Customer Segmentation Using Python

Overview

A superstore is planning for the year-end sale. They want to launch a new offer — gold membership, that gives a 20% discount on all purchases, for only 499*whichis*999 on other days. It will be valid only for existing customers and the campaign, through phone calls, is currently being planned for them. The management feels that the best way to reduce the cost of the campaign is to make a predictive model that will classify customers who might purchase the offer.

Success will include:

- Coming up with a model that correctly predicts the likelihood of a customer to give a positive response.
- Analyzing and establishing the factors that contribute towards a customer giving a positive response to the Superstore campaign.
- A model with an accuracy level of 80%.

1. Data Collection & Loading

In [2]:
```
# importing libraries
import pandas as pd
import numpy as np

# viz libraries
import matplotlib.pyplot as plt
import seaborn as sns

# randomForestClassifier for the model
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

In [3]:
```
# loading the dataset

store_data = pd.read_csv('superstore_data.csv')
```

2. Data Exploration

In [4]:
```
# previewing the data
store_data.head()
```

Out[4]:

	Id	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	MntFishProducts	MntS
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/2014	0	189	...	111	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/2014	0	464	...	7	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/2014	0	134	...	15	
3	1386	1967	Graduation	Together	32474.0	1	1	11/5/2014	0	10	...	0	
4	5371	1989	Graduation	Single	21474.0	1	0	8/4/2014	0	6	...	11	

5 rows × 22 columns

In [5]:
```
store_data.info()
# 18 columns, 2240 data points
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 22 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  Response             2240 non-null   int64  
 21  Complain             2240 non-null   int64  
dtypes: float64(1), int64(18), object(3)
memory usage: 385.1+ KB
```

In [6]:
```
store_data.describe(include='all')
# 5 categories in the Education column, 8 categories in the Marital_Status
```

Out[6]:

	Id	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWi
count	2240.000000	2240.000000	2240	2240	2216.000000	2240.000000	2240.000000	2240	2240.000000	2240.000
unique	NaN	NaN	5	8	NaN	NaN	NaN	663	NaN	N
top	NaN	NaN	Graduation	Married	NaN	NaN	NaN	8/31/2012	NaN	N
freq	NaN	NaN	1127	864	NaN	NaN	NaN	12	NaN	N
mean	5592.159821	1968.805804	NaN	NaN	52247.251354	0.444196	0.506250	NaN	49.109375	303.935
std	3246.662198	11.984069	NaN	NaN	25173.076661	0.538398	0.544538	NaN	28.962453	336.597
min	0.000000	1893.000000	NaN	NaN	1730.000000	0.000000	0.000000	NaN	0.000000	0.000
25%	2828.250000	1959.000000	NaN	NaN	35303.000000	0.000000	0.000000	NaN	24.000000	23.750
50%	5458.500000	1970.000000	NaN	NaN	51381.500000	0.000000	0.000000	NaN	49.000000	173.500
75%	8427.750000	1977.000000	NaN	NaN	68522.000000	1.000000	1.000000	NaN	74.000000	504.250
max	11191.000000	1996.000000	NaN	NaN	666666.000000	2.000000	2.000000	NaN	99.000000	1493.000

11 rows × 22 columns

In [9]:
```
# checking for a typo category in the categorical variables
store_data.groupby('Marital_Status')['Id'].count()

# no typos
```

Out[9]:
```
Marital_Status
Absurd        2
Alone         3
Divorced    232
Married     864
Single      480
Together    580
Widow        77
YOLO          2
Name: Id, dtype: int64
```

In [10]:
```
store_data.groupby('Education')['Id'].count()

# no typos
```

Out[10]:
```
Education
2n Cycle       203
Basic           54
Graduation    1127
Master         370
PhD            486
Name: Id, dtype: int64
```

In [11]:
```
# checking for nulls
store_data.isna().sum()

# 24 in the Income column
```

Out[11]:
```
Id                      0
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
Response                0
Complain                0
dtype: int64
```

In [12]:
```
# checking for duplicates
store_data.duplicated().sum()

