Unsupervised Learning

K-means clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

```
# read the dataset
retail_df = pd.read_csv("Online_Retail.csv", sep=",", encoding="ISO-8859-1", header=0)
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0
1	536365	71053	WHITE METAL	6	01-12-2010 08:26	3.39	17850.0

Clean the data

```
# missing values
round(100*(retail_df.isnull().sum())/len(retail_df), 2)
```

```
InvoiceNo       0.00
StockCode       0.00
Description     0.27
Quantity        0.00
InvoiceDate     0.00
UnitPrice       0.00
CustomerID     24.93
Country         0.00
dtype: float64
```

```
# drop all rows having missing values
retail_df = retail_df.dropna()
retail_df.shape
```

```
(406829, 8)
```

```
# new column: amount
retail_df['amount'] = retail_df['Quantity']*retail_df['UnitPrice']
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0
1	536365	71053	WHITE METAL	6	01-12-2010 08:26	3.39	17850.0

Prepare the data for modelling

```
# monetary
grouped_df = retail_df.groupby('CustomerID')['amount'].sum()
```

```
grouped_df = grouped_df.reset_index()
grouped_df.head()
```

	CustomerID	amount
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40

```
# frequency
frequency = retail_df.groupby('CustomerID')['InvoiceNo'].count()
frequency = frequency.reset_index()
frequency.columns = ['CustomerID', 'frequency']
frequency.head()
```

	CustomerID	frequency
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17

```
# merge the two dfs
grouped_df = pd.merge(grouped_df, frequency, on='CustomerID', how='inner')
grouped_df.head()
```

	CustomerID	amount	frequency
0	12346.0	0.00	2
1	12347.0	4310.00	182
2	12348.0	1797.24	31
3	12349.0	1757.55	73
4	12350.0	334.40	17

```
# recency
# convert to datetime
retail_df['InvoiceDate'] = pd.to_datetime(retail_df['InvoiceDate'],
                                          format='%d-%m-%Y %H:%M')
```

```
# compute the max date
max_date = max(retail_df['InvoiceDate'])
max_date
```

```
Timestamp('2011-12-09 12:50:00')
```

```
# merge
grouped_df = pd.merge(grouped_df, last_purchase, on='CustomerID', how='inner')
grouped_df.columns = ['CustomerID', 'amount', 'frequency', 'recency']
grouped_df.head()
```

	CustomerID	amount	frequency	recency
0	12346.0	0.00	2	325 days 02:33:00
1	12347.0	4310.00	182	1 days 20:58:00
2	12348.0	1797.24	31	74 days 23:37:00
3	12349.0	1757.55	73	18 days 02:59:00
4	12350.0	334.40	17	309 days 20:49:00

```
# removing (statistical) outliers
Q1 = grouped_df.amount.quantile(0.05)
Q3 = grouped_df.amount.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.amount >= Q1 - 1.5*IQR) & (grouped_df.amount <= Q3 + 1.5*IQR)]
```

```
# outlier treatment for recency
Q1 = grouped_df.recency.quantile(0.05)
Q3 = grouped_df.recency.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.recency >= Q1 - 1.5*IQR) & (grouped_df.recency <= Q3 + 1.5*IQR)]

# outlier treatment for frequency
Q1 = grouped_df.frequency.quantile(0.05)
Q3 = grouped_df.frequency.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.frequency >= Q1 - 1.5*IQR) & (grouped_df.frequency <= Q3 + 1.5*IQR)]
```

```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['amount', 'frequency', 'recency']
rfm_df_scaled.head()
```

	amount	frequency	recency
0	-0.723738	-0.752888	2.301611
1	1.731617	1.042467	-0.906466
2	0.300128	-0.463636	-0.183658
3	0.277517	-0.044720	-0.738141
4	-0.533235	-0.603275	2.143188

Modelling

```
# k-means with some arbitrary k
kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
kmeans.labels_
```

```
array([3, 1, 0, ..., 3, 0, 0], dtype=int32)
```

```
# help(KMeans)
```

Finding the Optimal Number of Clusters

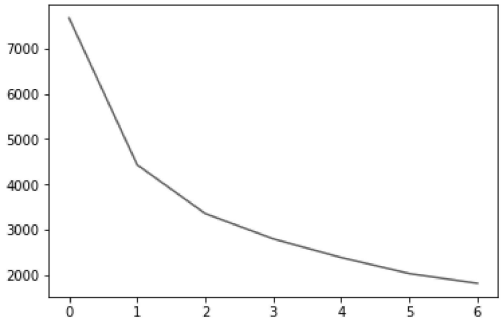
SSD

```
# elbow-curve/SSD
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    ssd.append(kmeans.inertia_)

# plot the SSDs for each n_clusters
# ssd
plt.plot(ssd)
```

```
[<matplotlib.lines.Line2D at 0x1a545bf400>]
```



Silhouette Analysis

```
# silhouette analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)
    cluster_labels = kmeans.labels_
    # silhouette score
    silhouette_avg = silhouette_score(rfm_df_scaled, cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhouette_avg))
```

```
For n_clusters=2, the silhouette score is 0.5415858652525395
For n_clusters=3, the silhouette score is 0.5084896296141937
For n_clusters=4, the silhouette score is 0.4814786837400834
For n_clusters=5, the silhouette score is 0.4658529685822305
For n_clusters=6, the silhouette score is 0.41707960376211345
For n_clusters=7, the silhouette score is 0.4158077420309644
For n_clusters=8, the silhouette score is 0.4059904161107271
```

```
# final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
kmeans.labels_
```

