Practical 4

Jumping Rivers

For our clustering example we will explore some data on credit card usage behaviour. This sort of data might be used to develop a customer segmentation in order to assist with a marketing strategy.

The data can be described with the following variables:

- CUST_ID : Identification of Credit Card holder (Categorical)
- BALANCE : Balance amount left in their account to make purchases
- BALANCE_FREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
- PURCHASES : Amount of purchases made from account
- ONEOFF_PURCHASES : Maximum purchase amount done in one-go
- INSTALLMENTS_PURCHASES : Amount of purchase done in installment
- CASH_ADVANCE : Cash in advance given by the user
- PURCHASES_FREQUENCY : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
- ONEOFF_PURCHASES_FREQUENCY : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
- PURCHASES_INSTALLMENTS_FREQUENCY : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
- CASH_ADVANCE_FREQUENCY : How frequently the cash in advance being paid
- CASH_ADVANCE_TRX : Number of Transactions made with "Cash in Advanced"
- PURCHASES_TRX : Numbe of purchase transactions made CREDIT_LIMIT : Limit of Credit Card for user
- PAYMENTS : Amount of Payment done by user
- MINIMUM_PAYMENTS : Minimum amount of payments made by user
- PRC_FULL_PAYMENT : Percent of full payment paid by user
- TENURE : Tenure of credit card service for user

It can be loaded from the package as follows

```
import jrpyml
ccdata = jrpyml.datasets.ccdata.load_data()
```

This data is a real sample so is a little messy, for example we have a number of missing values. The easiest way to deal with this for now is to remove them, however we could in theory deal with this with some form of imputation.

- Remove the missing values from the data
- Fit KMeans clustering to the data across a range of values to produce an elbow plot. How many cluster do you think are apparent here? Don't forget to scale your data appropriately first.
- Add the new cluster labels into the data set as a variable named cluster.
- Now that we have different clusters we might look to try to understand the customer segments, below is a definition of a function which will let you explore individual variables across the clusters.

```
import seaborn as sns

plot_data = ccdata.drop('CUST_ID', axis=1)

def examine(plot_data, column):
  grid = sns.FacetGrid(plot_data, col='cluster')
  grid.map(sns.distplot, column)
  plt.show()

examine(plot_data, 'CREDIT_LIMIT')
```

- It is possible that some outliers make understanding the clusters more difficult. We haven't really discussed outlier detection during the material but a `LocalOutlierFactor` measures deviation of a given sample with respect to its surrounding neighbourhood. We might trim outliers measured this way with the following code.

```
from sklearn.neighbors import LocalOutlierFactor
outlier = LocalOutlierFactor()

labels = outlier.fit_predict(ccdata_scaled)

ccdata_trimmed = ccdata_scaled[labels == 1]
```

- Does this changes anything about your cluster analysis?