# None found
```

Out[12]:
```
0
```

3. Feature Engineering

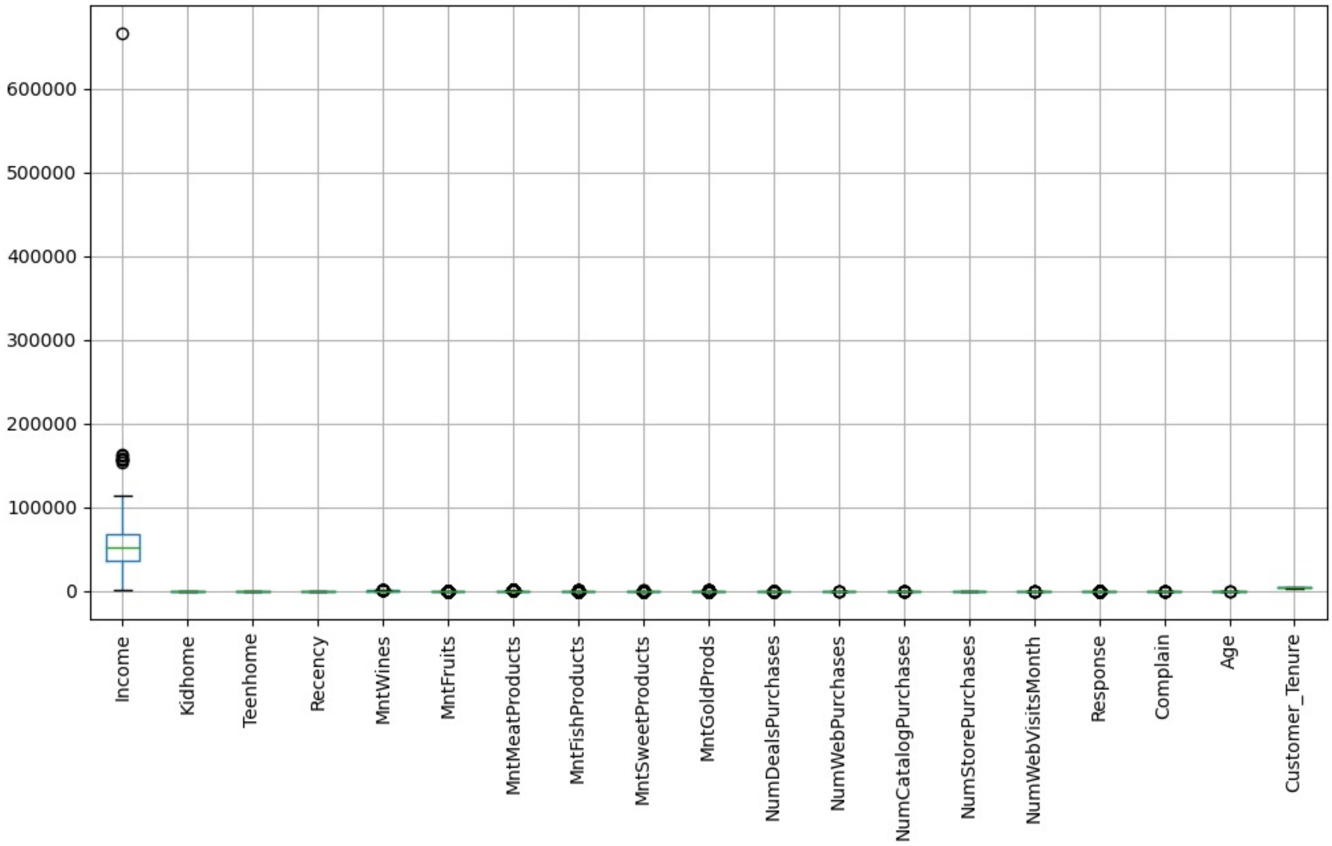
In [13]:
```
# replacing nulls with the median
store_data['Income'] = store_data['Income'].fillna(store_data['Income'].median())
```

In [14]:
```
# confirming replacement
store_data['Income'].isna().sum()
```

Out[14]:
```
0
```

In [15]:
```
# grouping Alone, Absurd & YOLO as Single
store_data['Marital_Status'] = store_data['Marital_Status'].apply(lambda x: 'Single' if x in
                                                                  ['Alone','Absurd','YOLO'] else x)

# creating Age column
store_data['Age'] = 2024 - store_data['Year_Birth']

# creating Customer Tenure column
store_data['Dt_Customer'] = pd.to_datetime(store_data['Dt_Customer'])

store_data['Customer_Tenure'] = (pd.Timestamp('2024-01-01') - store_data['Dt_Customer']).dt.days

# dropping Id, Year_Birth and Dt_Customer columns
store_data.drop(columns=['Id', 'Year_Birth','Dt_Customer'], axis=1, inplace=True)
```

In [16]:
```
# checking for outliers
store_data.boxplot(figsize=(12,6), rot=90)
plt.show()