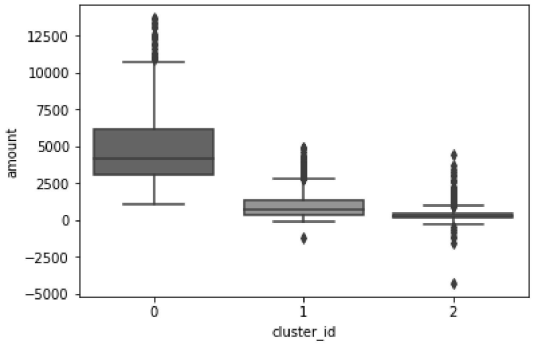
```
array([2, 0, 1, ..., 2, 1, 1], dtype=int32)
```

```
# assign the label
grouped_df['cluster_id'] = kmeans.labels_
grouped_df.head()
```

	CustomerID	amount	frequency	recency	cluster_id
0	12346.0	0.00	2	325	2
1	12347.0	4310.00	182	1	0
2	12348.0	1797.24	31	74	1
3	12349.0	1757.55	73	18	1
4	12350.0	334.40	17	309	2

```
# plot
sns.boxplot(x='cluster_id', y='amount', data=grouped_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a485086d8>
```



Hierarchical Clustering

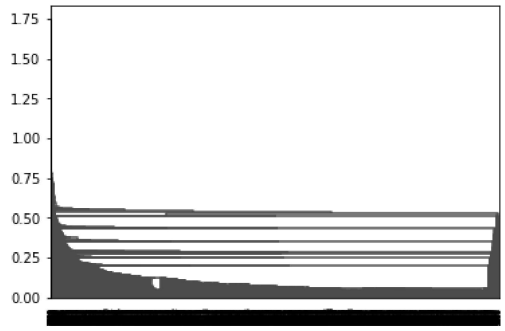
```
rfm_df_scaled.head()
```

	amount	frequency	recency
0	-0.723738	-0.752888	2.301611
1	1.731617	1.042467	-0.906466

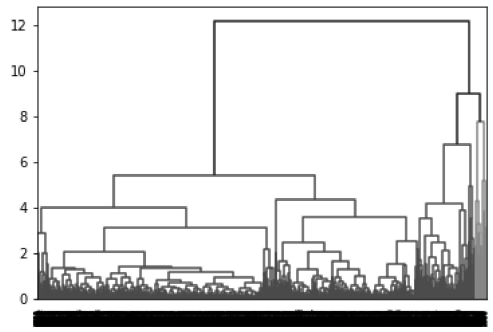
```
grouped_df.head()
```

	CustomerID	amount	frequency	recency	cluster_id
0	12346.0	0.00	2	325	2
1	12347.0	4310.00	182	1	0
2	12348.0	1797.24	31	74	1
3	12349.0	1757.55	73	18	1
4	12350.0	334.40	17	309	2

```
# single linkage
mergings = linkage(rfm_df_scaled, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```




```
# complete linkage
mergings = linkage(rfm_df_scaled, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```




```
# 3 clusters
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

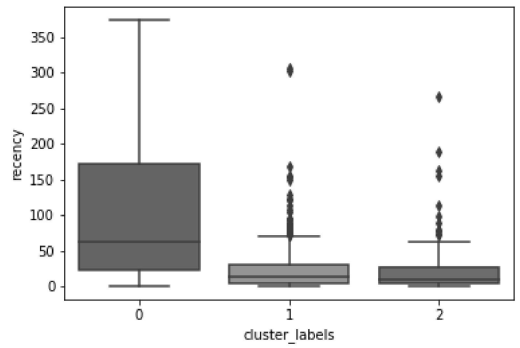
```
array([0, 1, 0, ..., 0, 0, 0])
```

```
# assign cluster labels
grouped_df['cluster_labels'] = cluster_labels
grouped_df.head()
```

	CustomerID	amount	frequency	recency	cluster_id	cluster_labels
0	12346.0	0.00	2	325	2	0
1	12347.0	4310.00	182	1	0	1
2	12348.0	1797.24	31	74	1	0
3	12349.0	1757.55	73	18	1	0
4	12350.0	334.40	17	309	2	0

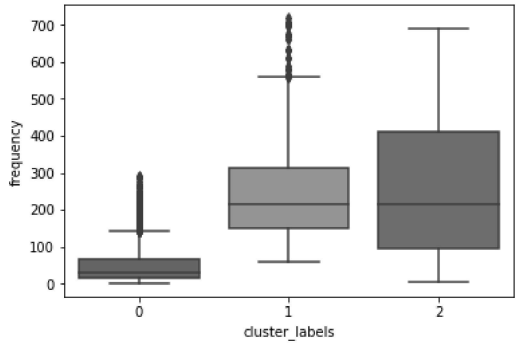
```
# plots
sns.boxplot(x='cluster_labels', y='recency', data=grouped_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5e154438>
```




```
# plots
sns.boxplot(x='cluster_labels', y='frequency', data=grouped_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a35360cc0>
```




```
# plots
sns.boxplot(x='cluster_labels', y='amount', data=grouped_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a48761320>
```