In [ ]:

```
import pandas as pd
import seaborn as sns
import numpy as np
```

In [ ]:

```
# Downloading the data from google drive
data = pd.read_csv("https://drive.google.com/uc?export=download&id=1m4Uc7-pknSFcPhsC1TSPjZ8hvOkpE-kU")
```

In [ ]:

```
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

Out[ ]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer
0	5524	1957	Graduation	Single	58138.0	0	0	04/09/2012
1	2174	1954	Graduation	Single	46344.0	1	1	08/03/2014
2	4141	1965	Graduation	Together	71613.0	0	0	21/08/2013
3	6182	1984	Graduation	Together	26646.0	1	0	10/02/2014
4	5324	1981	PhD	Married	58293.0	1	0	19/01/2014

In [ ]:

```
# Dropping what we are not using
redundant = data.loc[:, ["ID", "Z_CostContact", "Z_Revenue", "AcceptedCmp1", "AcceptedC
mp2",
             "AcceptedCmp3", "AcceptedCmp4", "AcceptedCmp5", "Response",
             "NumCatalogPurchases", "NumDealsPurchases", "NumWebPurchases",
             "NumStorePurchases", "NumWebVisitsMonth", "Dt_Customer"]]
data = data.drop(redundant, axis=1)
```

In [ ]:

```
# Checking if the features were dropped
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Year_Birth        2240 non-null   int64
 1   Education         2240 non-null   object
 2   Marital_Status    2240 non-null   object
 3   Income            2216 non-null   float64
 4   Kidhome           2240 non-null   int64
 5   Teenhome          2240 non-null   int64
 6   Recency           2240 non-null   int64
 7   MntWines          2240 non-null   int64
 8   MntFruits         2240 non-null   int64
 9   MntMeatProducts   2240 non-null   int64
 10  MntFishProducts   2240 non-null   int64
 11  MntSweetProducts  2240 non-null   int64
 12  MntGoldProds      2240 non-null   int64
 13  Complain          2240 non-null   int64
dtypes: float64(1), int64(11), object(2)
memory usage: 245.1+ KB
```

In [ ]:

```
data.Marital_Status.value_counts()
```

Out[ ]:

```
Married     864
Together    580
Single      480
Divorced    232
Widow        77
Alone         3
YOLO          2
Absurd        2
Name: Marital_Status, dtype: int64
```

In [ ]:

```
# Deleting nonsensical values from Marital_Status
data.loc[:, "Marital_Status"] = data["Marital_Status"].str.replace("YOLO", "Delete")
data.loc[:, "Marital_Status"] = data["Marital_Status"].str.replace("Absurd", "Delete")

data = data[data["Marital_Status"] != "Delete"]   #uses booleans to check for values th
at are "Delete", then only keeps the True results
```

In [ ]:

```
data.Marital_Status.unique()
```

Out[ ]:

```
array(['Single', 'Together', 'Married', 'Divorced', 'Widow', 'Alone'],
      dtype=object)
```

In [ ]:

```
# Replacing strings with ints so that the data can be used better for the analysis late
r
replacementwithzero = {"Single": "0", "Divorced": "0", "Widow": "0", "Alone": "0", "Mar
ried": "1", "Together": "1"}

data.Marital_Status = [replacementwithzero[item] for item in data.Marital_Status]
```

In [ ]:

```
data["Marital_Status"].value_counts()
```

Out[ ]:

```
1    1444
0     792
Name: Marital_Status, dtype: int64
```

In [ ]:

```
# Doing the same as in Marital_Status but for Education
# Phd (1), Master (1) + 2nd Cycle (1), Graduation (1), Basic (0)

# Create a dictionary
edu = {"PhD": 1, "Master": 1, "2n Cycle": 1, "Graduation": 1, "Basic": 0}

data.Education = [edu[item] for item in data.Education]
```

In [ ]:

```
# Checking if it worked
data["Education"].value_counts()
```

Out[ ]:

```
1    2182
0      54
Name: Education, dtype: int64
```

In [ ]:

```
data.describe()
```

Out[ ]:

	Year_Birth	Education	Income	Kidhome	Teenhome	Recency	Mı
count	2236.000000	2236.00000	2212.000000	2236.000000	2236.000000	2236.000000	2236
mean	1968.796512	0.97585	52232.510850	0.444991	0.506261	49.147138	303
std	11.980604	0.15355	25187.455359	0.538551	0.544615	28.954880	336
min	1893.000000	0.00000	1730.000000	0.000000	0.000000	0.000000	0
25%	1959.000000	1.00000	35233.500000	0.000000	0.000000	24.000000	23
50%	1970.000000	1.00000	51381.500000	0.000000	0.000000	49.000000	173
75%	1977.000000	1.00000	68522.000000	1.000000	1.000000	74.000000	505
max	1996.000000	1.00000	666666.000000	2.000000	2.000000	99.000000	1493

Dropping the missing variables. There are very few of them so it is not worth replacing it with other values such as dummy values.

In [ ]:

```
data = data.dropna()
```

Adding up the various columns that can be combined to calculate total spendings.

In [ ]:

```
amount_spent = data.loc[:, ["MntWines", "MntFruits", "MntMeatProducts", "MntFishProduct
s", "MntSweetProducts", "MntGoldProds"]]

data["total_amount_spent"] = pd.DataFrame(amount_spent.sum(axis=1))
```

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2212 entries, 0 to 2239
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Year_Birth          2212 non-null   int64
 1   Education           2212 non-null   int64
 2   Marital_Status      2212 non-null   object
 3   Income              2212 non-null   float64
 4   Kidhome             2212 non-null   int64
 5   Teenhome            2212 non-null   int64
 6   Recency             2212 non-null   int64
 7   MntWines            2212 non-null   int64
 8   MntFruits           2212 non-null   int64
 9   MntMeatProducts     2212 non-null   int64
 10  MntFishProducts     2212 non-null   int64
 11  MntSweetProducts    2212 non-null   int64
 12  MntGoldProds        2212 non-null   int64
 13  Complain            2212 non-null   int64
 14  total_amount_spent  2212 non-null   int64
dtypes: float64(1), int64(13), object(1)
memory usage: 276.5+ KB
```

When it comes to deleting outliers, we chose to delete outliers if there were values beyond the whiskers of the boxplot.

In [ ]:

```
# Deleting

data = data.drop(["Recency", "MntWines", "MntFruits", "MntMeatProducts", "MntFishProducts", "MntSweetProducts", "MntGoldProds"], axis=1)
```

The following variables are removed from the dataset: "MntWines", "MntFruits", "MntMeatProducts", "MntFishProducts", "MntSweetProducts", "MntGoldProds" as they are exressed in the "total_amount_spent". "Recency" is removed as well as the date of purchase isn't seen as relevant.

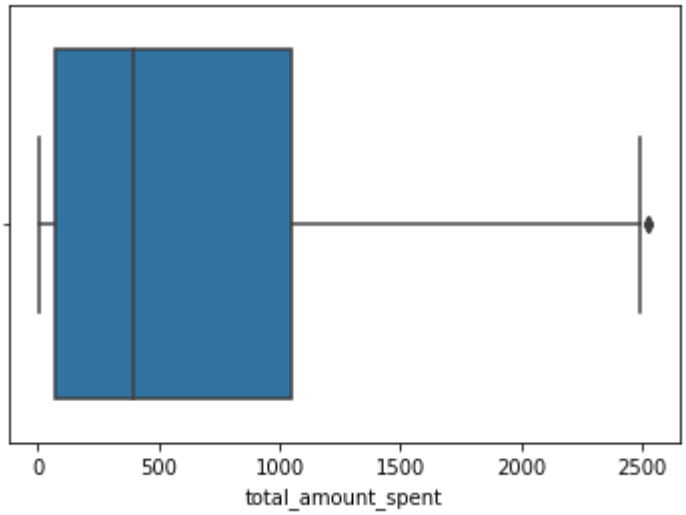
In [ ]:

```
# Creating boxplot to check for outliers
sns.boxplot(data["total_amount_spent"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a89a1b690>
```



In [ ]:

```
# Remove outliers for total_amount_spent
iqr_total_amount_spent = data["total_amount_spent"].quantile(0.75) - data["total_amount
_spent"].quantile(0.25)
```

In [ ]:

```
upper_priceTMS = data["total_amount_spent"].quantile(0.75) + 1.5*iqr_total_amount_spent
```

In [ ]:

```
# Removing the outlier observations using the column total_amount_spent
data = data[data["total_amount_spent"] <= upper_priceTMS]

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2209 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Year_Birth          2209 non-null   int64  
 1   Education           2209 non-null   int64  
 2   Marital_Status      2209 non-null   object 
 3   Income              2209 non-null   float64
 4   Kidhome             2209 non-null   int64  
 5   Teenhome            2209 non-null   int64  
 6   Complain            2209 non-null   int64  
 7   total_amount_spent  2209 non-null   int64  
dtypes: float64(1), int64(6), object(1)
memory usage: 155.3+ KB
```

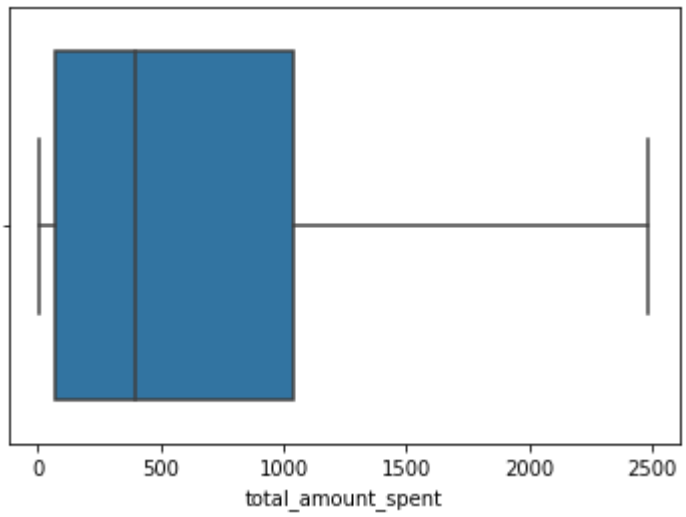
In [ ]:

```
# Checking if it worked
sns.boxplot(data["total_amount_spent"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a899dc190>
```