#we have a big outlier in the Income column
```



In [67]:
```
# singling out the outlier
store_data.query('Income>200000')
```

Out[67]:

	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	...	MntGo
527	Graduation	Together	666666.0	1	0	23	9	14	18	8	...	

1 rows × 21 columns

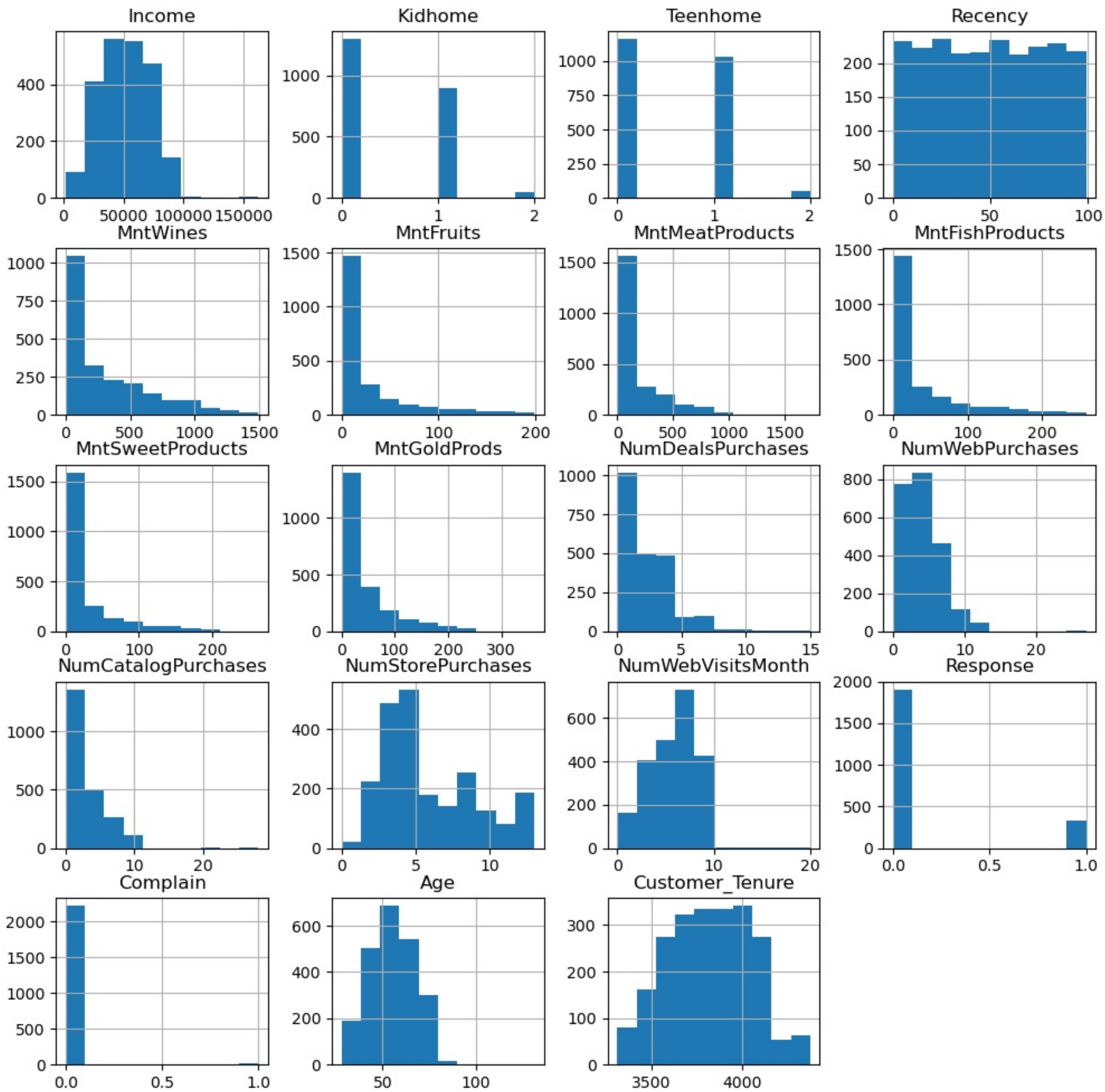
In [18]:
```
# filtering the outlier out
store_data.query('Income < 200000', inplace=True)
```

4. Encoding Categorical Variables

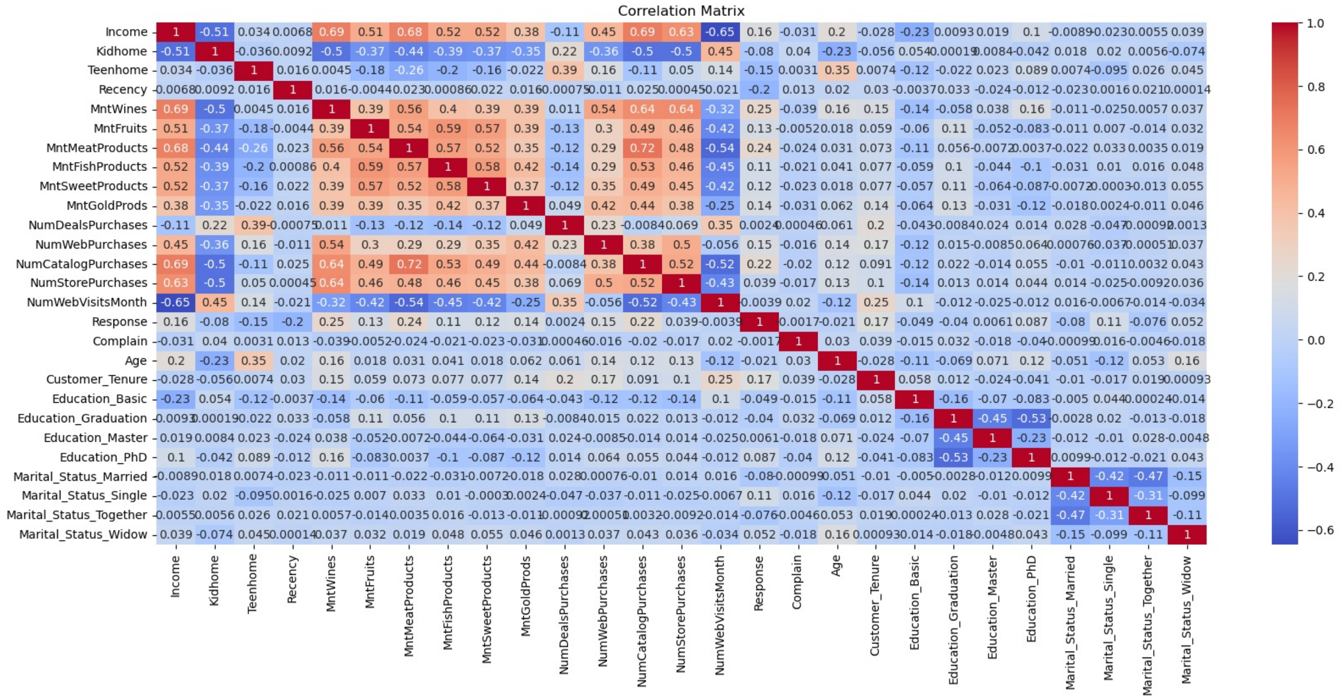
In [20]:
```
# Education & Marital_Status
stored_data = pd.get_dummies(store_data, columns=['Education','Marital_Status'], drop_first=True)
```

Visualization

In [25]:
```
# histogram
stored_data.hist(figsize=(12,12))
plt.show()
```




