Unsupervised Machine Learning

IBM Skills Network

Project Report

By:

Fahd Seddik

Table of Contents

Main Objective

As a company, you would want to know which of your customers would prefer a certain product or maybe you're making a new campaign and you're wondering how much percent of your customers would react positively to that campaign. This would be solved by using customer segmentation.

The target is to do customer segmentation based on customer's information using several clustering algorithms.

Brief Description

Customer Segmentation is to subdivide customers of a market into discrete customer groups that share similar characteristics. Customer Segmentation can be a powerful means to identify unsatisfied customer needs. Using the above data companies can then outperform the competition by developing uniquely appealing products and services. Our data set consists of **2000 examples** and we have **7 features** excluding an ID column. The data represents information about customers. The features are as follows:

- ID (unique)
- Sex
- Marital status
- Age
- Education
- Income
- Occupation
- Settlement size

```
data.head()
```

	ID	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
0	100000001	0	0	67	2	124670	1	2
1	100000002	1	1	22	1	150773	1	2
2	100000003	0	0	49	1	89210	0	0
3	100000004	0	0	45	1	171565	1	1
4	100000005	0	0	53	1	149031	1	1

Data cleaning & Feature engineering

We want to first examine our data set and check for any missing values. After that, we will be checking for duplicates by using the 'ID' column to check – for it being unique for each customer. We would use data.info() to check for missing values. As shown below, we can see that there are no missing values as we have 2000 non-null out of 2000 entries and there are no missing values.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               2000 non-null   int64
 1   Sex              2000 non-null   int64
 2   Marital status   2000 non-null   int64
 3   Age              2000 non-null   int64
 4   Education        2000 non-null   int64
 5   Income           2000 non-null   int64
 6   Occupation       2000 non-null   int64
 7   Settlement size  2000 non-null   int64
dtypes: int64(8)
memory usage: 125.1 KB
```

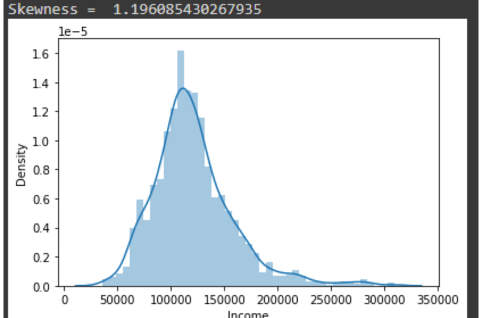
```
data.ID.is_unique

True
```

As we can see the features Sex,Marital status,Education, Occupation, and Settlement size are all ordinal encoded. We want to check for skewness values for Age, and Income.

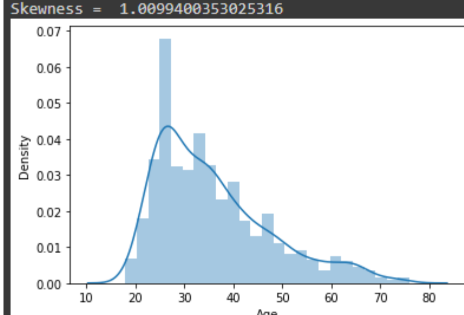
```
data.drop('ID',axis=1,inplace=True)
data.describe()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
count	2000.000000	2000.000000	2000.000000	2000.00000	2000.000000	2000.000000	2000.000000
mean	0.457000	0.496500	35.909000	1.03800	120954.419000	0.810500	0.739000
std	0.498272	0.500113	11.719402	0.59978	38108.824679	0.638587	0.812533
min	0.000000	0.000000	18.000000	0.00000	35832.000000	0.000000	0.000000
25%	0.000000	0.000000	27.000000	1.00000	97663.250000	0.000000	0.000000
50%	0.000000	0.000000	33.000000	1.00000	115548.500000	1.000000	1.000000
75%	1.000000	1.000000	42.000000	1.00000	138072.250000	1.000000	1.000000
max	1.000000	1.000000	76.000000	3.00000	309364.000000	2.000000	2.000000

```
print('Skewness = ',data['Income'].skew())
sns.distplot(data['Income']);

Skewness =  1.196085430267935
```



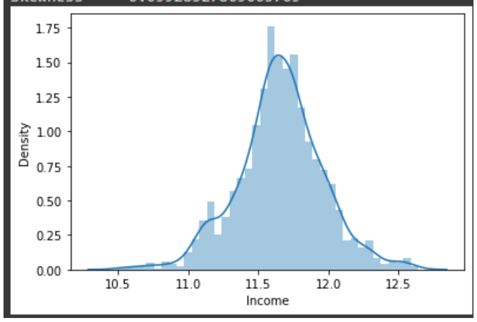

```
print('Skewness = ',data['Age'].skew())
sns.distplot(data['Age']);

Skewness =  1.0099400353025316
```



As we can see from the distplots above, the columns Age, and Income are both skewed with positive skewness. We will correct this skewness with np.log1p. Shown below are the results of the log transformation performed on each of the two columns.

