Question 1

In [1]:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
```

Import csv into dataframe

In [2]:

```python
df = pd.read_csv("spenddata.csv")
df.head()
```

Out[2]:

	Unnamed: 0	month	var8	var6	a.1	a.2	a.3	a.4	var5	b.5	...	c.276	c.277	c.278	c.279	c.
0	1	1	2.0	NaN	1	5	1	57	34	1	...	1	0	0	0	
1	2	1	2.0	NaN	1	4	1	57	34	2	...	0	0	0	0	
2	3	1	2.0	NaN	1	5	1	57	42	2	...	0	0	0	0	
3	4	1	2.0	NaN	1	6	1	57	34	2	...	0	0	0	0	
4	5	1	2.0	NaN	1	8	1	22	1	1	...	0	0	0	0	

5 rows × 301 columns

In [3]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18379 entries, 0 to 18378
Columns: 301 entries, Unnamed: 0 to t.158
dtypes: float64(134), int64(165), object(2)
memory usage: 42.2+ MB
```

Creation of Dummy Variable + Remove Columns not Used

The dummy variables are used to replace the text variables, such that they can be used for computation in the subsequent steps. Columns that will not be used for computation will also be removed. These columns do not add any value in the clustering algorithm.

In [4]:

```python
def dummy_var9(value):
    if value == "Mono":
        return 1
    elif value == "Multi":
        return 2
    else:
        return 0

df["var9_int"] = df["var9"].apply(dummy_var9)

## remove columns that do not add value
df.drop(columns = ["var9", "month", "Unnamed: 0", "year", "respondent.id"],
        inplace = True)

df.head()
```

Out[4]:

	var8	var6	a.1	a.2	a.3	a.4	var5	b.5	b.6	b.7	...	c.277	c.278	c.279	c.280	c.281	c.28
0	2.0	NaN	1	5	1	57	34	1	NaN	1	...	0	0	0	1	0	
1	2.0	NaN	1	4	1	57	34	2	3.0	1	...	0	0	0	0	0	
2	2.0	NaN	1	5	1	57	42	2	1.0	1	...	0	0	0	0	0	
3	2.0	NaN	1	6	1	57	34	2	4.0	1	...	0	0	0	1	0	
4	2.0	NaN	1	8	1	22	1	1	NaN	1	...	0	0	0	0	0	

5 rows × 297 columns

Replacing NaN values with 0

This is such that it can be used to calculate the standardised values. In this case, we assume that the columns are not binary. Thus, changing the NaN values to 0 will not "change" the value of the data.

In [5]:

```python
# replacing all NaN values with 0
df.fillna(0, inplace = True)
df.head()
```

Out[5]:

	var8	var6	a.1	a.2	a.3	a.4	var5	b.5	b.6	b.7	...	c.277	c.278	c.279	c.280	c.281	c.28
0	2.0	0.0	1	5	1	57	34	1	0.0	1	...	0	0	0	1	0	[illegible]
1	2.0	0.0	1	4	1	57	34	2	3.0	1	...	0	0	0	0	0	[illegible]
2	2.0	0.0	1	5	1	57	42	2	1.0	1	...	0	0	0	0	0	[illegible]
3	2.0	0.0	1	6	1	57	34	2	4.0	1	...	0	0	0	1	0	[illegible]
4	2.0	0.0	1	8	1	22	1	1	0.0	1	...	0	0	0	0	0	[illegible]

5 rows × 297 columns

Transformation Process

Transformation is done for variables that are highly skewed. Values that are lesser than -1 or greater than 1 will be considered as highly skewed.

