CUSTOMER SEGMENTATION USING DATA SCIENCE

Introduction:

Customer segmentation using data science relies on the availability and quality of data. Before any advanced analysis can take place, businesses must effectively load and preprocess their datasets. This process is essential for ensuring the data's cleanliness, consistency, and readiness for modeling

Given Dataset:

	A	B	C	D	E	F
1	CustomerID	Genre	Age	Annual Income (	Spending Score (1-100)	
2	1	Male	19	15	39	
3	2	Male	21	15	81	
4	3	Female	20	16	6	
5	4	Female	23	16	77	
6	5	Female	31	17	40	
7	6	Female	22	17	76	
8	7	Female	35	18	6	
9	8	Female	23	18	94	
10	9	Male	64	19	3	
11	10	Female	30	19	72	
12	11	Male	67	19	14	
13	12	Female	35	19	99	
14	13	Female	58	20	15	
15	14	Female	24	20	77	
16	15	Male	37	20	13	
17	16	Male	22	20	79	
18	17	Female	35	21	35	
19	18	Male	20	21	66	

Here is a general guideline on how to preprocess your dataset for customer segmentation:

1. **Import Libraries:** Begin by importing essential Python libraries for data analysis such as pandas, numpy, and matplotlib or seaborn for visualization.

Python code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

2. **Load the Data:** Load your dataset into a pandas DataFrame.

Python code :

```
data = pd.read_csv('your_dataset.csv')
```

3. **Exploratory Data Analysis (EDA):** Perform initial EDA to understand the data, its structure, and any potential issues.

python code:

```
# Display the first few rows of the dataset

print(data.head())

# Check the dimensions of the dataset

print(data.shape)

# Check for missing values

print(data.isnull().sum())

# Get statistical summary of the dataset

print(data.describe
```

4. **Data Cleaning:** Handle missing values: Depending on the nature and quantity of missing data, you can choose to drop the rows, fill in the missing values with mean or median, or use advanced imputation techniques.

5. **Remove duplicates:** Check for and remove any duplicate entries in the dataset.

Python code:

```
# Drop rows with missing values

data = data.dropna()

# Remove duplicates

data = data.drop_duplicates()
```

6. **Data Transformation:**
    - Convert categorical variables to numerical ones using encoding techniques such as one-hot encoding or label encoding.
    - Scale numerical features if they are on different scales using techniques such as Min-Max scaling or Standard scaling.

Python code:

```
# Perform one-hot encoding for categorical variables

data = pd.get_dummies(data, columns=['categorical_column'])

# Scale numerical features

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data[['numerical_column1', 'numerical_column2']] =
scaler.fit_transform(data[['numerical_column1', 'numerical_column2']])
```

7. **Feature Selection:** Select relevant features that are important for customer segmentation. This can be based on domain knowledge or using feature selection techniques like correlation analysis or using feature importance from machine learning models.

8. **Dimensionality Reduction:** If your dataset has high dimensionality, consider using techniques like Principal Component Analysis (PCA) to reduce the number of features while retaining the most important information.

Python code:

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2)

data_pca = pca.fit_transform(data)
```

9. **Data Normalization and Standardization:** Ensure that the data is in a suitable format for your chosen machine learning algorithms.

10. **Save Preprocessed Data:** Save the preprocessed data for use in your customer segmentation analysis.

Python code:

