussian Mixture Modelling

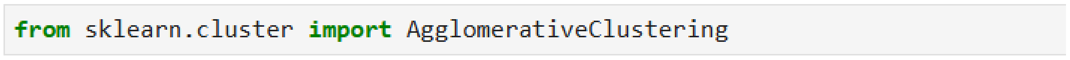


```python
from sklearn.mixture import GaussianMixture
```

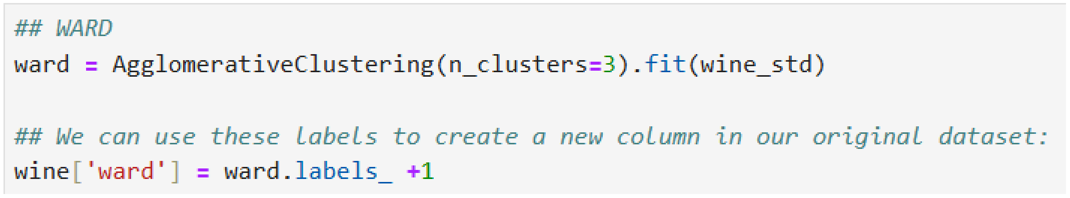


```python
## GMM
gmm = GaussianMixture(n_components=3).fit_predict(wine_std)

## We can use these labels to create a new column in our original dataset:
wine['gmm'] = gmm +1
```

- WARD



```python
from sklearn.cluster import AgglomerativeClustering
```



```python
## WARD
ward = AgglomerativeClustering(n_clusters=3).fit(wine_std)

## We can use these labels to create a new column in our original dataset:
wine['ward'] = ward.labels_ +1
```