In [ ]:

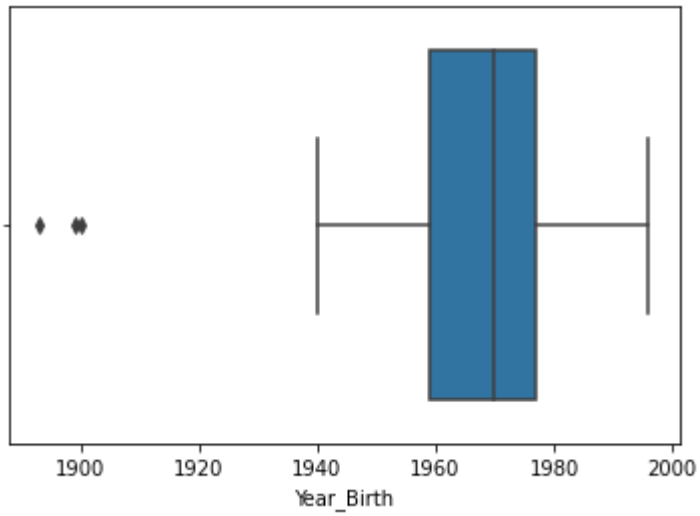
```
# Checking outliers
# Remove outliers that are past the whiskers of the boxplot - this column will be repla
ced with the "Age" column

sns.boxplot(data["Year_Birth"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a899bcf90>
```



In [ ]:

```
# Converting "Year_Birth" to age and adding a new column "Age"
data ["Age"] = 2021 - data["Year_Birth"]
```

In [ ]:

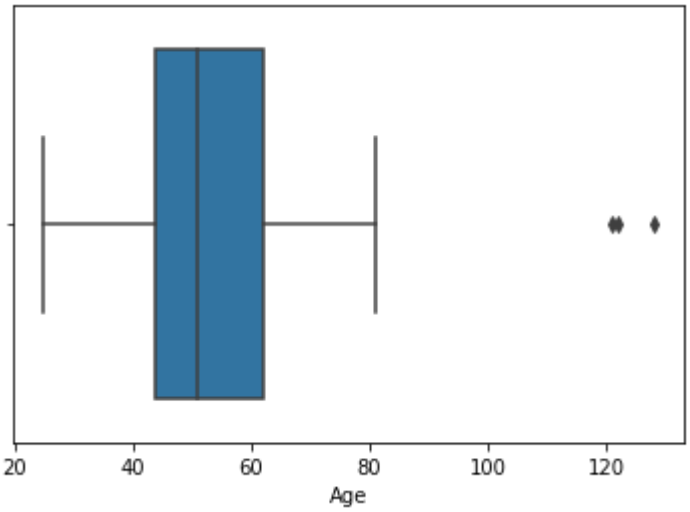
```
# Checking outliers
# Remove outliers that are past the whiskers of the boxplot

sns.boxplot(data["Age"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a899251d0>
```



In [ ]:

```
# Drop the outliers for Age

iqr_Age = data["Age"].quantile(0.75) - data["Age"].quantile(0.25)
upper_priceTMS = data["Age"].quantile(0.75) + 1.5*iqr_Age
data = data[data["Age"] <= upper_priceTMS]
```

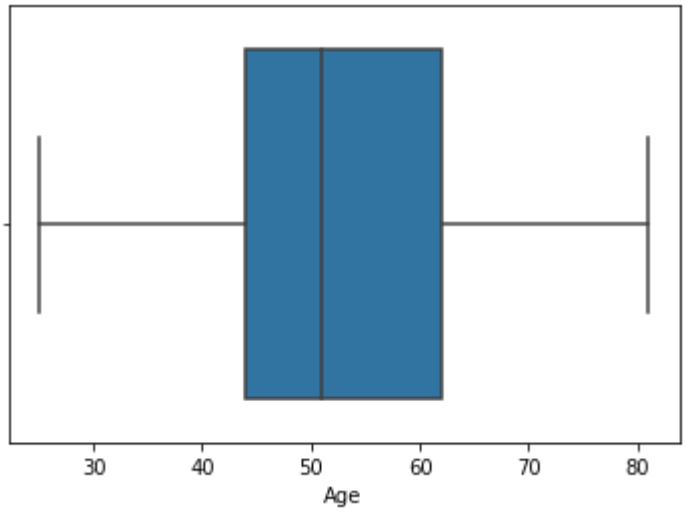
In [ ]:

```
# Checking if data was dropped correctly
sns.boxplot(data["Age"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a89c62610>
```



In [ ]:

```
# Delete the "Year_Birth" since it no longer useful for us because we have the Age for
 each customer
data = data.drop(["Year_Birth"], axis=1)
```

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2206 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Education           2206 non-null   int64
 1   Marital_Status      2206 non-null   object
 2   Income              2206 non-null   float64
 3   Kidhome             2206 non-null   int64
 4   Teenhome            2206 non-null   int64
 5   Complain            2206 non-null   int64
 6   total_amount_spent  2206 non-null   int64
 7   Age                 2206 non-null   int64
dtypes: float64(1), int64(6), object(1)
memory usage: 155.1+ KB
```

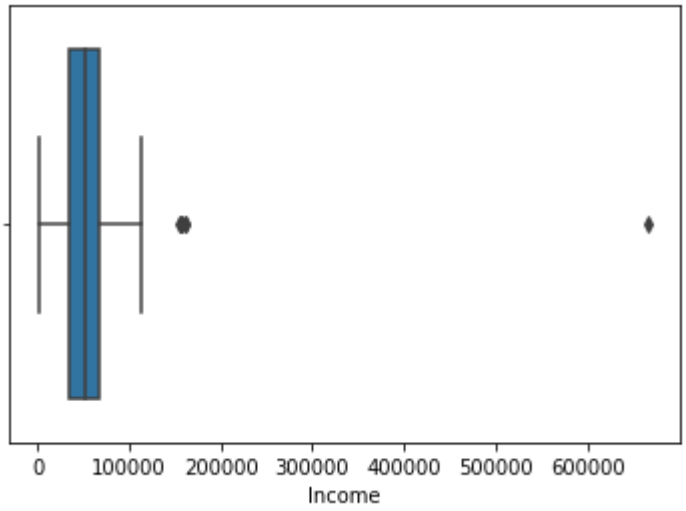
In [ ]:

```
# Check outliers for income
sns.boxplot(data["Income"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a89c38590>
```



In [ ]:

```
# Dropping the outliers for Income
iqr_Income = data["Income"].quantile(0.75) - data["Income"].quantile(0.25)
upper_priceTMS = data["Income"].quantile(0.75) + 1.5*iqr_Income
data = data[data["Income"] <= upper_priceTMS]
```

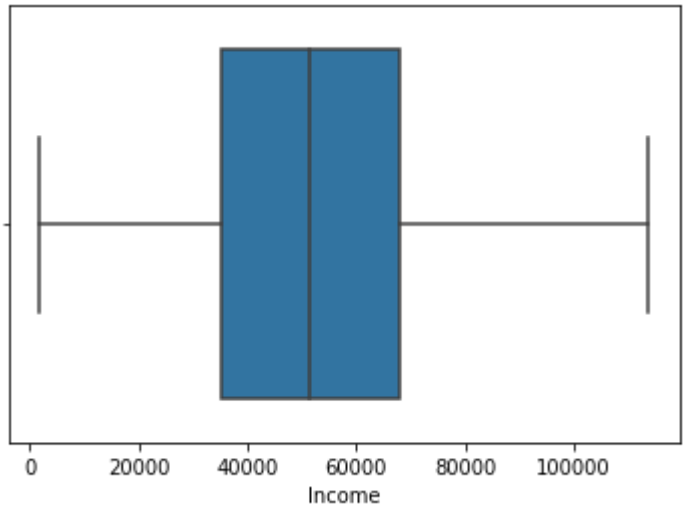
In [ ]:

```
sns.boxplot(data["Income"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a8a3cc610>
```



In [ ]:

```
# There are no outliers in the "Kidhome" or the "Teenhome" columns, therefore no need f
or deleting outliers
data["Teenhome"].value_counts()
```

Out[ ]:

```
0    1134
1    1013
2      51
Name: Teenhome, dtype: int64
```

In [ ]:

```
data["Kidhome"].value_counts()
```

Out[ ]:

```
0    1269
1     883
2      46
Name: Kidhome, dtype: int64
```

Unsupervised Machine Learning

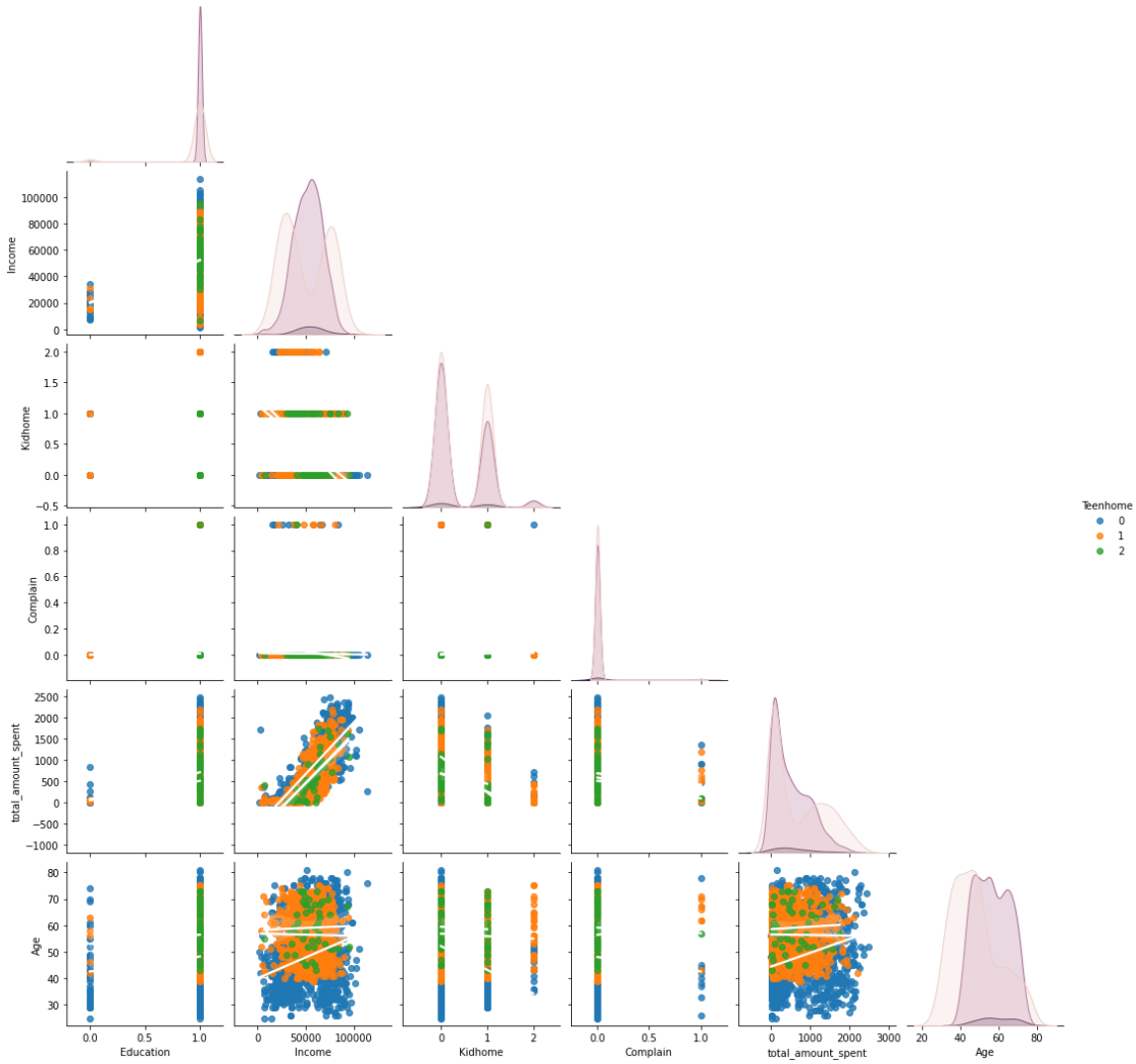
In [ ]:

```
# In order to get an overview of the correlations a pairplot is created.

# The pairplot shows that as the "Income" increases, so does the "total_amount_spent".
# There seems to be no corrolation between the other variables.
sns.pairplot(data, hue='Teenhome', kind="reg", corner=True, plot_kws={'line_kws':{'colo
r':'white'}})
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x7f9a8a6ce850>
```



Dimensionality reduction

To find patterns in the data, the dataset's dimensions are reduced and re-expressed in a compressed form. This process is called dimensionality reduction and is a part of unsupervised machine learning. Principal Component (PCA) is a dimensinality deduction method as it removes "noise" from the dataset and presents the principal components, e.i. the most important components which describe the directions the dataset varies the most. This is expressed through the variance of each component. In the following PCA is performed and then followed by Uniform Manifold Approximation and Projection (UMAP) which is another form of dimensionality reduction. Where PCA performs linear transformation UMAP is nonlinear, which may show other patterns.

In order to perform a PCA the data have to be meaured on the same scale. To do this we import and use StandardScaler from sklearn. Applying the StandardScaler will give all variables a mean of 0 and a standard deviation of 1

In [ ]:

```
# Standard Scaler is imported from sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [ ]:

```
# All variables are scaled
scaled_data = StandardScaler().fit_transform(data.iloc[:,0:-1])
```

In [ ]:

```
# The test performed below shows that the mean is 0 and the std is 1
np.mean(scaled_data), np.std(scaled_data)
```

Out[ ]:

```
(1.154528038086735e-17, 1.0)
```

In order to determine a good number of components to continue with, the PCA components are plotted on the x-axis in the bar chart and the y-axis show the variance for each component. The variance descibes the importance of each component.

In [ ]:

```
#placeholder
from sklearn.decomposition import PCA
pca = PCA()
pca = PCA(n_components=7)
```

In [ ]:

```
import matplotlib.pyplot as plt
```

The plot shows that the first two components (Index 0 and 1) have the highest variance, approximately 2.5 and 1.5. The next three (Index 2, 3, and 4) have almost the same variance just close to 1. From there the variance for Index 5, 6, 7 decreases rapidly from about 0.6.

Principal Component Analysis

In [ ]:

```
# To perform a Principal Component Analysis and reduce dimensionality we need to import
PCA
pca = PCA(n_components=7)
```

The number of principal components is set to 7. We tried different numbers of components, but the data from the PCA analysis did not show us much if we used less components.

In [ ]:

```
# The next step is to use to fit_transform the scaled_data to pca_data
pca_data = pca.fit_transform(scaled_data)
```

In [ ]:

```
# Checking what the data dimensions are
# pca_data consists fo 2198 rows and 7 columns, corrosponding to 7 components
pca_data.shape
```

Out[ ]:

```
(2198, 7)
```

The first two components are plotted on a scatter plot, which shows 3 horizontal groups at different heights of the y-axis. We tried setting the hue to different variables. What presented most salient was 'Teenhome' and 'total_amount_spent'.

Hue = 'Teenhome' shows the 3 horizontal groups which are all split in two. The groups 0 and 1 are significantly more dense than group 2.

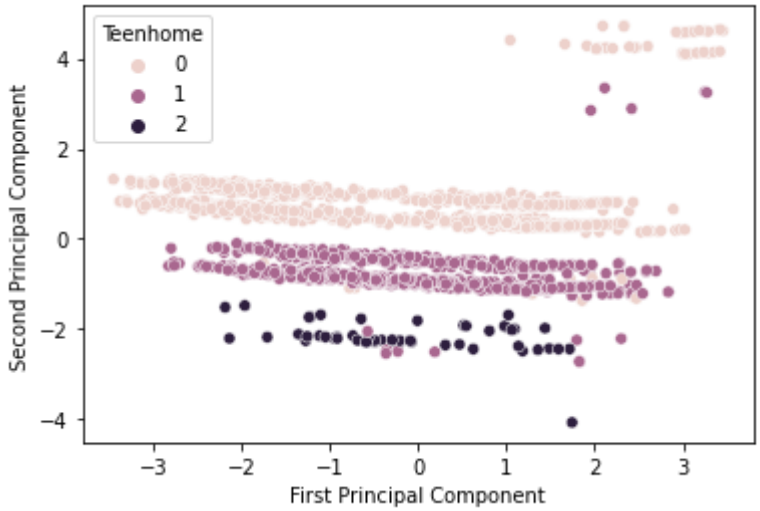
In [ ]:

```
sns.scatterplot(pca_data[:,0], pca_data[:,1], hue = data['Teenhome'])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
Text(0, 0.5, 'Second Principal Component')
```



At the scatterplot where hue = 'total_amount_spent', the groups 500, 1000, 1500, and 2000 show themselves in a gradient across the groups mentioned earlier.

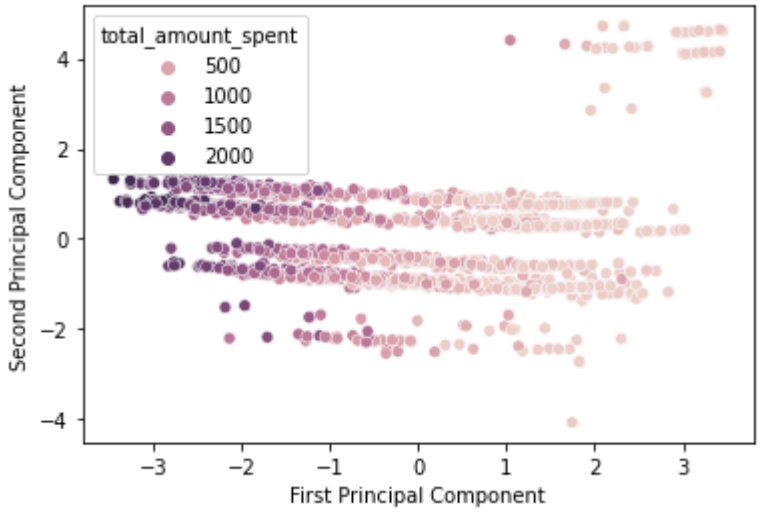
In [ ]:

```
sns.scatterplot(pca_data[:,0], pca_data[:,1], hue = data['total_amount_spent'])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
Text(0, 0.5, 'Second Principal Component')
```



When we look at the cumulative explained variance, which shows to which degree a given number of components describe the data, we see that 7 components describe the data with 100% accuracy, 1 component with 33% accuracy and the rest in between.