```
data.to_csv('preprocessed_data.csv', index=False)
```

Importance of loading and preprocessing dataset :

Loading and preprocessing the dataset are essential steps in customer segmentation using data science. They play a critical role in ensuring the quality and reliability of the subsequent analysis. Here are the key reasons why these steps are crucial:

Data Integrity and Quality: Loading the dataset allows you to check its integrity and assess the quality of the data. By preprocessing the data, you can clean it by handling missing values, removing duplicates, and dealing with outliers. Ensuring data integrity is crucial for obtaining accurate insights from the analysis.

Data Understanding: Loading the dataset enables you to understand the structure and format of the data. This understanding is crucial for choosing the appropriate preprocessing techniques. It allows you to identify the types of variables, such as categorical or numerical, and plan the preprocessing steps accordingly.

1. **Enhanced Analysis Efficiency:** Proper preprocessing enhances the efficiency of the subsequent analysis. It helps in reducing the computational complexity, making the algorithms run faster and enabling the exploration of more sophisticated and complex models without compromising performance.

2. **Improved Model Performance and Interpretability:** A well-preprocessed dataset leads to improved model performance and interpretability. It ensures that the segmentation model can effectively identify meaningful patterns and segments within the customer data, providing valuable insights for business decisions and marketing strategies.

Loading the dataset:

Here are the key steps to load a dataset for customer segmentation:

1. Data Collection:

- Identify the sources of customer data, which could include databases, CRM systems, spreadsheets, or external APIs.
- Ensure that you have the necessary permissions and access to retrieve the data.

2. Data Extraction:

- Extract the data from the chosen sources. This may involve running SQL queries, using data connectors, or downloading files.

3. Data Storage:

- Choose a suitable storage format for your data, such as a CSV file, a relational database, or a data warehouse.
-
- Consider using data management tools like Pandas (Python) or libraries like SQLalchemy for efficient data storage.

4. Data Cleaning:

- Examine the dataset for missing values, duplicate records, and inconsistencies.
- Perform data cleaning tasks like imputing missing values or removing outliers.

5. Data Exploration:

- Explore the loaded dataset to gain an initial understanding of its structure and contents. Use summary statistics and data visualization to identify patterns.

6. Data Preprocessing:

- Transform and prepare the data for analysis. This may include encoding categorical variables, scaling numerical features, and feature engineering.

7. Data Sampling:

- Depending on the dataset's size, you may choose to take a random sample for initial analysis to save processing time.

8. Data Splitting:

- Divide the dataset into training and testing sets if you plan to build and evaluate segmentation models.

9. Data Version Control:

- If working in a collaborative or version-controlled environment, ensure that you track changes to the dataset to maintain data integrity.

10. Data Security:

- Take measures to protect the privacy and security of customer data, particularly if it contains sensitive information.

11.Documentation:

- Maintain clear documentation about the dataset, including its source, structure, and any preprocessing steps applied. This is essential for reproducibility and collaboration.

12.Data Integration:

- If your analysis requires data from multiple sources, integrate and join datasets as needed.

Preprocessing the dataset:

Preprocessing the dataset is a crucial step in customer segmentation using data science. This phase involves cleaning and transforming the data to make it suitable for analysis. Here are the key steps for preprocessing a dataset in customer segmentation:

1. **Data Cleaning:**
   - Handle missing values: Decide whether to impute missing data or remove records with missing values based on the impact on the analysis.
   - Detect and handle duplicates: Identify and remove duplicate records to prevent bias in segmentation results.

2. Data Transformation:

- Encoding Categorical Variables: Convert categorical features into a numerical format using techniques like one-hot encoding or label encoding.

- Scaling Numerical Features: Normalize numerical features to ensure that all features have the same scale. Common methods include Min-Max scaling or standardization (mean and standard deviation scaling).

3. Feature Engineering:

- Create new features: Generate meaningful new features that may better represent customer characteristics.

- Reduce dimensionality: Use dimensionality reduction techniques, such as Principal Component Analysis (PCA), if dealing with high-dimensional data.

4. Handling Outliers:

- Identify and manage outliers, as they can skew segmentation results. You can use statistical methods or visualization techniques to detect outliers and decide whether to remove or transform them.

5. Data Sampling:

- If your dataset is large, consider taking a random sample for faster model development and testing. Ensure that the sample is representative of the overall dataset.

6. Data Splitting:

- Divide the dataset into training and testing sets if you plan to build and evaluate segmentation models. The training set is used for model development, and the testing set is for model evaluation.

7. Data Standardization:

- Standardize the data if you plan to use clustering techniques that are sensitive to feature scales. Standardization ensures that features have a mean of 0 and a standard deviation of 1.

8. Handling Imbalanced Data:

- If your dataset has imbalanced segments, consider oversampling or undersampling techniques to balance the representation of segments.

9. Data Imputation:

- Impute missing data using methods like mean imputation, median imputation, or more advanced imputation techniques.

10. Data Validation:

- Validate the quality of the preprocessed data to ensure that it aligns with the analysis goals and does not introduce bias or errors.

11. Data Version Control:

- Keep track of changes made to the dataset, especially in collaborative or version-controlled environments.

12. Documentation:

- Maintain documentation that outlines the preprocessing steps, transformation techniques, and reasons for decisions made during the process. This is essential for reproducibility.

Effective preprocessing of the dataset is critical for accurate and meaningful customer segmentation using data science. It ensures that the data is in a form that can be readily used for clustering or classification algorithms to identify distinct customer segments.

Import Libraries:

The libraries we will be required are :

- Pandas
- Numpy
- Matplotlib / Seaborn
- Sklearn

Here, how to load a dataset using Python

Program:

