# Project Name: Customer Segmentation based on Personality

## Goal: To split users/customers into segments based on behavior.

**Dataset**: Kaggle Customer Personality Analysis Dataset (https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis)

**About the Project**: In today's era, companies work hard to make their customers happy. They launch new technologies and services so that customers can use their products more. They try to be in touch with each of their customers so that they can provide goods accordingly. But practically, it's very difficult and non-realistic to keep in touch with everyone.
For this, the solution is to create segments of customers based on similar behavioral patterns, and use an algorhithm to keep in touch with all people of any particular segment. This is a much more feasible solution.

To facilitate this, we create a Machine Learning model, which when fed with appropriate data can create certain number of segments within it, hence making the desire to keep in touch with each user according to their needs possible.

### Section 1: Collecting the data

First off all, the necessary libraries need to be imported

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv("C:/Users/goura/Desktop/Data Science/Datasets/Customer Person
data
```

Out[2]:

| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_( |
|---|---|---|---|---|---|---|---|---|
| **0** | 5524 | 1957 | Graduation | Single | 58138.0 | 0 | 0 | 04 |
| **1** | 2174 | 1954 | Graduation | Single | 46344.0 | 1 | 1 | 08 |
| **2** | 4141 | 1965 | Graduation | Together | 71613.0 | 0 | 0 | 21 |
| **3** | 6182 | 1984 | Graduation | Together | 26646.0 | 1 | 0 | 1( |
| **4** | 5324 | 1981 | PhD | Married | 58293.0 | 1 | 0 | 1( |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **2235** | 10870 | 1967 | Graduation | Married | 61223.0 | 0 | 1 | 1: |
| **2236** | 4001 | 1946 | PhD | Together | 64014.0 | 2 | 1 | 1( |
| **2237** | 7270 | 1981 | Graduation | Divorced | 56981.0 | 0 | 0 | 25 |
| **2238** | 8235 | 1956 | Master | Together | 69245.0 | 0 | 1 | 24 |
| **2239** | 9405 | 1954 | PhD | Married | 52869.0 | 1 | 1 | 15 |

2240 rows × 29 columns

In [3]:

```
data.shape
```

Out[3]: (2240, 29)

In [4]:

```
data.describe()
```

Out[4]:

| | ID | Year_Birth | Income | Kidhome | Teenhome | Recency |
|---|---|---|---|---|---|---|
| **count** | 2240.000000 | 2240.000000 | 2216.000000 | 2240.000000 | 2240.000000 | 2240.00000( |
| **mean** | 5592.159821 | 1968.805804 | 52247.251354 | 0.444196 | 0.506250 | 49.109375 |
| **std** | 3246.662198 | 11.984069 | 25173.076661 | 0.538398 | 0.544538 | 28.962453 |
| **min** | 0.000000 | 1893.000000 | 1730.000000 | 0.000000 | 0.000000 | 0.00000( |
| **25%** | 2828.250000 | 1959.000000 | 35303.000000 | 0.000000 | 0.000000 | 24.00000( |
| **50%** | 5458.500000 | 1970.000000 | 51381.500000 | 0.000000 | 0.000000 | 49.00000( |
| **75%** | 8427.750000 | 1977.000000 | 68522.000000 | 1.000000 | 1.000000 | 74.00000( |
| **max** | 11191.000000 | 1996.000000 | 666666.000000 | 2.000000 | 2.000000 | 99.00000( |

8 rows × 26 columns

In [5]:

```
data.info()
```

```
<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
```

## Section 2: Data Cleaning / Manipulation

At the very first, we can remove the 'ID' column, because it doesn't have any actual influence or isnt a consequence of a person's behavior.

In [6]:
```
data=data.drop("ID", axis=1)
```

What's visible is that there's a column called "Dt_Customer" which encapsulates informtion about the date.
We can split up the date to form 3 columns "Year","Month",Day" for better performance.

In [7]:
```
parts = data["Dt_Customer"].str.split("-",n=3,expand=True)
data["day"] = parts[0].astype('int')
data["month"] = parts[1].astype('int')
data["year"] = parts[2].astype('int')
```

In [8]:
```
data = data.drop('Dt_Customer',axis=1)
```

Now, we do have a few columns which are "object" type, which means we need to encode such columns.

In [9]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

In [10]:
```
nums = []
objs = []
for i in data.columns:
    if data[i].dtype == object:
        objs.append(i)
    else:
        nums.append(i)
print(objs)
```