In [ ]:

```
# Tells us how accurate our results are according to how many features we use.
# Using more components we get a better result as it should be, and we end up with 1 wh
en using all variables.
pca.explained_variance_ratio_.cumsum()
```

Out[ ]:

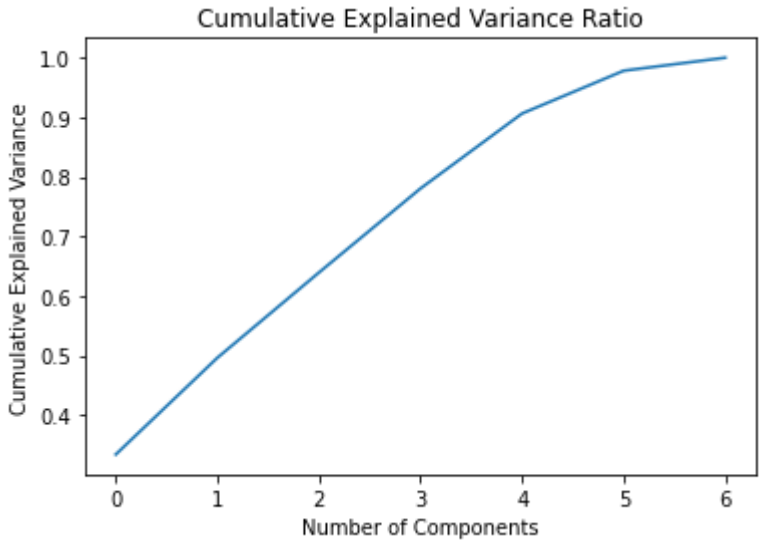
```
array([0.33400028, 0.4962499 , 0.63936649, 0.7804905 , 0.90636245,
       0.97818724, 1.        ])
```

In [ ]:

```
# The cumulative explained variance visually presented
cev = PCA().fit_transform(scaled_data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance Ratio')
```

Out[ ]:

```
Text(0.5, 1.0, 'Cumulative Explained Variance Ratio')
```



The graph shows two slight changes in slope at index 4 and 5 which could indicate that 5 or 6 components might be good choices, which also does makes sense as they respectively depict the data with 90% and 97% accuracy.

UMAP

The next dimensionality reduction method we try is UMAP to see if it performs better than PCA.

In [ ]:

```
# UMAP is installed
!pip install umap-learn[plot]
```

```
Collecting umap-learn[plot]
  Downloading umap-learn-0.5.1.tar.gz (80 kB)
     |████████████████████████████████| 80 kB 2.7 MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dis
t-packages (from umap-learn[plot]) (1.19.5)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python
3.7/dist-packages (from umap-learn[plot]) (0.22.2.post1)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist
-packages (from umap-learn[plot]) (1.4.1)
Requirement already satisfied: numba>=0.49 in /usr/local/lib/python3.7/dis
t-packages (from umap-learn[plot]) (0.51.2)
Collecting pynndescent>=0.5
  Downloading pynndescent-0.5.4.tar.gz (1.1 MB)
     |████████████████████████████████| 1.1 MB 20.3 MB/s
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-pac
kages (from umap-learn[plot]) (1.1.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist
-packages (from umap-learn[plot]) (3.2.2)
Collecting datashader
  Downloading datashader-0.13.0-py2.py3-none-any.whl (15.8 MB)
     |████████████████████████████████| 15.8 MB 643 bytes/s
Requirement already satisfied: bokeh in /usr/local/lib/python3.7/dist-pack
ages (from umap-learn[plot]) (2.3.3)
Requirement already satisfied: holoviews in /usr/local/lib/python3.7/dist-
packages (from umap-learn[plot]) (1.14.5)
Requirement already satisfied: colorcet in /usr/local/lib/python3.7/dist-p
ackages (from umap-learn[plot]) (2.0.6)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-pa
ckages (from umap-learn[plot]) (0.11.2)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.7/di
st-packages (from umap-learn[plot]) (0.16.2)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/l
ib/python3.7/dist-packages (from numba>=0.49->umap-learn[plot]) (0.34.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist
-packages (from numba>=0.49->umap-learn[plot]) (57.4.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/di
st-packages (from pynndescent>=0.5->umap-learn[plot]) (1.0.1)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.7/di
st-packages (from bokeh->umap-learn[plot]) (3.13)
Requirement already satisfied: packaging>=16.8 in /usr/local/lib/python3.
7/dist-packages (from bokeh->umap-learn[plot]) (21.0)
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/
python3.7/dist-packages (from bokeh->umap-learn[plot]) (3.7.4.3)
Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.7/di
st-packages (from bokeh->umap-learn[plot]) (5.1.1)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.7/dis
t-packages (from bokeh->umap-learn[plot]) (2.11.3)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/pyth
on3.7/dist-packages (from bokeh->umap-learn[plot]) (2.8.2)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.7/d
ist-packages (from bokeh->umap-learn[plot]) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.
7/dist-packages (from Jinja2>=2.9->bokeh->umap-learn[plot]) (2.0.1)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.
7/dist-packages (from packaging>=16.8->bokeh->umap-learn[plot]) (2.4.7)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-p
ackages (from python-dateutil>=2.1->bokeh->umap-learn[plot]) (1.15.0)
Requirement already satisfied: param>=1.7.0 in /usr/local/lib/python3.7/di
st-packages (from colorcet->umap-learn[plot]) (1.11.1)
Requirement already satisfied: pyct>=0.4.4 in /usr/local/lib/python3.7/dis
t-packages (from colorcet->umap-learn[plot]) (0.4.8)
```

```
Requirement already satisfied: xarray>=0.9.6 in /usr/local/lib/python3.7/d
ist-packages (from datashader->umap-learn[plot]) (0.18.2)
Collecting datashape>=0.5.1
  Downloading datashape-0.5.2.tar.gz (76 kB)
     |████████████████████████████████| 76 kB 3.8 MB/s
Requirement already satisfied: dask[complete]>=0.18.0 in /usr/local/lib/py
thon3.7/dist-packages (from datashader->umap-learn[plot]) (2.12.0)
Requirement already satisfied: toolz>=0.7.3 in /usr/local/lib/python3.7/di
st-packages (from dask[complete]>=0.18.0->datashader->umap-learn[plot])
(0.11.1)
Collecting distributed>=2.0
  Downloading distributed-2021.9.1-py3-none-any.whl (786 kB)
     |████████████████████████████████| 786 kB 65.0 MB/s
Collecting fsspec>=0.6.0
  Downloading fsspec-2021.9.0-py3-none-any.whl (123 kB)
     |████████████████████████████████| 123 kB 51.3 MB/s
Requirement already satisfied: cloudpickle>=0.2.1 in /usr/local/lib/python
3.7/dist-packages (from dask[complete]>=0.18.0->datashader->umap-learn[plo
t]) (1.3.0)
Collecting partd>=0.3.10
  Downloading partd-1.2.0-py3-none-any.whl (19 kB)
Collecting multipledispatch>=0.4.7
  Downloading multipledispatch-0.6.0-py3-none-any.whl (11 kB)
Requirement already satisfied: zict>=0.1.3 in /usr/local/lib/python3.7/dis
t-packages (from distributed>=2.0->dask[complete]>=0.18.0->datashader->uma
p-learn[plot]) (2.0.0)
Collecting cloudpickle>=0.2.1
  Downloading cloudpickle-2.0.0-py3-none-any.whl (25 kB)
Requirement already satisfied: click>=6.6 in /usr/local/lib/python3.7/dist
-packages (from distributed>=2.0->dask[complete]>=0.18.0->datashader->umap
-learn[plot]) (7.1.2)
Requirement already satisfied: msgpack>=0.6.0 in /usr/local/lib/python3.7/
dist-packages (from distributed>=2.0->dask[complete]>=0.18.0->datashader->
umap-learn[plot]) (1.0.2)
Requirement already satisfied: psutil>=5.0 in /usr/local/lib/python3.7/dis
t-packages (from distributed>=2.0->dask[complete]>=0.18.0->datashader->uma
p-learn[plot]) (5.4.8)
Collecting distributed>=2.0
  Downloading distributed-2021.9.0-py3-none-any.whl (779 kB)
     |████████████████████████████████| 779 kB 69.7 MB/s
  Downloading distributed-2021.8.1-py3-none-any.whl (778 kB)
     |████████████████████████████████| 778 kB 67.3 MB/s
  Downloading distributed-2021.8.0-py3-none-any.whl (776 kB)
     |████████████████████████████████| 776 kB 71.5 MB/s
Requirement already satisfied: tblib>=1.6.0 in /usr/local/lib/python3.7/di
st-packages (from distributed>=2.0->dask[complete]>=0.18.0->datashader->um
ap-learn[plot]) (1.7.0)
  Downloading distributed-2021.7.2-py3-none-any.whl (769 kB)
     |████████████████████████████████| 769 kB 66.9 MB/s
  Downloading distributed-2021.7.1-py3-none-any.whl (766 kB)
     |████████████████████████████████| 766 kB 71.0 MB/s
  Downloading distributed-2021.7.0-py3-none-any.whl (1.0 MB)
     |████████████████████████████████| 1.0 MB 47.2 MB/s
  Downloading distributed-2021.6.2-py3-none-any.whl (722 kB)
     |████████████████████████████████| 722 kB 84.4 MB/s
Requirement already satisfied: sortedcontainers!=2.0.0,!=2.0.1 in /usr/loc
al/lib/python3.7/dist-packages (from distributed>=2.0->dask[complete]>=0.1
8.0->datashader->umap-learn[plot]) (2.4.0)
  Downloading distributed-2021.6.1-py3-none-any.whl (722 kB)
     |████████████████████████████████| 722 kB 72.4 MB/s
  Downloading distributed-2021.6.0-py3-none-any.whl (715 kB)
```

```
     |████████████████████████████████| 715 kB 75.8 MB/s
   Downloading distributed-2021.5.1-py3-none-any.whl (705 kB)
     |████████████████████████████████| 705 kB 73.2 MB/s
   Downloading distributed-2021.5.0-py3-none-any.whl (699 kB)
     |████████████████████████████████| 699 kB 73.4 MB/s
   Downloading distributed-2021.4.1-py3-none-any.whl (696 kB)
     |████████████████████████████████| 696 kB 68.2 MB/s
   Downloading distributed-2021.4.0-py3-none-any.whl (684 kB)
     |████████████████████████████████| 684 kB 74.6 MB/s
   Downloading distributed-2021.3.1-py3-none-any.whl (679 kB)
     |████████████████████████████████| 679 kB 72.4 MB/s
   Downloading distributed-2021.3.0-py3-none-any.whl (675 kB)
     |████████████████████████████████| 675 kB 74.3 MB/s
   Downloading distributed-2021.2.0-py3-none-any.whl (675 kB)
     |████████████████████████████████| 675 kB 83.2 MB/s
   Downloading distributed-2021.1.1-py3-none-any.whl (672 kB)
     |████████████████████████████████| 672 kB 83.9 MB/s
   Downloading distributed-2021.1.0-py3-none-any.whl (671 kB)
     |████████████████████████████████| 671 kB 82.4 MB/s
   Downloading distributed-2020.12.0-py3-none-any.whl (669 kB)
     |████████████████████████████████| 669 kB 91.5 MB/s
   Downloading distributed-2.30.1-py3-none-any.whl (656 kB)
     |████████████████████████████████| 656 kB 53.3 MB/s
 Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/di
 st-packages (from pandas->umap-learn[plot]) (2018.9)
 Collecting locket
   Downloading locket-0.2.1-py2.py3-none-any.whl (4.1 kB)
 Requirement already satisfied: heapdict in /usr/local/lib/python3.7/dist-p
 ackages (from zict>=0.1.3->distributed>=2.0->dask[complete]>=0.18.0->datas
 hader->umap-learn[plot]) (1.0.1)
 Requirement already satisfied: pyviz-comms>=0.7.4 in /usr/local/lib/python
 3.7/dist-packages (from holoviews->umap-learn[plot]) (2.1.0)
 Requirement already satisfied: panel>=0.8.0 in /usr/local/lib/python3.7/di
 st-packages (from holoviews->umap-learn[plot]) (0.12.1)
 Requirement already satisfied: tqdm>=4.48.0 in /usr/local/lib/python3.7/di
 st-packages (from panel>=0.8.0->holoviews->umap-learn[plot]) (4.62.2)
 Requirement already satisfied: markdown in /usr/local/lib/python3.7/dist-p
 ackages (from panel>=0.8.0->holoviews->umap-learn[plot]) (3.3.4)
 Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-pac
 kages (from panel>=0.8.0->holoviews->umap-learn[plot]) (4.1.0)
 Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-p
 ackages (from panel>=0.8.0->holoviews->umap-learn[plot]) (2.23.0)
 Requirement already satisfied: webencodings in /usr/local/lib/python3.7/di
 st-packages (from bleach->panel>=0.8.0->holoviews->umap-learn[plot]) (0.5.
 1)
 Requirement already satisfied: importlib-metadata in /usr/local/lib/python
 3.7/dist-packages (from markdown->panel>=0.8.0->holoviews->umap-learn[plo
 t]) (4.8.1)
 Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
 packages (from importlib-metadata->markdown->panel>=0.8.0->holoviews->umap
 -learn[plot]) (3.5.0)
 Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python
 3.7/dist-packages (from matplotlib->umap-learn[plot]) (1.3.2)
 Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/di
 st-packages (from matplotlib->umap-learn[plot]) (0.10.0)
 Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/di
 st-packages (from requests->panel>=0.8.0->holoviews->umap-learn[plot]) (2.
 10)
 Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
 /usr/local/lib/python3.7/dist-packages (from requests->panel>=0.8.0->holov
 iews->umap-learn[plot]) (1.24.3)
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.7/dist-packages (from requests->panel>=0.8.0->holoviews->umap-learn[plo
t]) (2021.5.30)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python
3.7/dist-packages (from requests->panel>=0.8.0->holoviews->umap-learn[plo
t]) (3.0.4)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.7/
dist-packages (from scikit-image->umap-learn[plot]) (2.4.1)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.7/d
ist-packages (from scikit-image->umap-learn[plot]) (2.6.3)
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python
3.7/dist-packages (from scikit-image->umap-learn[plot]) (1.1.1)
Building wheels for collected packages: umap-learn, pynndescent, datashape
  Building wheel for umap-learn (setup.py) ... done
  Created wheel for umap-learn: filename=umap_learn-0.5.1-py3-none-any.whl
size=76564 sha256=b7f70f13018a0129a65c206b97ef1bb5e041b4c2e75925758abf6b4d
8dbf651d
  Stored in directory: /root/.cache/pip/wheels/01/e7/bb/347dc0e510803d7116
a13d592b10cc68262da56a8eec4dd72f
  Building wheel for pynndescent (setup.py) ... done
  Created wheel for pynndescent: filename=pynndescent-0.5.4-py3-none-any.w
hl size=52373 sha256=c3e31eab7343010b90282a83387b5b8db2332bbfc7f0d23f3b2b0
60d480ae694
  Stored in directory: /root/.cache/pip/wheels/d0/5b/62/3401692ddad1232424
9c774c4b15ccb046946021e2b581c043
  Building wheel for datashape (setup.py) ... done
  Created wheel for datashape: filename=datashape-0.5.2-py3-none-any.whl s
ize=59438 sha256=39fdf18728a4321a8203cb8d4b43f7d3896b6a6236cd034b7cae1a1b6
70e1075
  Stored in directory: /root/.cache/pip/wheels/b5/b7/80/333a5c3312ed4cd54f
5d5b869868c14e0c6002cb5c7238b52d
Successfully built umap-learn pynndescent datashape
Installing collected packages: locket, cloudpickle, partd, multipledispatc
h, fsspec, distributed, pynndescent, datashape, umap-learn, datashader
  Attempting uninstall: cloudpickle
    Found existing installation: cloudpickle 1.3.0
    Uninstalling cloudpickle-1.3.0:
      Successfully uninstalled cloudpickle-1.3.0
  Attempting uninstall: distributed
    Found existing installation: distributed 1.25.3
    Uninstalling distributed-1.25.3:
      Successfully uninstalled distributed-1.25.3
ERROR: pip's dependency resolver does not currently take into account all
 the packages that are installed. This behaviour is the source of the foll
owing dependency conflicts.
gym 0.17.3 requires cloudpickle<1.7.0,>=1.2.0, but you have cloudpickle 2.
0.0 which is incompatible.
Successfully installed cloudpickle-2.0.0 datashader-0.13.0 datashape-0.5.2
distributed-2.30.1 fsspec-2021.9.0 locket-0.2.1 multipledispatch-0.6.0 par
td-1.2.0 pynndescent-0.5.4 umap-learn-0.5.1
```