```
import numpy as np

import pandas as pd
```

```
import matplotlib.pyplot as plt

import seaborn as sb


from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.cluster import KMeans


import warnings

warnings.filterwarnings('ignore')
```

Importing Dataset:

The dataset taken for the task includes the details of customers includes their marital status, their income, number of items purchased, types of items purchased, and so on.

Program:

```
df = pd.read_csv('new.csv')

df.head()
```

Output:

```
     ID  Year_Birth   Education Marital_Status   Income  Kidhome  Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0         0
1  2174        1954  Graduation         Single  46344.0        1         1
2  4141        1965  Graduation       Together  71613.0        0         0
3  6182        1984  Graduation       Together  26646.0        1         0
4  5324        1981         PhD        Married  58293.0        1         0

  Dt_Customer  Recency  MntWines  ...  NumDealsPurchases  NumWebPurchases  \
0  04-09-2012       58       635  ...                  3                8
1  08-03-2014       38        11  ...                  2                1
2  21-08-2013       26       426  ...                  1                8
3  10-02-2014       26        11  ...                  2                2
4  19-01-2014       94       173  ...                  5                5

   NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth  Complain  \
0                   10                  4                  7         0
1                    1                  2                  5         0
2                    2                 10                  4         0
3                    0                  4                  6         0
4                    3                  6                  5         0

   Z_CostContact  Z_Revenue  Response      Accepted
0              3         11         1  AcceptedCmp1
1              3         11         0  AcceptedCmp1
2              3         11         0  AcceptedCmp1
3              3         11         0  AcceptedCmp1
4              3         11         0  AcceptedCmp1

[5 rows x 25 columns]
```

To check the shape of the dataset we can use data.shape method.

Program:

df.shape

Output:

```
(2240, 25)(2240, 25)
```

To get the information of the dataset like checking the null values, count of values, etc. we will use .info() method.

Data Processing

Program:

df.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 25 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  Complain             2240 non-null   int64  
 21  Z_CostContact        2240 non-null   int64  
 22  Z_Revenue            2240 non-null   int64  
 23  Response             2240 non-null   int64  
 24  Accepted             2240 non-null   object 
dtypes: float64(1), int64(20), object(4)
```

Program:

df.describe().T

Output:

	count	mean	std	min	25%	50%	75%	max
ID	2240.0	5592.159821	3246.662198	0.0	2828.25	5458.5	8427.75	11191.0
Year_Birth	2240.0	1968.805804	11.984069	1893.0	1959.00	1970.0	1977.00	1996.0
Income	2216.0	52247.251354	25173.076661	1730.0	35303.00	51381.5	68522.00	666666.0
Kidhome	2240.0	0.444196	0.538398	0.0	0.00	0.0	1.00	2.0
Teenhome	2240.0	0.506250	0.544538	0.0	0.00	0.0	1.00	2.0
Recency	2240.0	49.109375	28.962453	0.0	24.00	49.0	74.00	99.0
MntWines	2240.0	303.935714	336.597393	0.0	23.75	173.5	504.25	1493.0
MntFruits	2240.0	26.302232	39.773434	0.0	1.00	8.0	33.00	199.0
MntMeatProducts	2240.0	166.950000	225.715373	0.0	16.00	67.0	232.00	1725.0
MntFishProducts	2240.0	37.525446	54.628979	0.0	3.00	12.0	50.00	259.0
MntSweetProducts	2240.0	27.062946	41.280498	0.0	1.00	8.0	33.00	263.0
MntGoldProds	2240.0	44.021875	52.167439	0.0	9.00	24.0	56.00	362.0
NumDealsPurchases	2240.0	2.325000	1.932238	0.0	1.00	2.0	3.00	15.0
NumWebPurchases	2240.0	4.084821	2.778714	0.0	2.00	4.0	6.00	27.0
NumCatalogPurchases	2240.0	2.662054	2.923101	0.0	0.00	2.0	4.00	28.0
NumStorePurchases	2240.0	5.790179	3.250958	0.0	3.00	5.0	8.00	13.0
NumWebVisitsMonth	2240.0	5.316518	2.426645	0.0	3.00	6.0	7.00	20.0
Complain	2240.0	0.009375	0.096391	0.0	0.00	0.0	0.00	1.0
Z_CostContact	2240.0	3.000000	0.000000	3.0	3.00	3.0	3.00	3.0
Z_Revenue	2240.0	11.000000	0.000000	11.0	11.00	11.0	11.00	11.0
Response	2240.0	0.149107	0.356274	0.0	0.00	0.0	0.00	1.0

Improving the values in the Accepted column.