In [6]:

```python
skewness = df.skew()

var_to_transform = []

for name, value in skewness.iteritems():
    if value > 1 or value < -1:
        var_to_transform.append(name)
print(var_to_transform)
```

```
['var6', 'b.7', 'b.9', 'b.10', 'b.11', 'b.13', 'b.14', 'b.15', 'b.16', 'b.17',
'b.18', 'b.20', 'b.21', 'b.22', 'pov6', 'b.23', 'b.24', 'b.25', 'b.26', 'b.2
7', 'b.28', 'c.29', 'c.30', 'c.31', 'c.32', 'c.34', 'c.35', 'c.36', 'c.37',
'c.39', 'c.40', 'c.41', 'c.42', 'c.43', 'c.44', 'c.45', 'c.46', 'c.47', 'c.4
8', 'c.49', 'c.50', 'c.51', 'c.52', 'c.53', 'c.54', 'c.55', 'c.56', 'c.57',
'c.58', 'c.59', 'c.60', 'c.61', 'c.62', 'b.63', 'c.65', 'c.66', 'c.67', 'c.6
8', 'c.69', 'c.70', 'c.71', 'c.72', 'c.73', 'c.74', 'c.75', 'c.76', 'c.77',
'c.78', 'c.79', 'c.80', 'c.81', 'c.82', 'c.83', 'c.84', 'c.85', 'c.86', 'c.8
7', 'c.88', 'c.89', 'c.90', 'c.91', 'c.92', 'f.105', 'f.106', 'f.107', 'f.10
8', 'f.109', 'f.110', 'f.111', 'f.112', 'f.113', 'f.114', 'f.115', 'f.116',
'f.117', 'f.118', 'f.119', 'f.120', 'f.121', 'f.122', 'f.123', 'a.124', 'var
3', 'var4', 'c.125', 'c.126', 'c.127', 'pp.128', 'pp.129', 'pp.130', 'pp.131',
'pp.132', 'pp.133', 'pp.134', 'c.135', 'c.136', 'c.137', 'c.138', 'c.139', 'c.
140', 'c.141', 'c.142', 'c.143', 'c.144', 'c.145', 'c.146', 'c.147', 'c.148',
't7.149', 't7.150', 't7.151', 't7.152', 't7.153', 't7.154', 't7.155', 't7.15
6', 't7.157', 't7.158', 'c.160', 'c.161', 'c.162', 'c.163', 'c.166', 'c.167',
'c.168', 'c.169', 'c.170', 'c.171', 'c.172', 't7.174', 't7.175', 't7.179', 't
7.180', 't7.181', 't7.182', 'a.183', 'a.184', 'a.185', 'a.186', 'var2', 'totsh
opping.rep', 'var1', 'c.187', 'f.188', 'c.189', 'c.190', 'b.192', 'b.193', 'b.
194', 'b.195', 'c.196', 'c.197', 'c.198', 'c.199', 'c.200', 'c.201', 'c.202',
'c.203', 'c.204', 'c.205', 'c.206', 'c.207', 'c.208', 'c.209', 'c.210', 'c.21
1', 'c.212', 'c.213', 'c.214', 'c.215', 'c.216', 'c.217', 'c.218', 'c.219',
'c.220', 'c.221', 'c.222', 'c.223', 'c.224', 'c.227', 'c.228', 'c.229', 'c.23
0', 'c.232', 'c.233', 'c.237', 'c.238', 'c.239', 'c.240', 'c.241', 'c.242',
'c.243', 'c.244', 'c.247', 'c.248', 'c.249', 'c.250', 'c.251', 'c.252', 'c.25
3', 'c.254', 'c.255', 'c.256', 'c.257', 'c.258', 'c.259', 'c.260', 'c.261',
'c.262', 'c.263', 'c.265', 'c.268', 'c.269', 'c.271', 'c.272', 'c.273', 'c.27
4', 'c.275', 'c.277', 'c.278', 'c.279', 'c.280', 'c.281', 'c.282', 'c.283']
```

In [7]:

```python
for variable in var_to_transform:
    new_name = "log_" + variable
    df[new_name] = np.log(df[variable] + 1)

# removed non-transformed columns
df.drop(columns = var_to_transform, inplace = True)
df.head()
```

Out[7]:

	var8	a.1	a.2	a.3	a.4	var5	b.5	b.6	b.8	b.12	...	log_c.273	log_c.274	log_c.275	log_c.2
0	2.0	1	5	1	57	34	1	0.0	1.0	2	...	0.0	0.0	0.0	(
1	2.0	1	4	1	57	34	2	3.0	1.0	2	...	0.0	0.0	0.0	(
2	2.0	1	5	1	57	42	2	1.0	1.0	1	...	0.0	0.0	0.0	(
3	2.0	1	6	1	57	34	2	4.0	1.0	2	...	0.0	0.0	0.0	(
4	2.0	1	8	1	22	1	1	0.0	2.0	1	...	0.0	0.0	0.0	(

5 rows × 297 columns

Dataset Standardisation

Data standardisation is peorformed such that all the values will be on the same scale.