In [ ]:

```
# UMAP is imported
import umap
reducer = umap.UMAP()
```

In [ ]:

```
# Reducing the scaled data
umap_data = reducer.fit_transform(scaled_data)
umap_data.shape
```

```
/usr/local/lib/python3.7/dist-packages/numba/np/ufunc/parallel.py:363: Num
baWarning: The TBB threading layer requires TBB version 2019.5 or later i.
e., TBB_INTERFACE_VERSION >= 11005. Found TBB_INTERFACE_VERSION = 9107. Th
e TBB threading layer is disabled.
  warnings.warn(problem)
```

Out[ ]:

```
(2198, 2)
```

To visualize the UMAP we plot it on a scatter plot. As hue we chose "Teenhome".

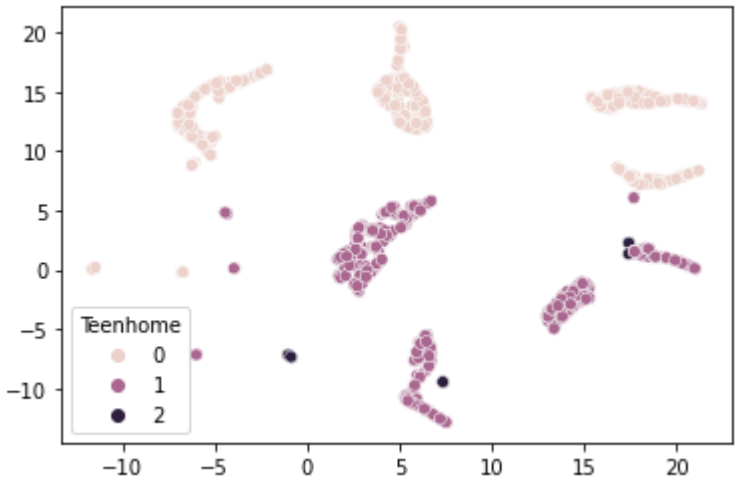
In [ ]:

```
# Plotting the data with hue = 'Teenhome'
sns.scatterplot(umap_data[:,0], umap_data[:,1],hue = data['Teenhome'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a37846dd0>
```



The UMAP shows 8 clear groups where all seem to be tighter than on the PCA. They are clearly divided between Teenhome = 1 and Teenhome = 2. Teenhome = 2 is also visible, but it is not salient.

Below we try UMAP with hue = 'total_amount_spent'.

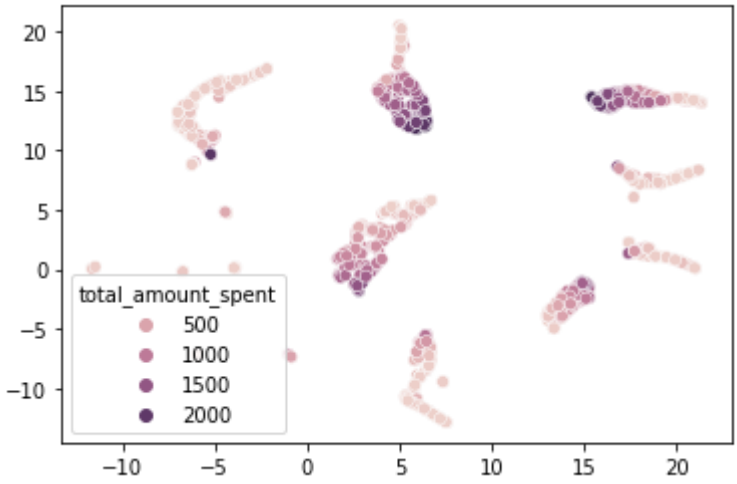
In [ ]:

```
# Plotting the data with hue = 'total_amount_spent'
sns.scatterplot(umap_data[:,0], umap_data[:,1],hue = data['total_amount_spent'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a3817f6d0>
```



With hue = 'total_amount_spent' the groups are the same but the color division is not as clear.