Program:

df['Accepted'] = df['Accepted'].str.replace('Accepted', '')

To check the null values in the dataset

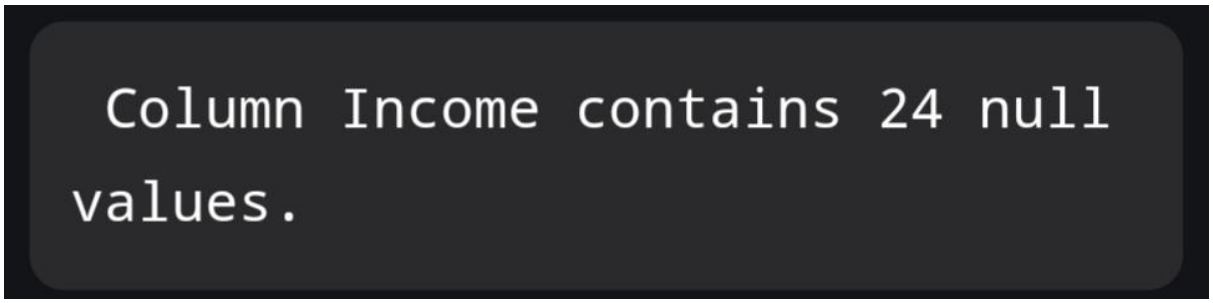
Program:

for col in df.columns:

    temp = df[col].isnull().sum()

```
    if temp > 0:
        print(f'Column {col} contains {temp} null values.')
```

Output:

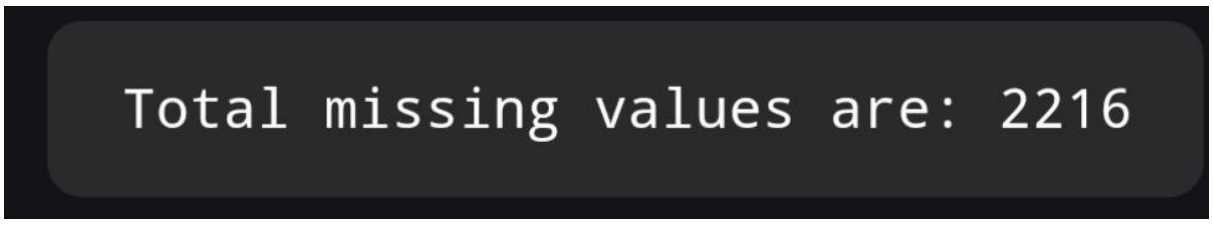


Now, once we have the count of the null values and we know the values are very less we can drop them (it will not affect the dataset much).

Program:

```
df = df.dropna()

print("Total missing values are:", len(df))
```

Output :



To find the total number of unique values in each column we can use data.unique() method.

Program:

```
df.nunique()
```

Output:

ID	2216
Year_Birth	59
Education	5
Marital_Status	8
Income	1974
Kidhome	3
Teenhome	3
Dt_Customer	662
Recency	100
MntWines	776
MntFruits	158
MntMeatProducts	554
MntFishProducts	182
MntSweetProducts	176
MntGoldProds	212
NumDealsPurchases	15
NumWebPurchases	15
NumCatalogPurchases	14
NumStorePurchases	14
NumWebVisitsMonth	16
Complain	2
Z_CostContact	1
Z_Revenue	1
Response	2
Accepted	5

Here we can observe that there are columns which contain single values in the whole column so, they have no relevance in the model development.

Also dataset has a column Dt_Customer which contains the date column, we can convert into 3 columns i.e. day, month, year.

Program:

```
parts = df["Dt_Customer"].str.split("-", n=3, expand=True)

df["day"] = parts[0].astype('int')

df["month"] = parts[1].astype('int')

df["year"] = parts[2].astype('int')
```

Now we have all the important features, we can now drop features like Z_CostContact, Z_Revenue, Dt_Customer.