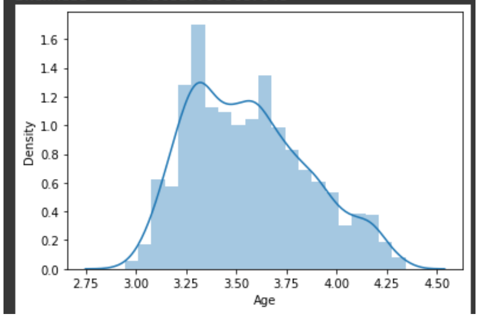
```python
print('Skewness = ',data['Income'].skew())
sns.distplot(data['Income']);
```

```
Skewness =  -0.09928527809005709
```



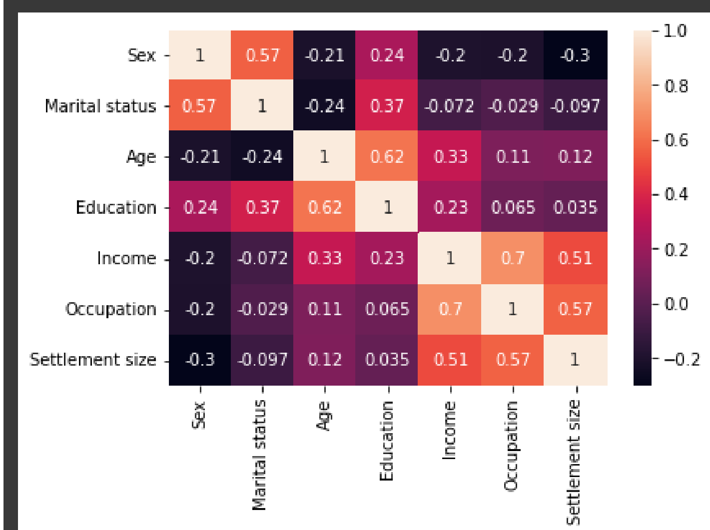

```python
print('Skewness = ',data['Age'].skew())
sns.distplot(data['Age']);
```

```
Skewness =  0.4303257393687048
```



We will visualize correlation between our features in order to have a basic idea of which of them correlate with which of the other features. As you can see below, there are the most correlation features printed together on each row using the idxmax() function. We will then apply standard scaling.

```python
corr_data = data.corr()
sns.heatmap(corr_data,annot=True);
```



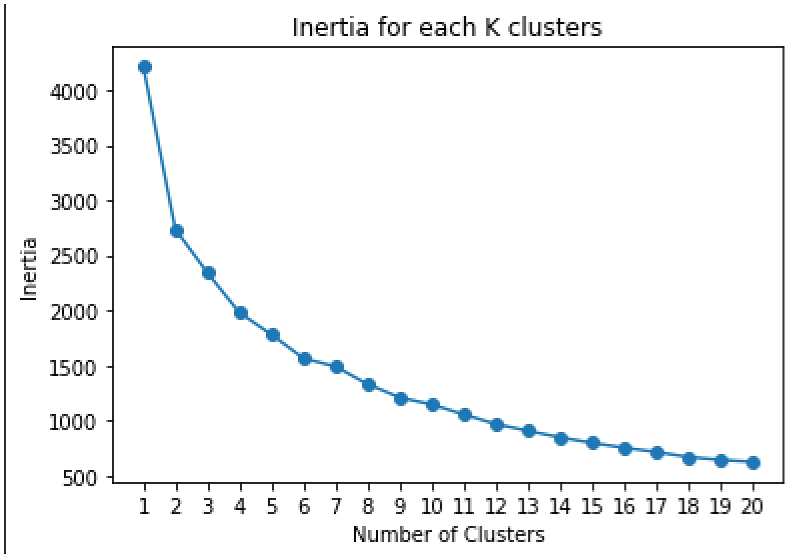

```python
for x in range(len(data.columns)):
  corr_data.iloc[x,x]=0
corr_data.idxmax()
```

```
Sex                Marital status
Marital status                Sex
Age                     Education
Education                     Age
Income                 Occupation
Occupation                 Income
Settlement size        Occupation
dtype: object
```

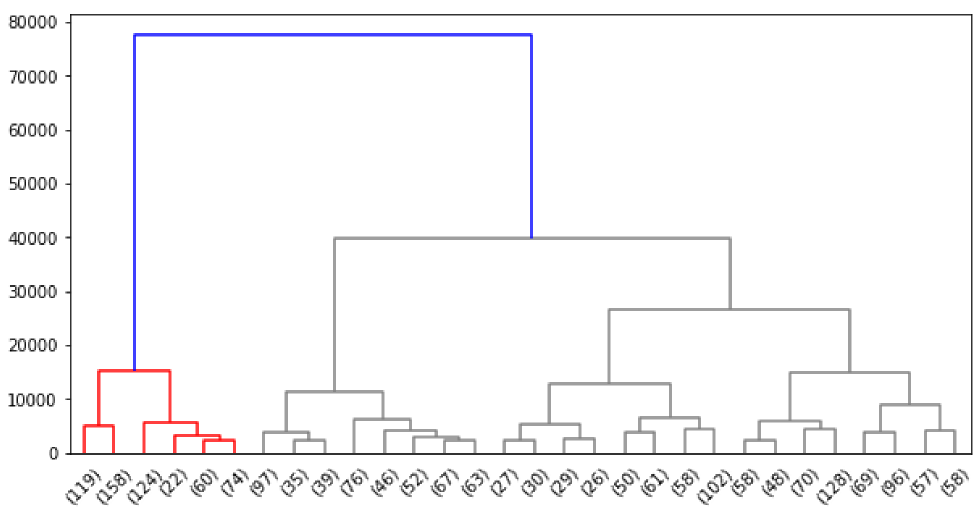
```python
from sklearn.preprocessing import StandardScaler
float_columns = ['Age','Income']
sc = StandardScaler()
data[float_columns] = sc.fit_transform(data[float_columns])
```

Unsupervised Models

We will be using **K-Means**, **AgglomerativeClustering**, and **DBSCAN** as our clustering models. We will first examine the K-Means model as a base-line. We will be using the MiniBatch version as it is fast. We will be using the elbow method to determine which k is best using an inertia plot. However, the plot did not give us any idea of the best value for k.



We will then train our AgglomerativeClustering model and try to visualize the dendrogram of the model using **ward linkage**.




```
ag = AgglomerativeClustering(n_clusters=10, linkage='ward', compute_full_tree=True)
ag.fit(data)

AgglomerativeClustering(compute_full_tree=True, n_clusters=10)

Z = hierarchy.linkage(ag.children_,method='ward')
fig,ax = plt.subplots(figsize=(10,5))
hierarchy.set_link_color_palette(['red','gray'])
den = hierarchy.dendrogram(Z,orientation='top',
                           p=30,truncate_mode = 'lastp',
                           show_leaf_counts = True,ax=ax,
                           above_threshold_color='blue')
```

For our last model we will be using DBSCAN to try to not specify number of clusters and see if we can better cluster our data. However, DBSCAN requires two parameters and finding appropriate values of epsilon and n_clu can be difficult. We have used a eps=0.5 and min_samples=20 for our DBSCAN model which lead to group our data into 16 clusters.

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.5,min_samples=20)
db=db.fit(data)
clusters=db.labels_

pd.DataFrame(clusters).nunique()

0    16
dtype: int64
```

Recommended Model

Without a doubt, each of our models did its best to try to cluster our data given the specific hyperparameters of each of them. However, we can see that a DBSCAN can be hard to train and find the right hyperparameters. Therefore, we recommend the usage of **AgglomerativeClustering** as the recommended model in order to help visualize our customer segmentation more. This would help us negotiate with decision makers and lead to moderate results.

Key Findings and Insights

We found that using elbow method in an inertia plot for a K-Means algorithm does not always yield the results we want. As seen in previous sections, the curve for inertia did not have any “elbow” inflection point. On the contrary, a model like Agglomerative Clustering can be easily visualized using dendrograms which would help us in many data sets specially in customer segmentation to visualize groups of our customers and better understand their needs.

Suggestions

It is certain that improvements can be made for all our models. Some of the suggestions would be to use grid search to choose hyperparameters of our model. We can also try and to use other evaluation metrics. On top of that, the Mean-shift model might be able to perform better on such a problem since it eliminates having to choose a value for “k”. Lastly, if the data set contained more features we can use PCA or LDA to try to figure out underlying correlations.