```python
# correlation
corr_matrix = stored_data.corr()

plt.figure(figsize=(20,8))
sns.heatmap(corr_matrix, annot=True, cmap=('coolwarm'))

plt.title('Correlation Matrix')
plt.show()

# Red - Strong positive correlation
# Blue - Strong negative correlation
# Neutral colors - Weak or no correlation
```



5. Modelling & Training

In [27]:
```python
# splitting data

X = stored_data.drop('Response', axis=1)
y = stored_data['Response']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [28]:
```python
# feature scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [29]:
```python
# model training

model = RandomForestClassifier(random_state=42)

model.fit(X_train, y_train)
```

Out[29]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [30]:
```python
# prediction
y_pred = model.predict(X_test)
```

6. Model Evaluation

In [33]:
```python
# accuracy score

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8586309523809523
```

In [34]:
```python
# confusion matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[552  19]
 [ 76  25]]
```

In [35]:
```python
# classification report
cr = classification_report(y_test, y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92       571
           1       0.57      0.25      0.34       101

    accuracy                           0.86       672
   macro avg       0.72      0.61      0.63       672
weighted avg       0.83      0.86      0.83       672
```

7. Feature Importance

In [37]:

```
# feature importance
feature_importances = pd.DataFrame(model.feature_importances_, index=X.columns, columns=['importance'])
                      .sort_values('importance', ascending=False)
print(feature_importances)
```

```
                         importance
Recency                    0.103558
Income                     0.098653
MntWines                   0.085124
Customer_Tenure            0.081380
MntMeatProducts            0.077487
MntGoldProds               0.062622
NumStorePurchases          0.050944
NumCatalogPurchases        0.050609
Age                        0.050178
MntSweetProducts           0.049180
MntFishProducts            0.046985
MntFruits                  0.044712
NumWebVisitsMonth          0.035905
NumWebPurchases            0.035589
NumDealsPurchases          0.035462
Teenhome                   0.016910
Education_PhD              0.012300
Marital_Status_Single      0.011668
Kidhome                    0.009472
Education_Graduation       0.008654
Marital_Status_Together    0.008611
Marital_Status_Married     0.008567
Education_Master           0.007530
Marital_Status_Widow       0.005451
Complain                   0.001673
Education_Basic            0.000775
```

Conclusion

- The model was successfully built, with an accuracy of 86%.

Top factors that contribute towards a customer giving a positive response:

- Recency
- Income
- MntWines
- Customer_Tenure
- MntMeatProducts
- MntGoldProds
- NumCatalogPurchases
- NumStorePurchases
- Age

Processing math: 100%