Program:

```
df.drop(['Z_CostContact', 'Z_Revenue', 'Dt_Customer'],
                axis=1,
                inplace=True)
```

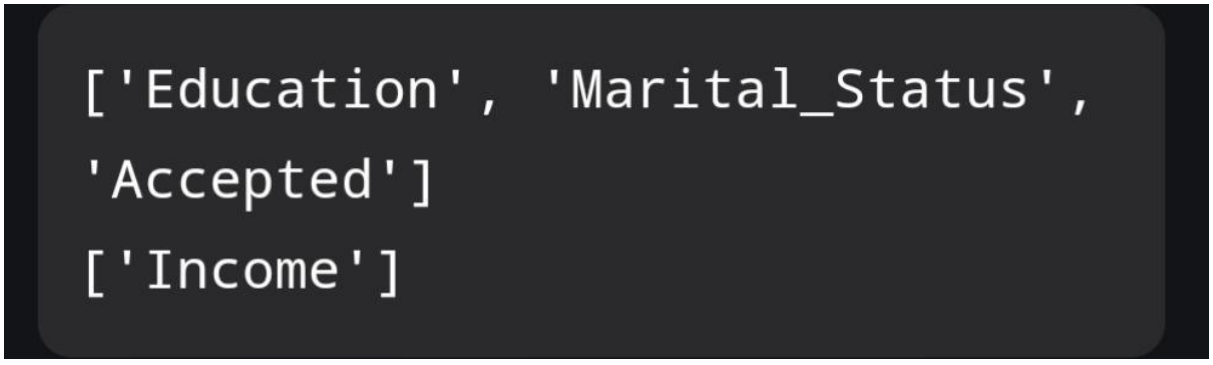
Data Visualization and Analysis

Data Visualization is the graphical representation of information and data in a pictorial or graphical format. Here we will be using bar plot and count plot for better visualization.

Program:

```
floats, objects = [], []
for col in df.columns:
    if df[col].dtype == object:
        objects.append(col)
    elif df[col].dtype == float:
        floats.append(col)

print(objects)
print(floats)
```

Output:

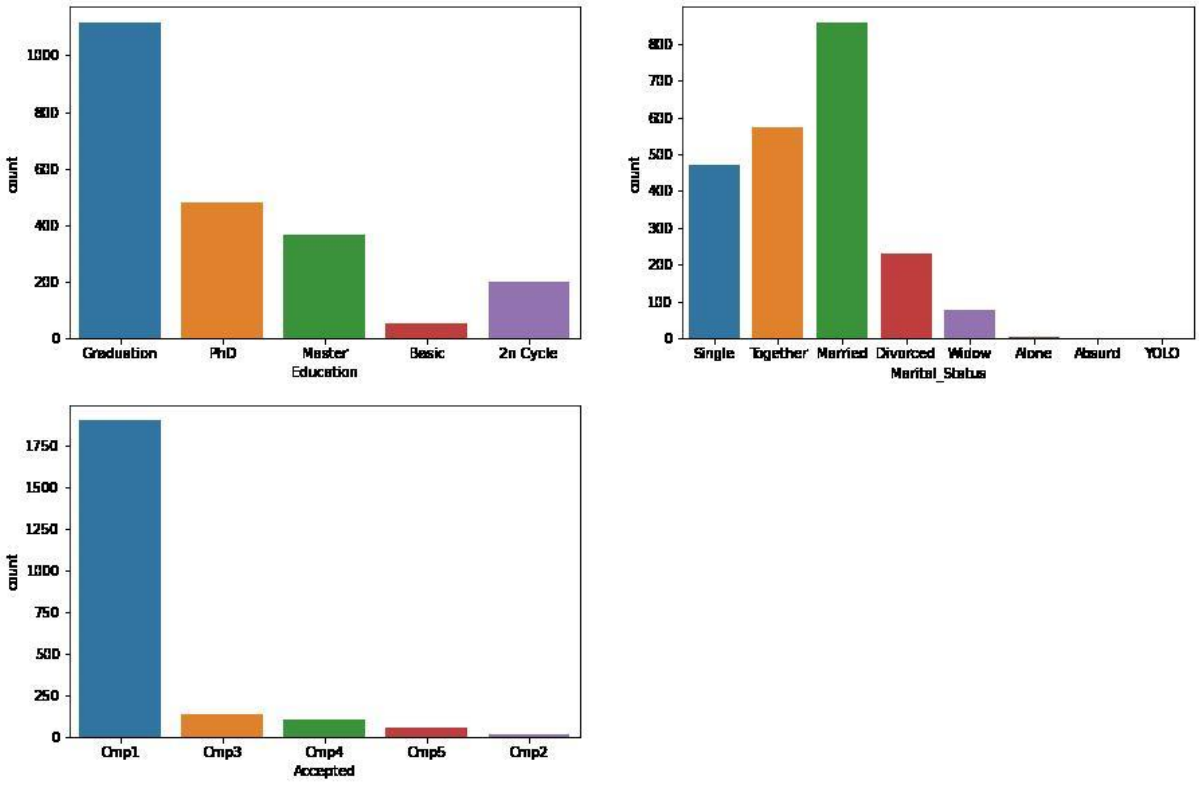


To get the count plot for the columns of the datatype – object, refer the code below.