In [8]:
```
scaler = StandardScaler()
scaler.fit(df)
```
Out[8]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```
In [9]:
```
df_scaled = scaler.transform(df)
df_scaled
```
Out[9]:
```
array([[ 0.41797295,  0.        , -0.54669362, ..., -0.19972256,
        -0.2750927 , -0.13416986],
       [ 0.41797295,  0.        , -0.95477104, ..., -0.19972256,
         3.6351383 , -0.13416986],
       [ 0.41797295,  0.        , -0.54669362, ..., -0.19972256,
        -0.2750927 , -0.13416986],
       ...,
       [-1.39358816,  0.        , -0.54669362, ..., -0.19972256,
        -0.2750927 , -0.13416986],
       [-1.39358816,  0.        , -0.95477104, ..., -0.19972256,
         3.6351383 , -0.13416986],
       [ 1.32375351,  0.        , -0.13861621, ..., -0.19972256,
        -0.2750927 , -0.13416986]])
```
In [10]:
```
df_prepared = pd.DataFrame(df_scaled, columns = df.columns)
df_prepared.head()
```
Out[10]:

	var8	a.1	a.2	a.3	a.4	var5	b.5	b.6	b.8	b.12
0	0.417973	0.0	-0.546694	0.0	0.008724	0.826032	-1.129119	-0.879448	-0.483734	0.757060
1	0.417973	0.0	-0.954771	0.0	0.008724	0.826032	0.885646	1.237100	-0.483734	0.757060
2	0.417973	0.0	-0.546694	0.0	0.008724	1.578506	0.885646	-0.173932	-0.483734	-1.320899
3	0.417973	0.0	-0.138616	0.0	0.008724	0.826032	0.885646	1.942616	-0.483734	0.757060
4	0.417973	0.0	0.677539	0.0	-0.961049	-2.277924	-1.129119	-0.879448	0.239133	-1.320899

5 rows × 297 columns

KMeans Method

The SSE is calculated and the value of K will be defined by the Elbow Method.

In [11]:

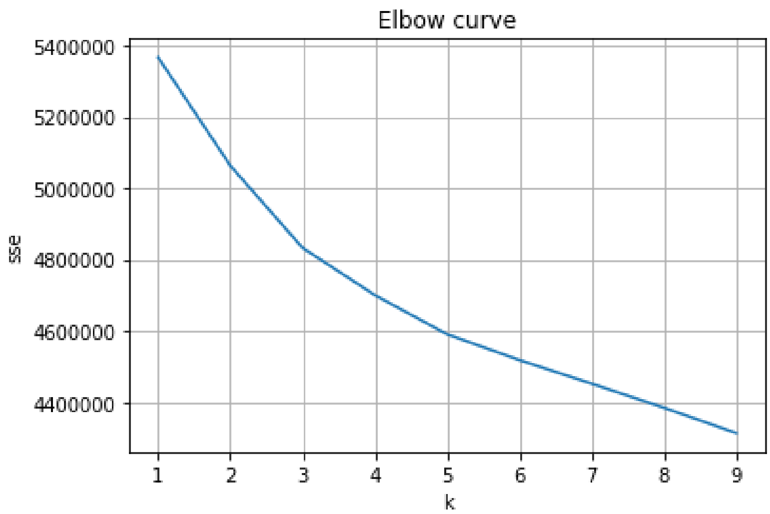
```
sse = []

for i in range (1, 10):
    model = KMeans(n_clusters = i, random_state = 0)
    model.fit(df_prepared)
    sse.append(model.inertia_)

sse
```

Out[11]:

```
[5366668.000000001,
 5063219.498710746,
 4832642.066105606,
 4700390.77190287,
 4590694.495576666,
 4518524.424036916,
 4452508.960699122,
 4384778.9260005765,
 4314006.047367832]
```

In [12]:

```
k = (range(1, 10))
plt.plot(k, sse)
plt.title("Elbow curve")
plt.xlabel("k")
plt.ylabel("sse")
plt.grid(True)
plt.show()
```



Performing Clustering

Based on the Elbow curve that is plotted, K can be either 3 or 5. However, the change in SSE when K = 3 is slightly larger, thus the K = 3 will be used in the clustering algorithm.

In [13]:

```
model = KMeans(n_clusters = 3, random_state = 0)
model.fit(df_prepared)
```

Out[13]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

In [14]:

```
print("SSE:", round(model.inertia_, 2))
print("Number of Iterations:", model.n_iter_)
```

```
SSE: 4832642.07
Number of Iterations: 19
```

In [15]:

```
df["cluster"] = model.labels_
df["cluster"].value_counts()
```

Out[15]:

```
1    9063
0    6817
2    2499
Name: cluster, dtype: int64
```

In [16]:

```
cluster_mean = pd.DataFrame(model.cluster_centers_, columns = df_prepared.(
cluster_mean
```

Out[16]:

	var8	a.1	a.2	a.3	a.4	var5	b.5	b.6	b.8	b.12
0	-0.213164	0.0	0.119627	0.0	0.036770	-0.091919	0.369616	0.035539	-0.040173	0.068472
1	0.188604	0.0	-0.081252	0.0	0.027805	0.016175	-0.132074	0.044036	0.060869	0.054778
2	-0.102516	0.0	-0.031657	0.0	-0.201141	0.192082	-0.529285	-0.256651	-0.111164	-0.385443

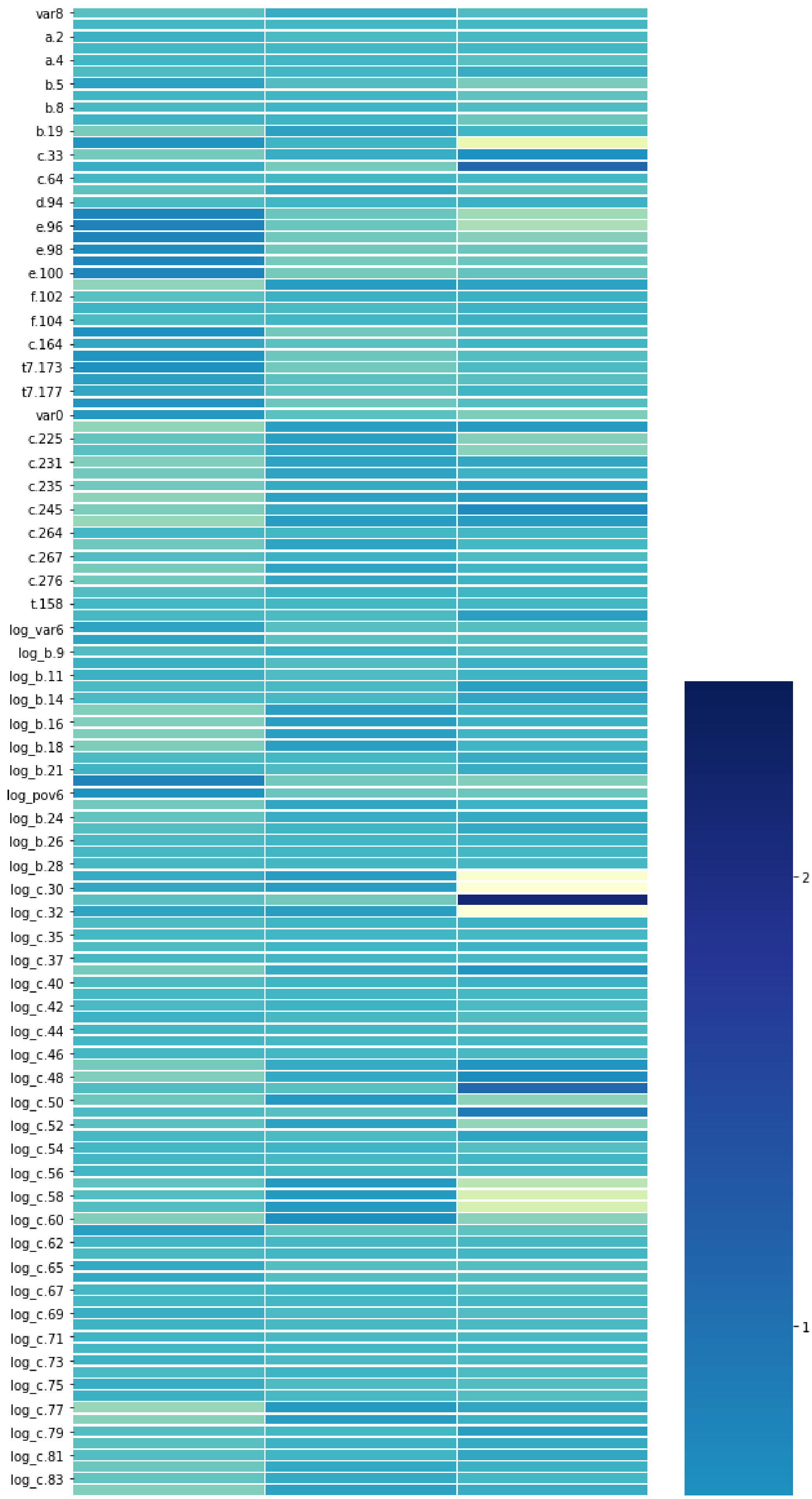
3 rows × 297 columns

Visualising the Results

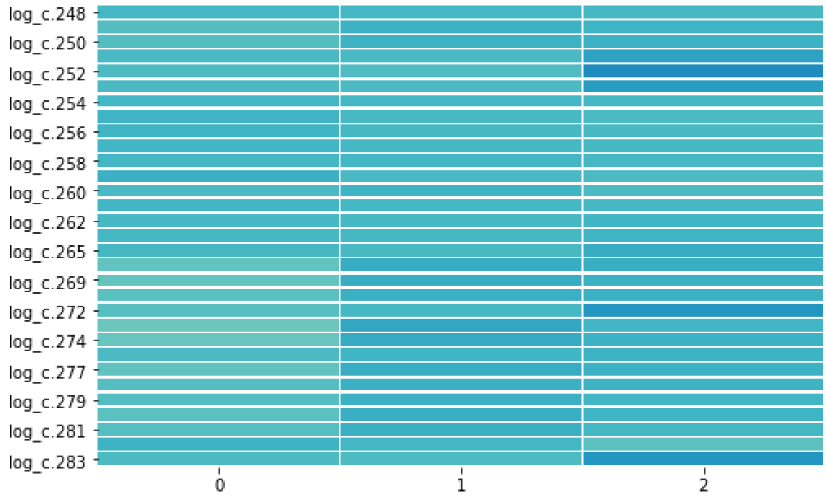
In [17]:

```
import seaborn as sns
plt.figure(figsize = (10,50))
sns.heatmap(cluster_mean.T, linewidths = .5, cmap = "YlGnBu")
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1c14a550b48>

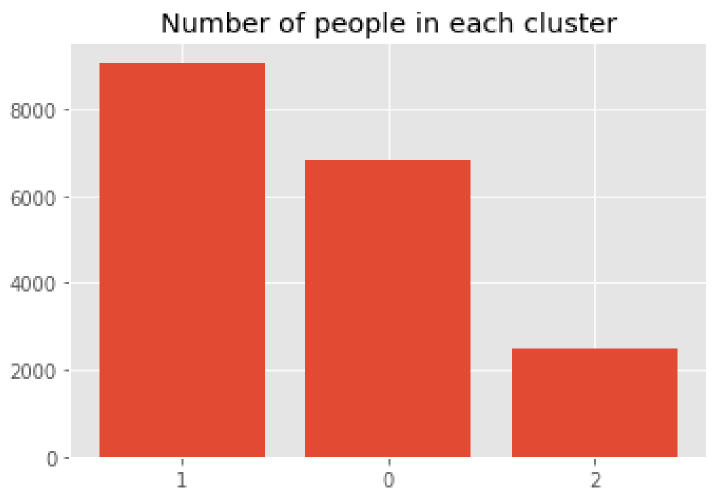


log_c.85
log_c.87
log_c.89
log_c.91
log_f.105
log_f.107
log_f.109
log_f.111
log_f.113
log_f.115
log_f.117
log_f.119
log_f.121
log_f.123
log_var3
log_c.125
log_c.127
log_pp.129
log_pp.131
log_pp.133
log_c.135
log_c.137
log_c.139
log_c.141
log_c.143
log_c.145
log_c.147
log_t7.149
log_t7.151
log_t7.153
log_t7.155
log_t7.157
log_c.160
log_c.162
log_c.166
log_c.168
log_c.170
log_c.172
log_t7.175
log_t7.180
log_t7.182
log_a.184
log_a.186
log_totshopping.rep
log_c.187
log_c.189
log_b.192
log_b.194
log_c.196
log_c.198
log_c.200
log_c.202
log_c.204
log_c.206
log_c.208
log_c.210
log_c.212
log_c.214
log_c.216
log_c.218
log_c.220
log_c.222
log_c.224
log_c.228
log_c.230
log_c.233
log_c.238
log_c.240
log_c.242
log_c.244
0
−1
−2



In [19]:

```
clusters = df["cluster"].value_counts()

cluster_index = []
cluster_value = []

for index, value in clusters.iteritems():
    cluster_index.append(str(index))
    cluster_value.append(value)

plt.bar(cluster_index, cluster_value)
plt.style.use("ggplot")
plt.title("Number of people in each cluster")
plt.show()
```



Based on the results, we see that:

- Cluster 0: Generally light colours, which might indicate that they are the lower spenders
- Cluster 1: Colours are in between clusters 0 and 2, thus indicating that they belong to mid range
- Cluster 2: Colours are darker, thus indicating that they are the higher spenders

Most respondents come from cluster 1, followed by cluster 0 and 2.