```
['Education', 'Marital_Status']
```

As well as encoding, we will have to keep track of which object gets encoded to which numeric value, for which we initialize a list.

In [11]:
```
codeval=[]
for i in objs:
    data[i] = encoder.fit_transform(data[i])
    codeval.append(dict(zip(encoder.classes_, encoder.transform(encoder.classes_
```

In [12]:
```
codeval
```

Out[12]:
```
[{'2n Cycle': 0, 'Basic': 1, 'Graduation': 2, 'Master': 3, 'PhD': 4},
 {'Absurd': 0,
  'Alone': 1,
  'Divorced': 2,
  'Married': 3,
  'Single': 4,
  'Together': 5,
  'Widow': 6,
  'YOLO': 7}]
```

In [13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 30 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Year_Birth           2240 non-null   int64  
 1   Education            2240 non-null   int32  
 2   Marital_Status       2240 non-null   int32  
 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  NumDealsPurchases    2240 non-null   int64  
 14  NumWebPurchases      2240 non-null   int64  
 15  NumCatalogPurchases  2240 non-null   int64  
 16  NumStorePurchases    2240 non-null   int64  
 17  NumWebVisitsMonth    2240 non-null   int64  
 18  AcceptedCmp3         2240 non-null   int64  
 19  AcceptedCmp4         2240 non-null   int64  
 20  AcceptedCmp5         2240 non-null   int64  
 21  AcceptedCmp1         2240 non-null   int64  
 22  AcceptedCmp2         2240 non-null   int64  
 23  Complain             2240 non-null   int64  
 24  Z_CostContact        2240 non-null   int64  
 25  Z_Revenue            2240 non-null   int64  
 26  Response             2240 non-null   int64  
 27  day                  2240 non-null   int32  
 28  month                2240 non-null   int32  
 29  year                 2240 non-null   int32  
dtypes: float64(1), int32(5), int64(24)
memory usage: 481.4 KB
```

Since we will need this information later, lets create dictionaries to keep note of which object was encoded to which numeric value for each of the columns.

In [14]:
```
data
```

Out[14]:

| | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Recency | M |
|---|---|---|---|---|---|---|---|---|
| **0** | 1957 | 2 | 4 | 58138.0 | 0 | 0 | 58 | |
| **1** | 1954 | 2 | 4 | 46344.0 | 1 | 1 | 38 | |
| **2** | 1965 | 2 | 5 | 71613.0 | 0 | 0 | 26 | |
| **3** | 1984 | 2 | 5 | 26646.0 | 1 | 0 | 26 | |
| **4** | 1981 | 4 | 3 | 58293.0 | 1 | 0 | 94 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **2235** | 1967 | 2 | 3 | 61223.0 | 0 | 1 | 46 | |
| **2236** | 1946 | 4 | 5 | 64014.0 | 2 | 1 | 56 | |
| **2237** | 1981 | 2 | 2 | 56981.0 | 0 | 0 | 91 | |
| **2238** | 1956 | 3 | 5 | 69245.0 | 0 | 1 | 8 | |
| **2239** | 1954 | 4 | 3 | 52869.0 | 1 | 1 | 40 | |

2240 rows × 30 columns

In [15]:

```
columns = [ i for i in data.columns ]
```

All of the essential changes to the data have been done.
We can further Standardize the data to make it easier for the model to learn from.

In [16]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data[columns] = scaler.fit_transform(data[columns])
data
```

Out[16]:

| | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Recency |
|---|---|---|---|---|---|---|---|
| **0** | -0.985345 | -0.350141 | 0.251004 | 0.234063 | -0.825218 | -0.929894 | 0.307039 |
| **1** | -1.235733 | -0.350141 | 0.251004 | -0.234559 | 1.032559 | 0.906934 | -0.383664 |
| **2** | -0.317643 | -0.350141 | 1.180340 | 0.769478 | -0.825218 | -0.929894 | -0.798086 |
| **3** | 1.268149 | -0.350141 | 1.180340 | -1.017239 | 1.032559 | -0.929894 | -0.798086 |
| **4** | 1.017761 | 1.428354 | -0.678332 | 0.240221 | 1.032559 | -0.929894 | 1.550305 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2235** | -0.150717 | -0.350141 | -0.678332 | 0.356642 | -0.825218 | 0.906934 | -0.107383 |
| **2236** | -1.903435 | 1.428354 | 1.180340 | 0.467539 | 2.890335 | 0.906934 | 0.237969 |
| **2237** | 1.017761 | -0.350141 | -1.607669 | 0.188091 | -0.825218 | -0.929894 | 1.446700 |
| **2238** | -1.068807 | 0.539106 | 1.180340 | 0.675388 | -0.825218 | 0.906934 | -1.419719 |
| **2239** | -1.235733 | 1.428354 | -0.678332 | 0.024705 | 1.032559 | 0.906934 | -0.314594 |

2240 rows × 30 columns

Lets have a look at the correlation matrix of this dataset

In [17]:

```
corr_matrix = data.corr()
corr_matrix
```

Out[17]:

| | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenl |
|---|---|---|---|---|---|---|
| **Year_Birth** | 1.000000 | -0.171390 | -0.060580 | -0.161791 | 0.230176 | -0.3 |
| **Education** | -0.171390 | 1.000000 | 0.007090 | 0.120692 | -0.045564 | 0.1 |
| **Marital_Status** | -0.060580 | 0.007090 | 1.000000 | 0.021353 | -0.022553 | -0.0 |
| **Income** | -0.161791 | 0.120692 | 0.021353 | 1.000000 | -0.428669 | 0.0 |
| **Kidhome** | 0.230176 | -0.045564 | -0.022553 | -0.428669 | 1.000000 | -0.0 |
| **Teenhome** | -0.352111 | 0.118485 | -0.003596 | 0.019133 | -0.036133 | 1.0 |
| **Recency** | -0.019871 | -0.011728 | 0.014159 | -0.003970 | 0.008827 | 0.0 |
| **MntWines** | -0.157773 | 0.197576 | 0.008205 | 0.578650 | -0.496297 | 0.0 |
| **MntFruits** | -0.017917 | -0.080412 | 0.000593 | 0.430842 | -0.372581 | -0.1 |
| **MntMeatProducts** | -0.030872 | 0.033625 | 0.030689 | 0.584633 | -0.437129 | -0.2 |
| **MntFishProducts** | -0.041625 | -0.112223 | 0.035808 | 0.438871 | -0.387644 | -0.2 |
| **MntSweetProducts** | -0.018133 | -0.105217 | 0.017382 | 0.440744 | -0.370673 | -0.1 |
| **MntGoldProds** | -0.061818 | -0.095489 | 0.001688 | 0.325916 | -0.349595 | -0.0 |
| **NumDealsPurchases** | -0.060846 | 0.030075 | -0.021772 | -0.083101 | 0.221798 | 0.3 |
| **NumWebPurchases** | -0.145040 | 0.081908 | -0.001894 | 0.387878 | -0.361647 | 0.1 |
| **NumCatalogPurchases** | -0.121275 | 0.070782 | 0.015125 | 0.589162 | -0.502237 | -0.1 |
| **NumStorePurchases** | -0.128272 | 0.070483 | 0.001412 | 0.529362 | -0.499683 | 0.0 |
| **NumWebVisitsMonth** | 0.121139 | -0.040281 | -0.031210 | -0.553088 | 0.447846 | 0.1 |
| **AcceptedCmp3** | 0.061774 | 0.005836 | -0.027113 | -0.016174 | 0.014674 | -0.0 |
| **AcceptedCmp4** | -0.060510 | 0.053266 | 0.014381 | 0.184400 | -0.161600 | 0.0 |
| **AcceptedCmp5** | 0.007123 | 0.033346 | 0.012817 | 0.335943 | -0.205634 | -0.1 |
| **AcceptedCmp1** | -0.005930 | -0.010845 | -0.017097 | 0.276820 | -0.172339 | -0.1 |
| **AcceptedCmp2** | -0.006539 | 0.021369 | 0.018417 | 0.087545 | -0.081716 | -0.0 |
| **Complain** | -0.030128 | -0.050540 | -0.005718 | -0.027225 | 0.040207 | 0.0 |
| **Z_CostContact** | NaN | NaN | NaN | NaN | NaN | |
| **Z_Revenue** | NaN | NaN | NaN | NaN | NaN | |
| **Response** | 0.021325 | 0.090819 | -0.011403 | 0.133047 | -0.080008 | -0.1 |
| **day** | -0.009193 | 0.018291 | -0.016087 | -0.031244 | -0.001718 | 0.0 |
| **month** | 0.024246 | -0.011304 | 0.017708 | -0.014955 | -0.023571 | -0.0 |
| **year** | -0.028188 | 0.045356 | -0.018176 | 0.022451 | 0.053339 | -0.0 |

30 rows × 30 columns

We see something strange, there are Null correlation values for all columns with Z_CostContact & Z_Revenue
Looking back at the dataset, we can notice that all values for both the columns are 0, post standardization.

In [18]:
```
print(data['Z_CostContact'].unique())
print(data['Z_Revenue'].unique())
```

```
[0.]
[0.]
```

Since either of those are just constant values, they play no significant role in our data, thus it is safe & wise to drop them.

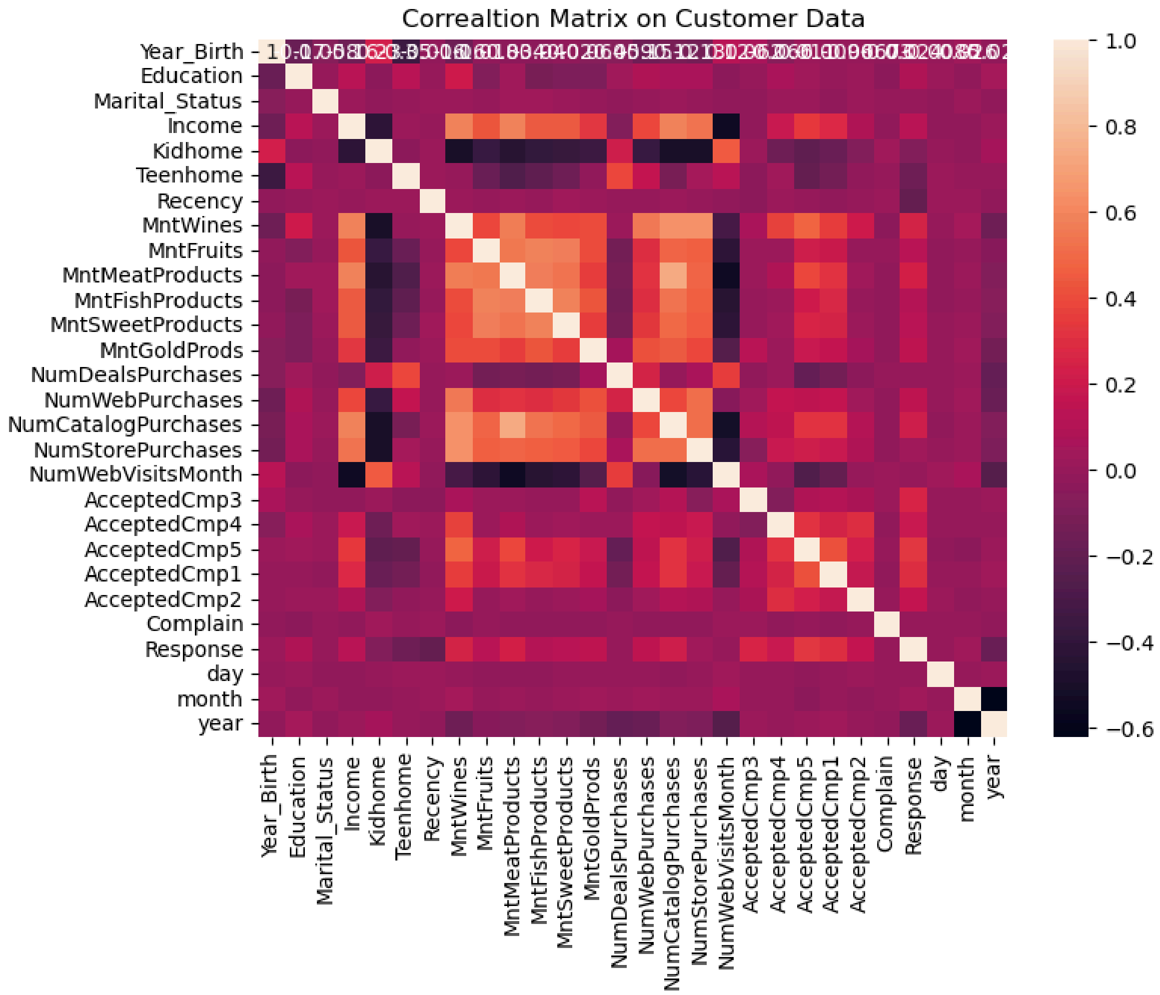
In [19]:
```
data = data.drop("Z_CostContact",axis=1)
data = data.drop("Z_Revenue",axis=1)
```

Our next step would be to drop all the Null values.

In [20]:
```
data = data.dropna(axis=0)
```

Since our data has now been finalized, lets try to visualize this using a correlation matrix.

In [21]:
```
corr_matrix = data.corr()
plt.figure(figsize=(8,6))
plt.title("Correaltion Matrix on Customer Data")
sns.heatmap(corr_matrix,annot=True)
plt.show()
```




Lets move on to the creation of the required Machine Learning Model.

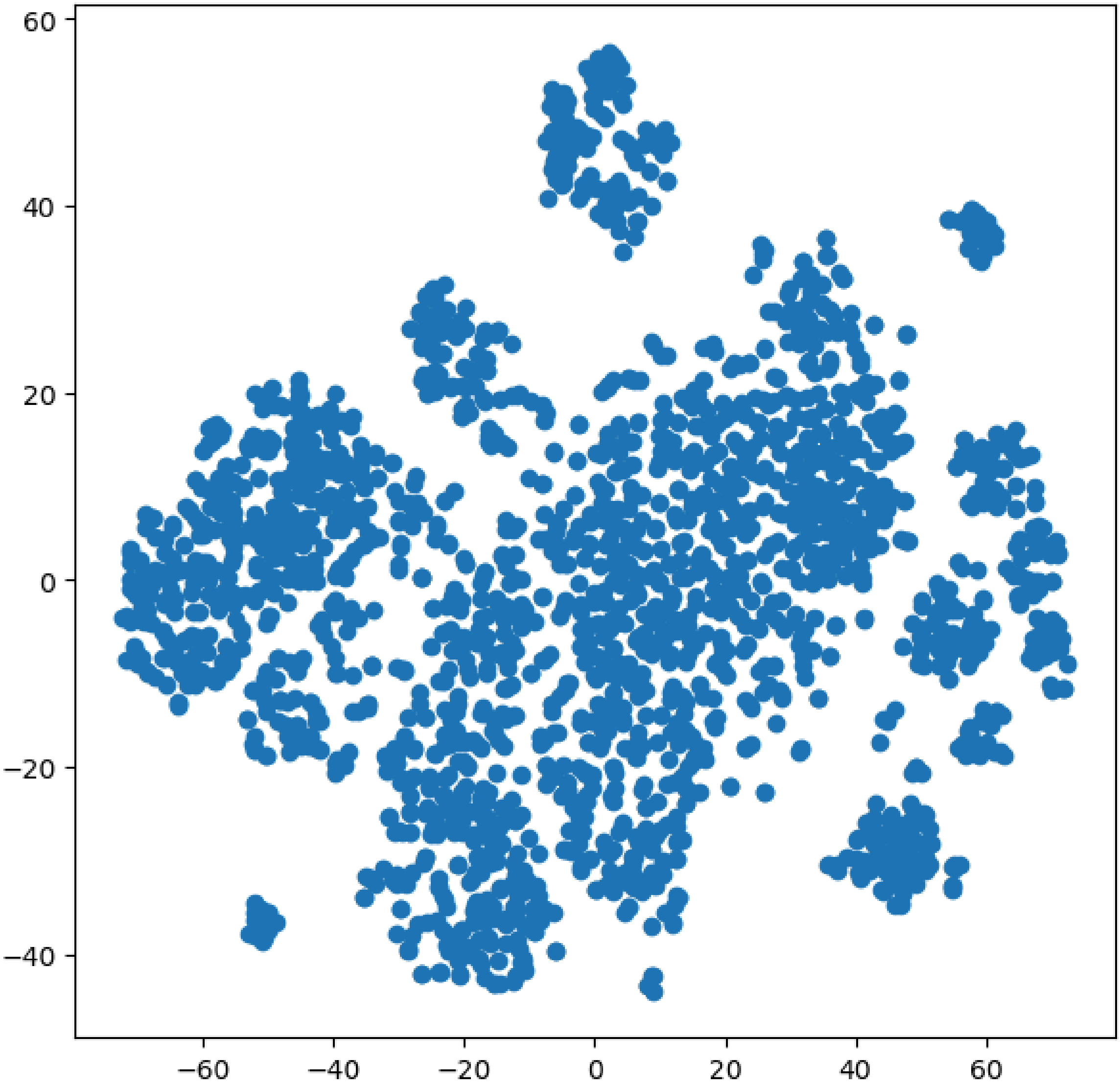
## Section 3: Model Selection

We will be using T-distributed Stochastic Neighbor Embedding. It helps in visualizing high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the values to low-dimensional embedding.
Note: Maximum permitted dimetion: 3

In [22]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components = 2, random_state=30)
tsne_data = model.fit_transform(data)
tsne_data
```

Out[22]:
```
array([[ 30.011686 ,  28.94765  ],
       [-10.480262 , -31.956112 ],
       [ 30.15759  ,   1.3275617],
       ...,
       [ 51.3011   , -26.50215  ],
       [ 19.630033 ,  -7.863148 ],
       [-18.908077 ,  22.416216 ]], dtype=float32)
```

In [23]:
```
plt.figure(figsize=(7, 7))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1])
plt.show()
```




From this diagram, some clusters are already visible.
But, there's a issue we will face latter on. Due to very the points being very evenly scattered, we may need a large no of clusters to reach optimal state.
This simply means the customers are a diverse set of people, which is often seen when countering a large userbase/mainstream app.

We will be using the KMeans model to form the clusters.

We plot down the error of the clustering for a large range of "number of clusters" (0-100)
Visualizing the graph can help us with finding the optimal number of clusters.
How? The number from which onwards the error starts becoming a straight line, is the optimal point.

Two possible scenarios-

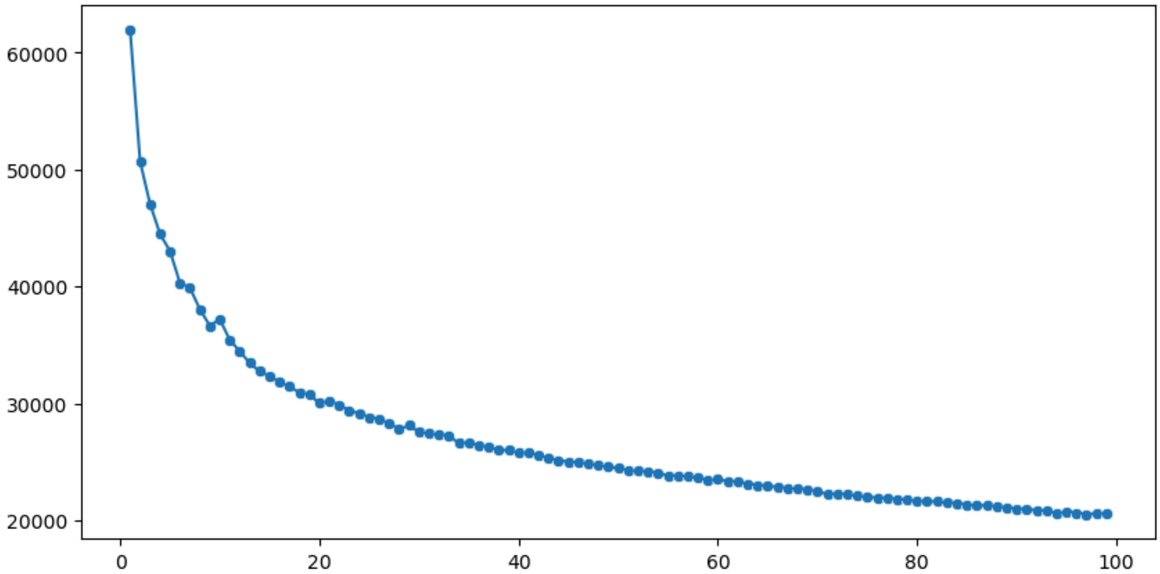
- The straight line is nearly flat/parallel to the X-axis
- The straight line still has a significant negative slope.

The first scenario is favourable, but in the second scenario, the ony options to flatten the line error-number line is to increase the number of clusters. Generally, it'd be a very large number compared to what we defined as the "optimal" number. So, unless the company has the resources to manage a large number of clusters and is willing to, we settle with the defined optimal number.

In [24]:
```
from sklearn.cluster import KMeans
error = []
for n_clusters in range(1, 100):
    model = KMeans(init='k-means++',
                   n_clusters=n_clusters,
                   max_iter=500,
                   random_state=22)
    model.fit(data)
    error.append(model.inertia_)
```

Note: In model.intertia_, .inertia is simply the error between all points of a cluster, summed up for all clusters.

In [25]:
```
plt.figure(figsize=(10, 5))
sns.lineplot(x=range(1, 100), y=error)
sns.scatterplot(x=range(1, 100), y=error)
plt.show()
```




The error decrease reaches constant slope at around 15-20 clusters, even if the slope is still a significant one.
We will continue with 20 to be our optimal number of clusters, because affording more clusters places a financial burden on the company in question.

In [26]:
```
model = KMeans ( init = 'k-means++',
                n_clusters = 20,
                max_iter = 100,
                random_state = 42)
segments = model.fit_predict(data)
```

In [27]:
```
segments
```

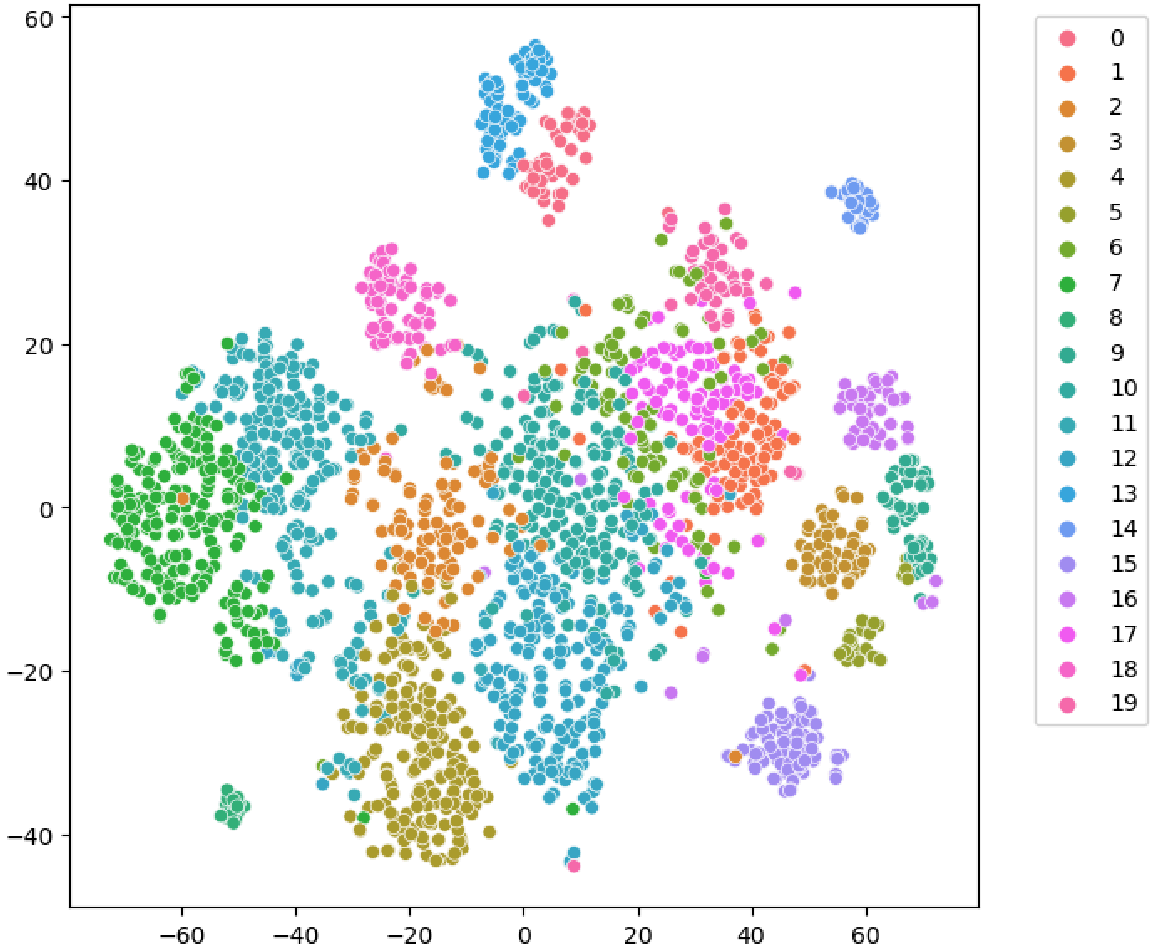
Out[27]:
```
array([19,  4,  1, ..., 15, 10, 18])
```

Segments now determines each instance/row/data point's cluster.

Lets plot our findings for a better understanding.

In [28]:
```
palette = sns.color_palette("husl", len(np.unique(segments)))
```

In [29]:
```
plt.figure(figsize = (7,7))
sns.scatterplot(x=tsne_data[:, 0], y=tsne_data[:, 1], hue=segments, palette = pa
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```




This above plot represents our clusters. While there is a rough boundary between all clusters, its not very clear to the eye due to frequent errors in classification, caused by the diversity.

Now, we shall proceed to make our program user-interactive, such that a person can be classified based on their behavioral patterns, provided as input.

## Section 4: User-Interactive Space

In [30]:
```
data.columns
```

Out[30]:
```
Index(['Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Complain', 'Response', 'day', 'month', 'year'],
      dtype='object')
```

In [32]:
```
ques = [ i for i in data.columns ]
c=0
while True:
    if c==0:
        print("Do you want to make any predictions?")
    elif c>0:
        print("Do you want to make any more predictions?")
    print("Enter 1 if Yes, else 0")
    a = int(input("Choice:"))
    if a!=0 and a!=1:
        print("Invalid Choice")
    elif a==0:
        print("Choice is No (0)")
        print("Exit Program")
        if c>0:
            print("Thank you for using our services.")
        break
    elif a==1:
        print("Choice is Yes (1)")
        lst = []
        for i in ques:
            if i not in ["Marital_Status","Education"]:
                lst.append(float(input(f"Enter value of {i}: ")))
            else:
                a=input(f"Enter value of {i}: ")
                a =codeval[objs.index(i)][a]
                lst.append(a)
        df = pd.DataFrame([lst], columns=ques)
        df[ques] = scaler.fit_transform(df[ques])
        preds = model.predict(df)
        print("Reported Parameters: ",lst)
        print("The customer is determined to be of cluster",preds[0])
        c+=1
```

```
Do you want to make any predictions?
Enter 1 if Yes, else 0
Choice is Yes (1)
Reported Parameters:  [2005.0, 2, 4, 50000.0, 0.0, 0.0, 3.0, 4.0, 2.0, 5.0, 7.0,
2.0, 5.0, 20.0, 32.0, 34.0, 55.0, 11.0, 3.0, 5.0, 52.0, 6.0, 7.0, 3.0, 4.0, 5.0,
6.0, 2022.0]
The customer is determined to be of cluster 12
Do you want to make any more predictions?
Enter 1 if Yes, else 0
Choice is No (0)
Exit Program
Thank you for using our services.
```

Since the model has been defined, and set to actively accept provided data and provide predicted classifications,

our program is ready to use and the project is complete.

## Conclusion

This marks the end of our project.

This program assists us in classifying the customers of the store/app based on their behavioral patterns.
We acheived this via creating a KMeans Model.

We noticed that the error-n_clusters line had a significant downwards slope after straightening, for the provided data, thus we can conclude the dataset to be very diverse and thus tough to classify. Despite that, our model has perfomed fairly well and provided us with visually-distinctive clusters.
Therefore, the Program is ready to be used.