By performing both PCA and UMAP we can see that UMAP reduces the dimensionality better than PCA, and provides tighter groups.

Clustering

KMeans

Clustering with KMeans is performed on the unscaled data and divides the data into k clusters. Data is clustered based on similarities, where there are differences between clusters. The similarity is calculated from the distance measured between datapoints.

Inertia measures the quality of the clustering and descibes how tight the clusters are. When the inertia is plotted against the number of clusters it can be used to determine a good amount of clusters for a particular dataset.

Below we import KMeans, then create a KMeans instance with clusters which we name "model". We fit the model to the data and then append the inertia. From there we plot the numbers of clusters on the x-axis and the inertia on the y-axis.

A good amount of clusters is where the inertia starts to decrease at a slower rate, so on the graph below a good amount would probably be 3 clusters.

In [ ]:

```
from sklearn.cluster import KMeans
```

In [ ]:

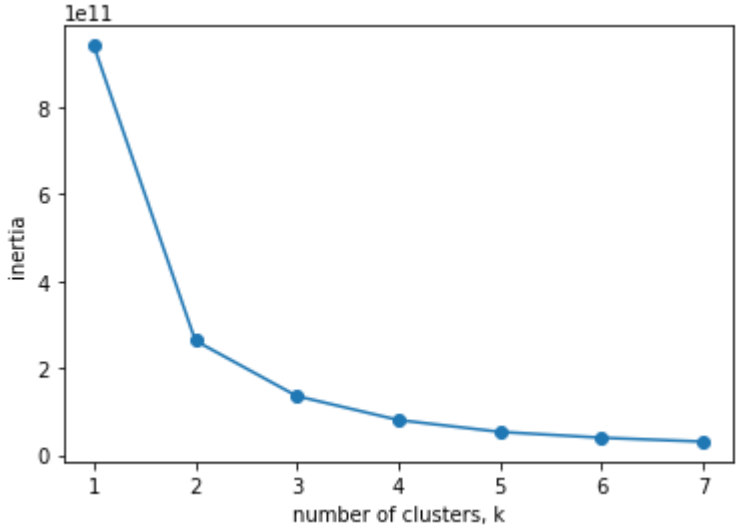
```
ks = range(1, 8)
inertias = []

for k in ks:
  # Create a KMeans instance with clusters: model
  model = KMeans(n_clusters=k)

  # Fit model to samples
  model.fit(data)

  # Append the inertia to the list of inertias
  inertias.append(model.inertia_)

# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



From here we create a clusterer with 3 clusters and fit to the scaled data.

In [ ]:

```
clusterer = KMeans(n_clusters=3)
clusterer.fit(scaled_data)
```

Out[ ]:

```
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=None, tol=0.0001, verbose=0)
```

The UMAP is plotted once again and the cluster.labels are used as hue, which presents the following:

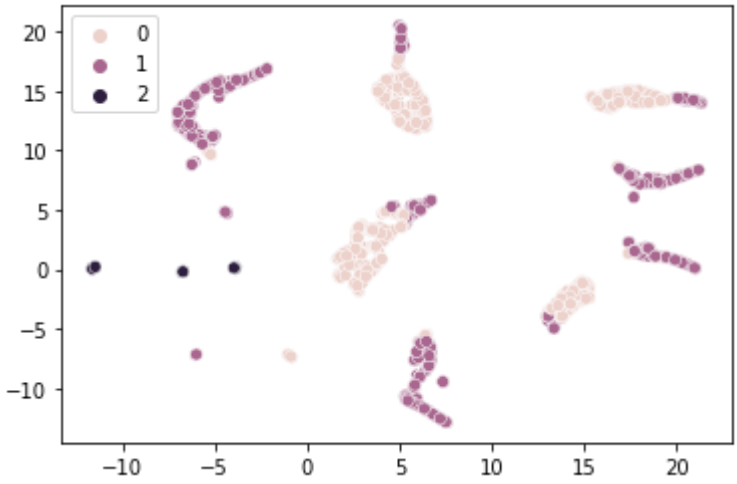
In [ ]:

```
sns.scatterplot(umap_data[:,0], umap_data[:,1], hue = clusterer.labels_)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a3757c610>
```



The plot shows some clear clusters in the right side of the plot, but the rest are mixed.

From here we can perform a crosstabulation between the clusterer.labels and a given variable. Here we choose 'Teenhome'.

'Teenhome' is depicted as the columns and the clusters as rows.

This crosstabulation does not give a clear insights on any correlations in the data as the numbers are spread evenly across the crosstab.

In [ ]:

```
pd.crosstab(clusterer.labels_, data['Teenhome'])
```

Out[ ]:

Teenhome	0	1	2
row_0			
0	513	524	32
1	572	484	19
2	49	5	0

Hierarchical clustering

Another form of clustering is hierarchical clustering. In hierarchical clustering every datapoint begins as a seperate cluster and then in the end all datapoints are in the same cluster.

Below we import AgglomerativeClustering and create a clusterer from it which is fitted to the scaled data. The UMAP is plotted once again and the clusterer label from this clusterer is used as hue.

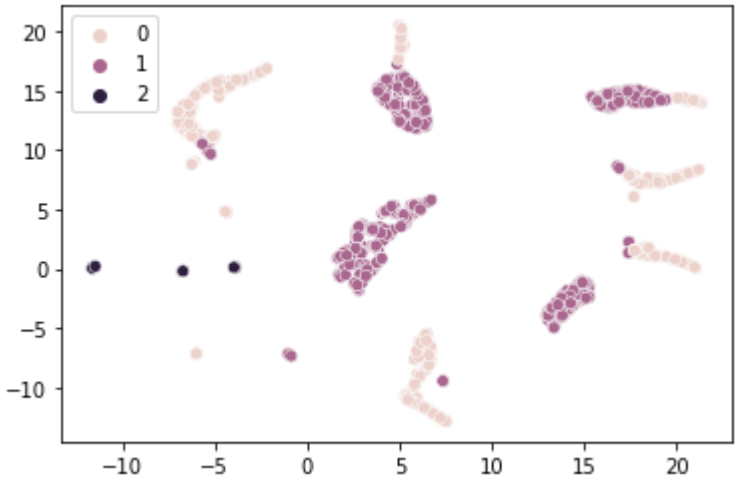
In [ ]:

```
from sklearn.cluster import AgglomerativeClustering
clusterer = AgglomerativeClustering(n_clusters=3).fit(scaled_data)
sns.scatterplot(umap_data[:,0], umap_data[:,1], hue = clusterer.labels_ )
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a37c52ed0>
```



The plot does give 2 clear clusters in the middle of the plot, but again the rest of the clusters are mixed.

Below we perform another crosstabulation, which confirms the mixed picture and that there are no clear correlations.

In [ ]:

```
# Let's do the cross-tab with clusters vs. Marital Status
pd.crosstab(clusterer.labels_, data['Teenhome'])
```

Out[ ]:

Teenhome	0	1	2
row_0			
0	568	396	1
1	517	612	50
2	49	5	0

SML - regression problem: Predict the total amount spent

The regression problem we've chosen consists of using the dataset to predict the total amount that customers may spend on the company's products.

Train-test preparation

In [ ]:

```
# Convert the Income feature to a float
data["Income"] = data["Income"].astype(float)
```

In [ ]:

```
# Take a look at the data, so that we are able to subset it next
data.head()
```

Out[ ]:

	Education	Marital_Status	Income	Kidhome	Teenhome	Complain	total_amount_spent	A
0	1	0	58138.0	0	0	0	1617	
1	1	0	46344.0	1	1	0	27	
2	1	1	71613.0	0	0	0	776	
3	1	1	26646.0	1	0	0	53	
4	1	1	58293.0	1	0	0	422	

In [ ]:

```
# Make a new dataframe with only the variables of interest
# The columns on the amounts spent on various product categories are contained within t
he "total_amount_spent"
# As such, they have an inherent strong positive correlation, which would de redundat f
or the SML problem at hand
# For that reason, we leave them out from the analysis
data_SML = data.loc[:, ["Age", "Income", "Kidhome", "Teenhome", "total_amount_spent",
"Marital_Status", "Education"]]
```

Take a look at the correlation between the variables first

This is done in order to get an initial idea of how(if) the data is correlated

In [ ]:

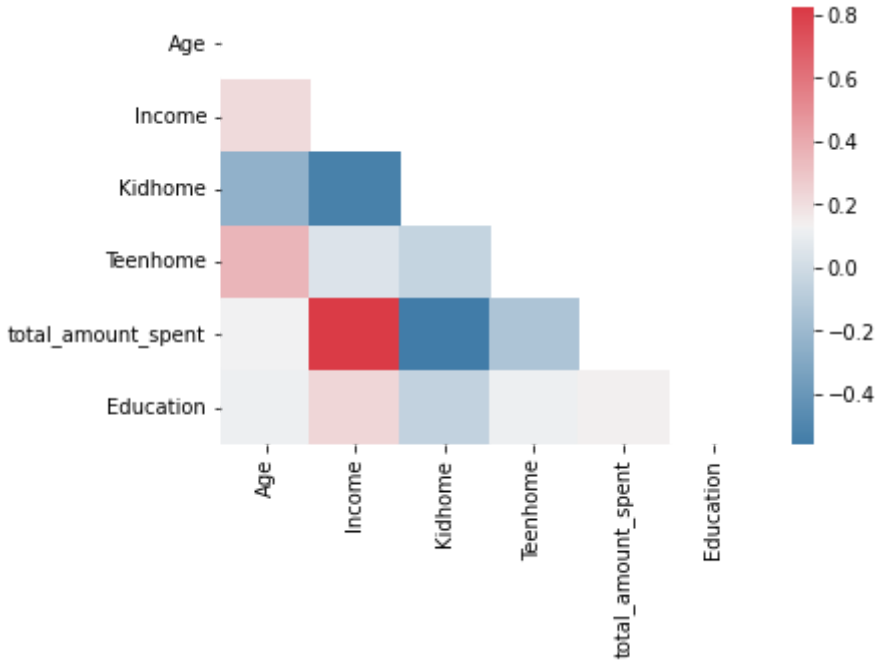
```
# Compute the pairwise correlation of the variables from the data_SML dataframe
corr = data_SML.corr()
```

In [ ]:

```
# Make a customized heatmap; use "cmap" to set the blue-white-red palette
cmap = sns.diverging_palette(240, 10, as_cmap=True)
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr,mask=mask, cmap=cmap)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a8a6cef10>
```



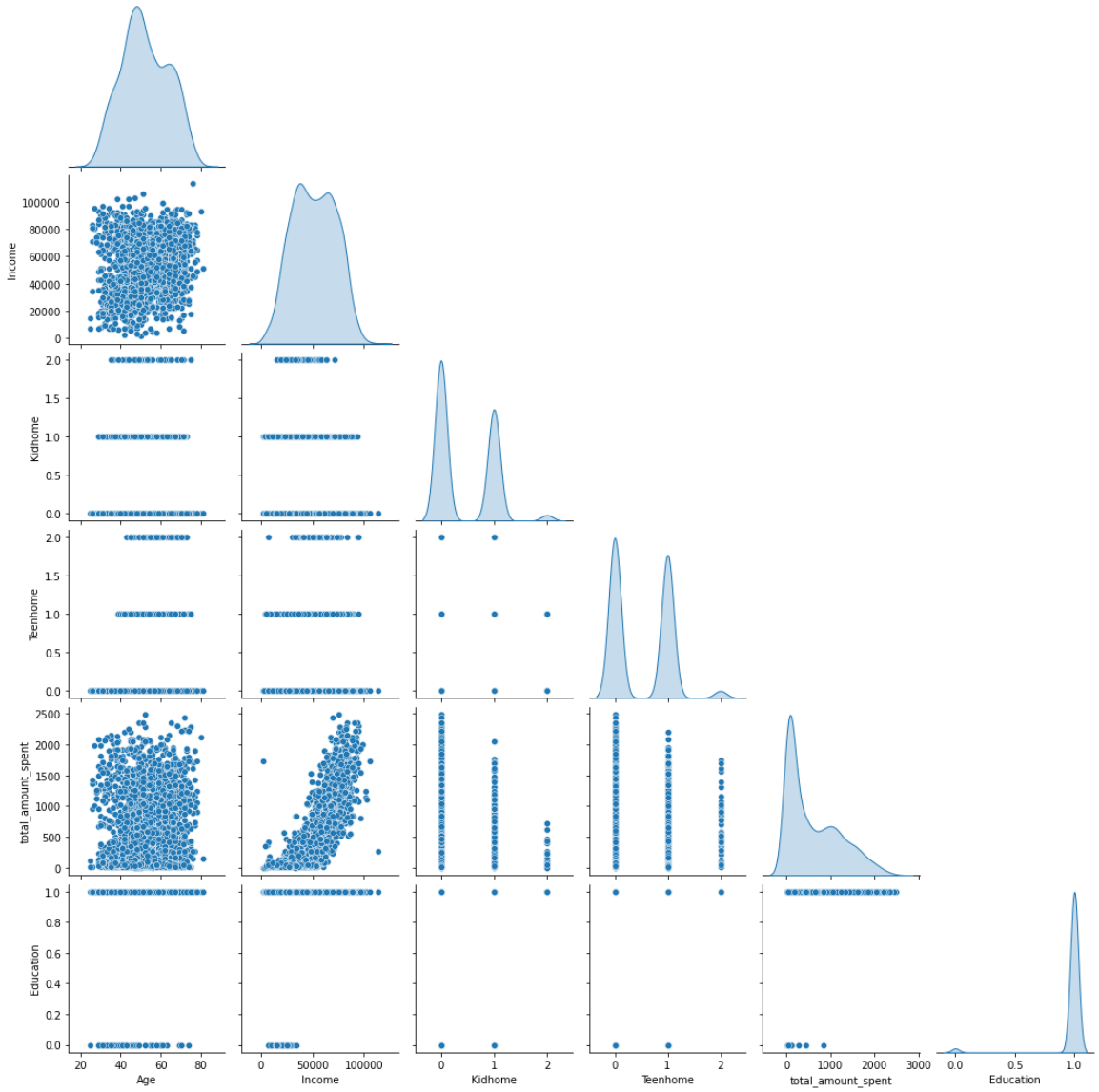
The heatmap shows a relatively strong positive correlation between the total_amount_spent and the Income, and some negative correlation between total_amount_spent and Kidhome

In [ ]:

```
# Plot a pairplot using sns.pairplot()
# Use "corner=True" to get just the lower corner of the grid
# Use "diag_kind= 'kde" to set the type of graph on the diagonal line
sns.pairplot(data_SML, corner=True, diag_kind='kde')
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x7f9a379baed0>
```



Train-test split (again)

In [ ]:

```
# Start out with the y variable which we want to predict: total_amount_spent (which con
sists of the formerly 6 product categories)
y = data_SML["total_amount_spent"]
```

In [ ]:

```
data_SML.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2198 entries, 0 to 2239
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 2198 non-null   int64
 1   Income              2198 non-null   float64
 2   Kidhome             2198 non-null   int64
 3   Teenhome            2198 non-null   int64
 4   total_amount_spent  2198 non-null   int64
 5   Marital_Status      2198 non-null   object
 6   Education           2198 non-null   int64
dtypes: float64(1), int64(5), object(1)
memory usage: 137.4+ KB
```

In [ ]:

```
# X is comprised of the columns from data_SML, except for the total amount spent
X = data_SML.loc[:, ["Age", "Income", "Kidhome", "Teenhome", "Marital_Status", "Educati
on"]]
```

In [ ]:

```
# The data is on different scales, and has to be standardized
# Use StandardScaler to scale, fit and transform the data- leaving out the total_amount
_spent
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
```

In [ ]:

```
# Import train_test_split and prepare the train and test data
# Choose a test_size of 0.25 -leave out 25% of the data in the training
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_stat
e = 25)
```

In [ ]:

```
# Import the packages for the tests we will use
# 4 different tests: LinearRegression, ElasticNet, RandomForest, and XGBRegressor
# By having multiple different tests, we will be able to compare and contrast their sco
res
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
```

In [ ]:

```
# Define the models using conventional notation for reproducibility
model_ols = LinearRegression()
model_el = ElasticNet()
model_rf = RandomForestRegressor(n_estimators=25)
model_xgb = XGBRegressor()
```

In [ ]:

```
# Fit the models first
model_ols.fit(X_train, y_train)
model_el.fit(X_train, y_train)
model_rf.fit(X_train, y_train)
model_xgb.fit(X_train, y_train)
```

```
[07:19:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:li
near is now deprecated in favor of reg:squarederror.
```

Out[ ]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1
00,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=
0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

In [ ]:

```
# Now check out the scores on the test data
print("OLS Model test score is" + ' ' + str(model_ols.score(X_test, y_test)))
print("EL Model test score is" + ' ' + str(model_el.score(X_test, y_test)))
print("RF Model test score is" + ' ' + str(model_rf.score(X_test, y_test)))
print("XGB Model test score is" + ' ' + str(model_xgb.score(X_test, y_test)))
```

```
OLS Model test score is 0.7212985797174969
EL Model test score is 0.6425560061758354
RF Model test score is 0.7557992905743298
XGB Model test score is 0.7639659645118867
```

In [ ]:

```
# Check out the scores on the train data
print("OLS Model train score is" + ' ' + str(model_ols.score(X_train, y_train)))
print("EL Model train score is" + ' ' + str(model_el.score(X_train, y_train)))
print("RF Model train score is" + ' ' + str(model_rf.score(X_train, y_train)))
print("XGB Model train score is" + ' ' + str(model_xgb.score(X_train, y_train)))
```

```
OLS Model train score is 0.7324877213623169
EL Model train score is 0.6717369043915375
RF Model train score is 0.9598465015361467
XGB Model train score is 0.8251421894648006
```

The scores represent the coefficient of determination R^2 of the prediction, meaning that the value of the errors (distances between the predicted value and the true value) is calculated and squared, to give a positive value between 0 and 1. (The positive and negative error values cancel each other out)

All models performed better on the training data compared to the performance on the test data.

In [ ]:

```
# Define y variables in order to be able to visualize how well the algorithm is doing a
t predicting the total amount spent
y_predicted_ols = model_ols.predict(X_test)
y_predicted_el = model_el.predict(X_test)
y_predicted_rf = model_rf.predict(X_test)
y_predicted_xgb = model_xgb.predict(X_test)
```

Visualise the performance of the 4 models by plotting the y_test and the y_predicted for each model.

Optimally, if the model was perfect (always predicting the real total_amount_spent), the datapoints would all be arranged in a straight line. The less scattered the data is, the better the model performed

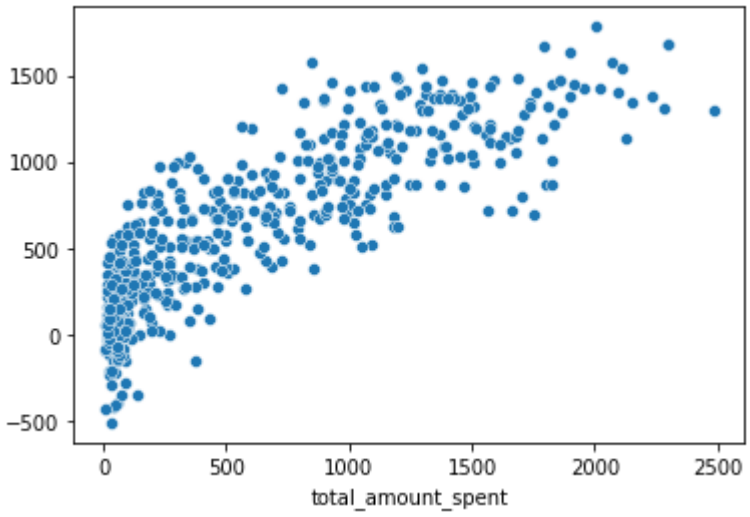
In [ ]:

```
# Visualisation of LinearRegression()
sns.scatterplot(y_test, y_predicted_ols)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a36b12390>
```



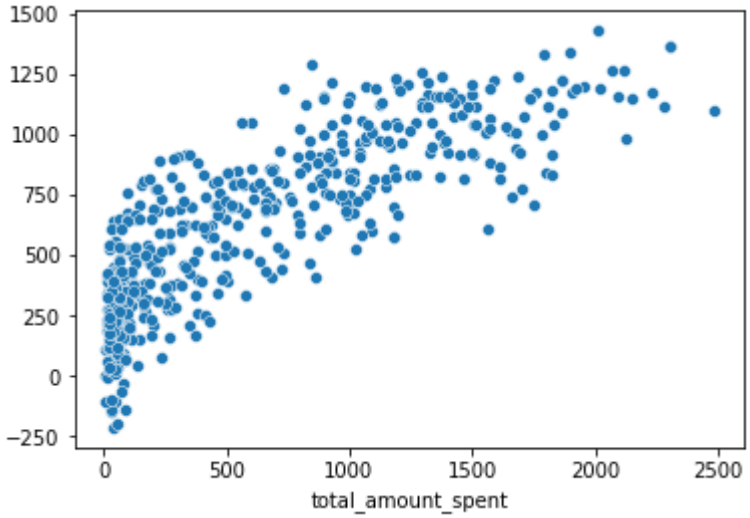
In [ ]:

```
# Visualisation of ElasticNet()
sns.scatterplot(y_test, y_predicted_el)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a37a8bc10>
```



In [ ]:

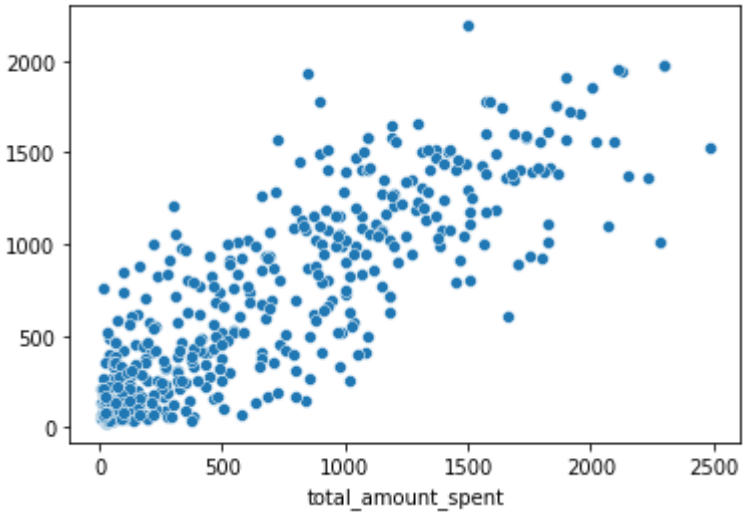
```
# Visualisation of RandomForestRegressor()

sns.scatterplot(y_test, y_predicted_rf)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a37fead50>
```



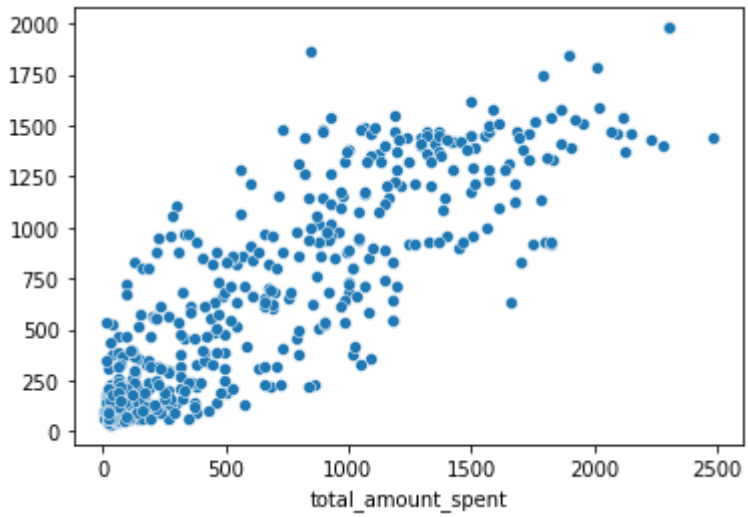
In [ ]:

```
# Visualisation of XGBRegressor()
sns.scatterplot(y_test, y_predicted_xgb)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpr
etation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a38014550>
```



Hyper parameter adjustement using GridSearchCV

In [ ]:

```
# Import the necessary functions
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

scorer = make_scorer(r2_score)
# scorer = make_scorer(mean_squared_error), The reason for this is explained after the
 tuning of all models
```

Hyperparameter optimizing for ElasticNet()

In [ ]:

```
params_el = {"alpha":[0.1, 0.5, 1.0], "l1_ratio":[0.1, 0.5, 0.75]}
```

In [ ]:

```
# Perform grid search on the classifier using 'scorer' as the scoring method.
grid_obj = GridSearchCV(model_el, params_el, scoring=scorer)
```

In [ ]:

```
# Use grid_obj.fit to perform the 5-fold crossvalidation to optimize the parameters
# The train data is split up 5 times, 4 parts of that data are used to shuffle through
 the different parameters
# The algorithm finds which parameters were the best between all the sets
grid_fit = grid_obj.fit(X, y)
```

In [ ]:

```
# Get the best estimator
best_reg = grid_fit.best_estimator_

# Fit the new model and check the parameters for the best result based on the scorer
best_reg.fit(X_train, y_train)
```

Out[ ]:

```
ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.75,
           max_iter=1000, normalize=False, positive=False, precompute=Fals
e,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=F
alse)
```

In [ ]:

```
best_reg.score(X_test, y_test)
```

Out[ ]:

```
0.7187213028285555
```

Hyperparameter optimizing for RandomForestRegressor()

In [ ]:

```
model_rf = RandomForestRegressor()
```

In [ ]:

```
params_rf = {"bootstrap": [True, False],
 "max_depth": [10, 20, None],
 "min_samples_split": [2, 5, 10],
 "n_estimators": [25, 50],
 "max_features": ["auto", "log2", "sqrt"],}
```

In [ ]:

```
# Perform grid search on the classifier using 'scorer' as the scoring method.
grid_obj = GridSearchCV(model_rf, params_rf, scoring=scorer)
```

In [ ]:

```
grid_fit = grid_obj.fit(X, y)
```

In [ ]:

```
# Get the estimator
best_reg = grid_fit.best_estimator_

# Fit the new model
best_reg.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=10, max_features='sqrt', max_leaf_nodes=No
ne,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=50, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [ ]:

```
best_reg.score(X_test, y_test)
```

Out[ ]:

```
0.7620244808596405
```

Hyperparameter optimizing for XGBRegressor()

In [ ]:

```
# Use the model xgb, add the argument "objective ='reg:squarederror'" in order to
# not get numerous warning error messages about a deprecated obj "reg:linear"
model_xgb = XGBRegressor(objective ='reg:squarederror')
```

In [ ]:

```
params_xgb = {"subsample": [0.6, 0.8, 1.0],
        "colsample_bytree": [0.6, 0.8, 1.0],
        "max_depth": [3, 4, 5]}
```

In [ ]:

```
grid_obj = GridSearchCV(model_xgb, params_xgb, scoring=scorer)
```

In [ ]:

```
grid_fit = grid_obj.fit(X, y)
```

In [ ]:

```
# Get the estimator.
best_reg = grid_fit.best_estimator_

# Fit the new model.
best_reg.fit(X_train, y_train)
```

Out[ ]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1.0, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1
00,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=
1,
             seed=None, silent=None, subsample=0.8, verbosity=1)
```

In [ ]:

```
best_reg.score(X_test, y_test)
```

Out[ ]:

```
0.7649697583025818
```

Summary of the results from the train-test split and the parameter hypertuning

	EL	RF	XGB
Before tuning	0.6624651748114022	0.7568844424375024	0.7554936163299296
mean_squared_error	0.5793639106955962	0.6059492929616079	0.7648156090706919
r2_score	0.7187213028285555	0.7607411117479351	0.7649697583025818

Some takeaways:

- EL has performed worse than RF and XGB at all stages, which can be attributed to the complexity of the algorithms.
- RF and XGB had very similar results (only a difference of 0.001 before tuning)
- While having mean_squared_error as the scorer for hypertuning with GridSeach, both EL and RF performed worse after the tuning, which was not considered optimal.
- As such, we used a different scorer, namely r2_score to check whether the tuning would work better, and it did. The code for the mean_squared_error was left in the analysis marked as a comment in order to be able to quickly check the results of the tests.
- XGB performed the best after tuning, with a score of 0.7649, with RF being a close second at 0.7607. However, the tests did not have an excellent performance, the score could have been better.
- This could be due to the data used for training the models, for example there were not enough features that contributed to the total amount spent. In the heatmap, it was seen that there are only 2 stronger correlations (one positive and one negative) with the total_amount_spent, Income and the nr of children at home. The models would have been more accurate at predicting the total amount spent if there were more features with a stronger correlation.
- Even though the data is not the most useful for predicting total spending, a value this high for XGB is relatively good compared to the lack of insight we could gain from UML.