Program:

```
plt.subplots(figsize=(15, 10))

for i, col in enumerate(objects):

        plt.subplot(2, 2, i + 1)

        sb.countplot(df[col])

plt.show()
```

Output:



Let’s check the value_counts of the Marital_Status of the data.

Prrogram:

```
df['Marital_Status'].value_counts()
```

Output:

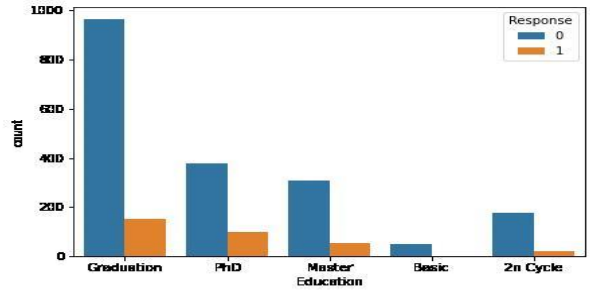
```
Married     857
Together    573
Single      471
Divorced    232
Widow        76
Alone         3
Absurd        2
YOLO          2
Name: Marital_Status, dtype: int64
```

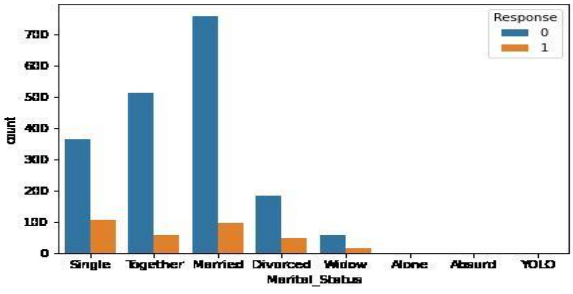
Now lets see the comparison of the features with respect to the values of the responses.

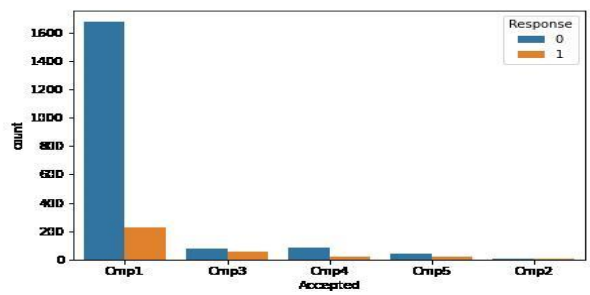
Program:

```
plt.subplots(figsize=(15, 10))

for i, col in enumerate(objects):

        plt.subplot(2, 2, i + 1)

        sb.countplot(df[col], hue=df['Response'])

plt.show()
```

Output:







Label Encoding

Label Encoding is used to convert the categorical values into the numerical values so that model can understand it.

Program:

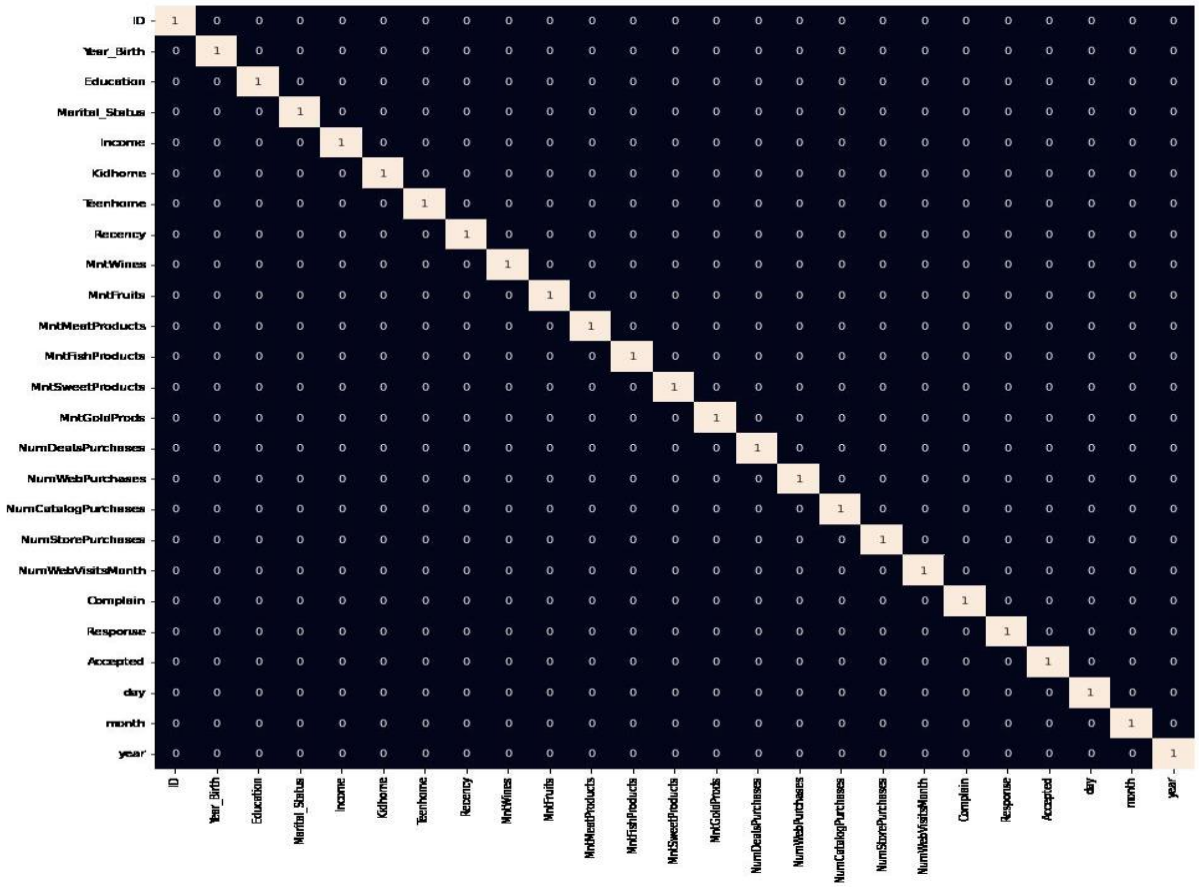
```
for col in df.columns:
    if df[col].dtype == object:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
```

Heat map is the best way to visualize the correlation among the different features of dataset. Let's give it the value of 0.8

Program :

```
plt.figure(figsize=(15, 15))
sb.heatmap(df.corr() > 0.8, annot=True, cbar=False)
plt.show()
```

Output:



Standardization

Standardization is the method of feature scaling which is an integral part of feature engineering. It scales down the data and making it easier for the machine learning model to learn from it. It reduces the mean to '0' and the standard deviation to '1'.

Program:

```
scaler = StandardScaler()

data = scaler.fit_transform(df)
```

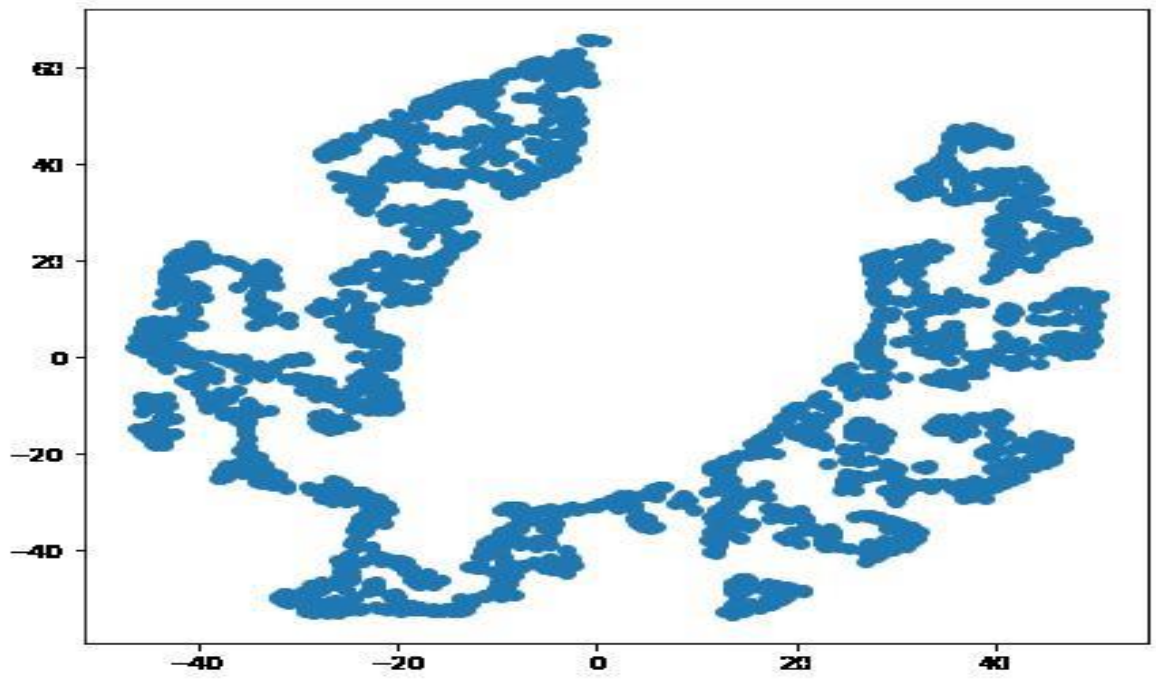
Segmentation

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.

Program:

```
from sklearn.manifold import TSNE

model = TSNE(n_components=2, random_state=0)

tsne_data = model.fit_transform(df)

plt.figure(figsize=(7, 7))

plt.scatter(tsne_data[:, 0], tsne_data[:, 1])

plt.show()
```

Output:



There are certainly some clusters which are clearly visual from the 2-D representation of the given data. Let's use the KMeans algorithm to find those clusters in the high dimensional plane itself

KMeans Clustering can also be used to cluster the different points in a plane.

Program:

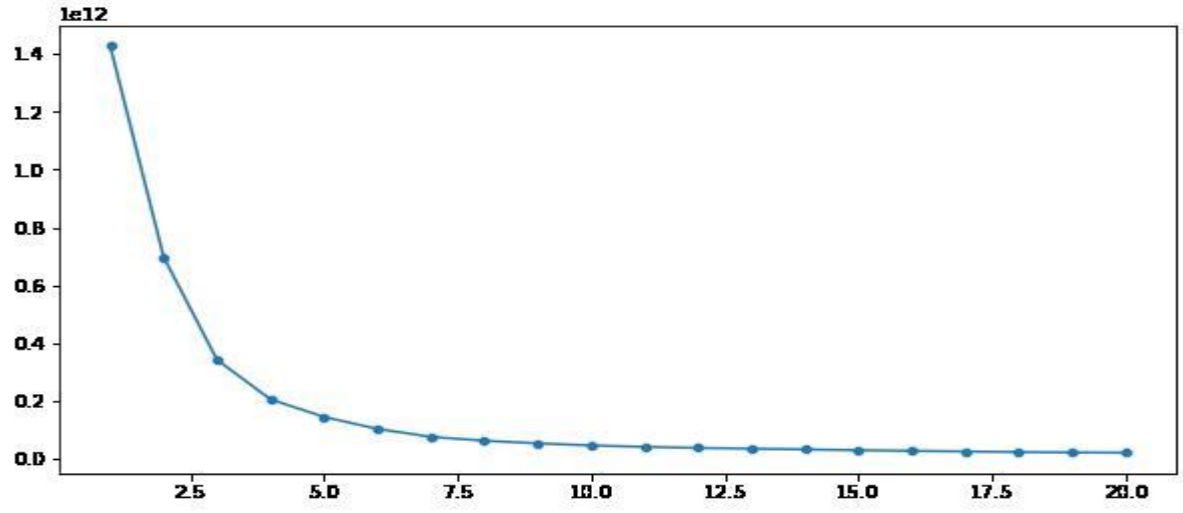
```
error = []

for n_clusters in range(1, 21):
        model = KMeans(init='k-means++',
                                n_clusters=n_clusters,
                                max_iter=500,
                                random_state=22)
        model.fit(df)
        error.append(model.inertia_)
```

Here inertia is nothing but the sum of squared distances within the cluster.

Program:

```
plt.figure(figsize=(10, 5))
sb.lineplot(x=range(1, 21), y=error)
sb.scatterplot(x=range(1, 21), y=error)
plt.show()
```

Output:



Here by using the elbow method we can say that k = 6 is the optimal number of clusters that should be made as after k = 6 the value of the inertia is not decreasing drastically.

Program:

```
# create clustering model with optimal k=5

model = KMeans(init='k-means++',

                    n_clusters=5,

                    max_iter=500,

                    random_state=22)

segments = model.fit_predict(df)
```

Scatterolot will be used to see all the 6 clusters formed by KMeans Clustering.

Program:

```
plt.figure(figsize=(7, 7))

sb.scatterplot(tsne_data[:, 0], tsne_data[:, 1], hue=segments)

plt.show()
```

Output: