Photo by Nareeta Martin on Unsplash

HANDS-ON TUTORIAL

The k-prototype as Clustering Algorithm for Mixed Data Type (Categorical and Numerical)

The explanation of the theory and its application in real problems

Audhi Aprilliant Jan 17 · 9 min read ★

techniques and efficiently used for large data is the K-Means algorithm. However, its method is not good and suitable for data that contains categorical variables. This problem happens when the cost function in K-Means is calculated using the Euclidian distance that is only suitable for numerical data. While K-Mode is only suitable for categorical data only, not mixed data types.

Facing these problems, Huang proposed an algorithm called K-Prototype which is created in order to handle clustering algorithms with the mixed data types (numerical and categorical variables). K-Prototype is a clustering method based on partitioning. Its algorithm is an improvement of the K-Means and K-Mode clustering algorithm to handle clustering with the mixed data types.

Read the full of K-Prototype clustering algorithm ***HERE*.**

It's important to know well about the scale measurement from the data.

> ***Note:*** *K-Prototype has an advantage because it's not too complex and is able to handle large data and is better than hierarchical based algorithms*

General formula for the measure of similarity is denoted as follows.

$$d(X_i, Z_l) = \sum_{j=1}^{m} \delta(x_{ij}, z_{lj}) \quad (1)$$

Where $Z_l = \{z_{l1}, z_{l2}, \dots, z_{lm}\}^T$ is a prototype for cluster l. A measure of similarity for numerical variables is well-known as euclidian distance that is denoted as follows.

$$d(X_i, Z_l) = \sqrt{\sum_{j=1}^{m_r} (x_{ij}^r - z_{lj}^r)^2} \quad (2)$$

Where x_{ij}^r is a value of numerical variables j, z_{lj}^r is the average of prototype for numerical variables j cluster m, and number of numerical variables.

While a measure of similarity for categorical variables is denoted as follows.

$$d(X_i, Z_l) = \gamma_l \sum_{j=1+1}^{m_c} \delta(x_{ij}^c, z_{lj}^c) \quad (3)$$

Where simple matching similarity measure for categorical variables is denoted as follows.

$$\delta(x_{ij}^c, z_{lj}^c) = \begin{cases} 0, & x_{ij}^c = z_{lj}^c \\ 1, & x_{ij}^c \neq z_{lj}^c \end{cases} \quad (4)$$

Where γ_l denotes the weight for categorical variables for cluster l that is standard deviation of numerical variables in each clusters. The x_{ij}^c denotes the categorical variables, z_{lj}^c is the mode for variables j cluster l, and m_c denotes the number of categorical variables.

The modification of simple matching similarity measure as follows.

$$\delta(x_{ij}^c, z_{lj}^c) = \begin{cases} 1 - \omega(x_{ij}^c, l), & x_{ij}^c = z_{lj}^c \\ 1 \quad , & x_{ij}^c \neq z_{lj}^c \end{cases} \quad (5)$$

The above formula increases the object similarity within cluster with categorical variables so that the result will be better where $\omega(x_{ij}^c, l)$ denotes the weight for x_{ij}^c where

$$\omega(x_{ij}^c, l) = \frac{f(x_{ij}^c | c_l)}{|c_l| \cdot f(x_{ij}^c | D)} \quad (6)$$

Where $f(x_{ij}^c | c_l)$ is the frequency of x_{ij}^c in cluster l and $|c_l|$ is the number of object in cluster l, and $f(x_{ij}^c | c_l)$ is the frequency of x_{ij}^c in the whole of data.

$$d(X_i, Z_l) = \sqrt{\sum_{j=1} (x_{ij}^r - z_{lj})^2} + \gamma_l \sum_{j=1+1} \delta(x_{ij}^c, z_{lj}^c) \quad (7)$$

Huang Cost Function
Huang declared that cost function equation for mixed data type (numerical and categorical) is as follows.

$$Cost_l = \sum_{l=1}^{k} u_{il} \sum_{j=1}^{m_r} (x_{ij}^r - z_{ij}^r)^2 + \gamma_l \sum_{j=1}^{m_c} u_{il} \sum_{j=1}^{m_c} \delta(x_{ij}^c, z_{ij}^c)$$
$$Cost_l = Cost_l^r + Cost_l^c \quad (8)$$

Where $Cost_l^r$ denotes the total cost of all the numerical variables for the entire objects within cluster l. $Cost_l^r$ is minimized while z_{lj} being calculated with following equation.

$$z_{lj} = \frac{1}{n_l} \sum_{i=1}^{n} u_{il} \cdot x_{ij} \text{ for } j = 1, 2, \dots, m \quad (9)$$

Where $n_l = \sum_{i=1}^{n} u_{il} \cdot x_{ij}$ is the number of objects within cluster l.

Further, the categorical variables e.g. C_j is a set of unique value in each categorical variables j and $p(q_{ij}^c \in C_j | l)$ is the probability for c_j within cluster l. So, $Cost_l^c$ can be rewritten as follows.

$$Cost_l^c = \gamma_l \sum_{j=1}^{m_c} n_l (1 - p(q_{ij}^c \in C_j | l)) \quad (10)$$

where n_l denotes the objects within cluster l. The solution in order to minimize the $Cost_l^c$ is explained clearly in **lemma 1**.

Lemma 1
For special cluster l, $Cost_l^c$ is minimized if and only if $p(z_{ij}^c \in C_j | l) \geq p(c_j \in C_j | l)$ for $z_{ij}^c \neq c_j$ to all categorical variables. So that *cost function* can be rewritten as follows.

$$Cost = \sum_{l=1}^{k} (Cost_l^r + Cost_l^c)$$
$$Cost = \sum_{l=1}^{k} Cost_l^r + \sum_{l=1}^{k} Cost_l^c \quad (11)$$
$$Cost = Cost^r + Cost^c$$

Because $Cost^r$ and $Cost^c$ are non-negative, $Cost$ minimalization can be done by minimizing the $Cost^r$ and $Cost^c$.

The mathematics formula for K-Prototype clustering algorithm (Image by Author)

The application of K-Prototype

In this part, we will demonstrate the implementation of K-Prototype using Python. Before that, it's important to install the `kmodes` module first using the terminal or Anaconda prompt.

There are a few modules used for demonstration. They are `pandas` for data manipulation, `numpy` for linear algebra calculation, `plotnine` as data visualization, and `kmodes` for K-Prototype clustering algorithm.

```
# Import module for data manipulation
import pandas as pd
# Import module for linear algebra
import numpy as np
# Import module for data visualization
from plotnine import *
import plotnine

# Import module for k-protoype cluster
from kmodes.kprototypes import KPrototypes
```

```
# Format scientific notation from Pandas
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

The data can be downloaded ***here*** or you can easily generate this data by visiting ***this website***. It's totally free of charge. Enjoy!

```
# Load the data
df = pd.read_csv('data/10000 Sales Records.csv')
# The dimension of data
print('Dimension data: {} rows and {} columns'.format(len(df),
len(df.columns)))
# Print the first 5 rows
df.head()
```

The data is actually the **Country Sales Data.** The data has 10,000 rows and 14 columns with mixed data types (numerical and categorical). It records the transaction of sales by country around the world.

```
Dimension data: 10000 rows and 14 columns
```

	Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
0	Sub-Saharan Africa	Chad	Office Supplies	Online	L	1/27/2011	292494523	2/12/2011	4484	651.21	524.96	2920025.64	2353920.64	566105.00
1	Europe	Latvia	Beverages	Online	C	12/28/2015	361825549	1/23/2016	1075	47.45	31.79	51008.75	34174.25	16834.50
2	Middle East and North Africa	Pakistan	Vegetables	Offline	C	1/13/2011	141515767	2/1/2011	6515	154.06	90.93	1003700.90	592408.95	411291.95
3	Sub-Saharan Africa	Democratic Republic of the Congo	Household	Online	C	9/11/2012	500364005	10/6/2012	7683	668.27	502.54	5134318.41	3861014.82	1273303.59
4	Europe	Czech Republic	Beverages	Online	C	10/27/2015	127481591	12/5/2015	3491	47.45	31.79	165647.95	110978.89	54669.06

The country sales data generated by MS Excel for K-Prototype (Image by Author)

To make sure the data type of each column is mapped properly, we must inspect their type manually using `df.info()` command. If we found there is false mapping, we should correct it to the right data type.

```
# Inspect the data type
df.info()
```

```
---  ------          --------------  -----
 0   Region          10000 non-null  object
 1   Country         10000 non-null  object
 2   Item Type       10000 non-null  object
 3   Sales Channel   10000 non-null  object
 4   Order Priority  10000 non-null  object
 5   Order Date      10000 non-null  object
 6   Order ID        10000 non-null  int64
 7   Ship Date       10000 non-null  object
 8   Units Sold      10000 non-null  int64
 9   Unit Price      10000 non-null  float64
 10  Unit Cost       10000 non-null  float64
 11  Total Revenue   10000 non-null  float64
 12  Total Cost      10000 non-null  float64
 13  Total Profit    10000 non-null  float64
dtypes: float64(5), int64(2), object(7)
memory usage: 1.1+ MB
```

The data type for columns of country sales data (Image by Author)

Luckily, all the columns have the right data type. Please ignore the `Order ID` because we will not use it and will be removed later.

There are seven categorical variables in the dataset. The `Country` that has the 185 unique values, `Order Date` with 2691 unique values, and `Ship Date` with 2719 unique values will be removed from cluster analysis because they have a lot of unique values. The rest of the columns will be kept. They are `Region` with 7 unique values, `Item Type` with 12 unique values, `Sales Channel` that has the 2 unique values and `Order Priority` with 4 unique values.

```
# Inspect the categorical variables
df.select_dtypes('object').nunique()
```

```
Region              7
Country           185
Item Type          12
Sales Channel       2
Order Priority      4
Order Date       2691
Ship Date        2719
dtype: int64
```

Categorical variables in sales country data (Image by Author)

```
# Inspect the numerical variables
df.describe()
```

	Order ID	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
count	1.000000e+04	10000.000000	10000.000000	10000.000000	1.000000e+04	1.000000e+04	1.000000e+04
mean	5.498719e+08	5002.855900	268.143139	188.806639	1.333355e+06	9.382658e+05	3.950893e+05
std	2.607835e+08	2873.246454	217.944092	176.445907	1.465026e+06	1.145914e+06	3.775550e+05
min	1.000892e+08	2.000000	9.330000	6.920000	1.679400e+02	1.245600e+02	4.338000e+01
25%	3.218067e+08	2530.750000	109.280000	56.670000	2.885511e+05	1.647855e+05	9.832914e+04
50%	5.485663e+08	4962.000000	205.700000	117.110000	8.000512e+05	4.816058e+05	2.890990e+05
75%	7.759981e+08	7472.000000	437.200000	364.690000	1.819143e+06	1.183822e+06	5.664227e+05
max	9.999342e+08	10000.000000	668.270000	524.960000	6.680027e+06	5.241726e+06	1.738178e+06

The summary statistic for numerical data of country sales data (Image by Author)

The last task before going to data exploration and analysis is to make sure the data doesn't contain missing values.

```
# Check missing value
df.isna().sum()
```

```
Region            0
Country           0
Item Type         0
Sales Channel     0
Order Priority    0
Order Date        0
Order ID          0
Ship Date         0
Units Sold        0
Unit Price        0
Unit Cost         0
Total Revenue     0
Total Cost        0
Total Profit      0
dtype: int64
```

Number of missing value in-country sales data (Image by Author)

capture the phenomenon in the data.

We have the hypothesis that the number of purchases in each region has a strong linear correlation to the total profit. To conclude this, we have two options, descriptive analysis, and inference analysis. For this section, we will choose the first option. Let's see!

```
# The distribution of sales each region
df_region = pd.DataFrame(df['Region'].value_counts()).reset_index()
df_region['Percentage'] = df_region['Region'] /
df['Region'].value_counts().sum()
df_region.rename(columns = {'index':'Region', 'Region':'Total'},
inplace = True)
df_region = df_region.sort_values('Total', ascending =
True).reset_index(drop = True)
# The dataframe
df_region = df.groupby('Region').agg({
    'Region': 'count',
    'Units Sold': 'mean',
    'Total Revenue': 'mean',
    'Total Cost': 'mean',
    'Total Profit': 'mean'
    }
).rename(columns = {'Region':
'Total'}).reset_index().sort_values('Total', ascending = True)
```

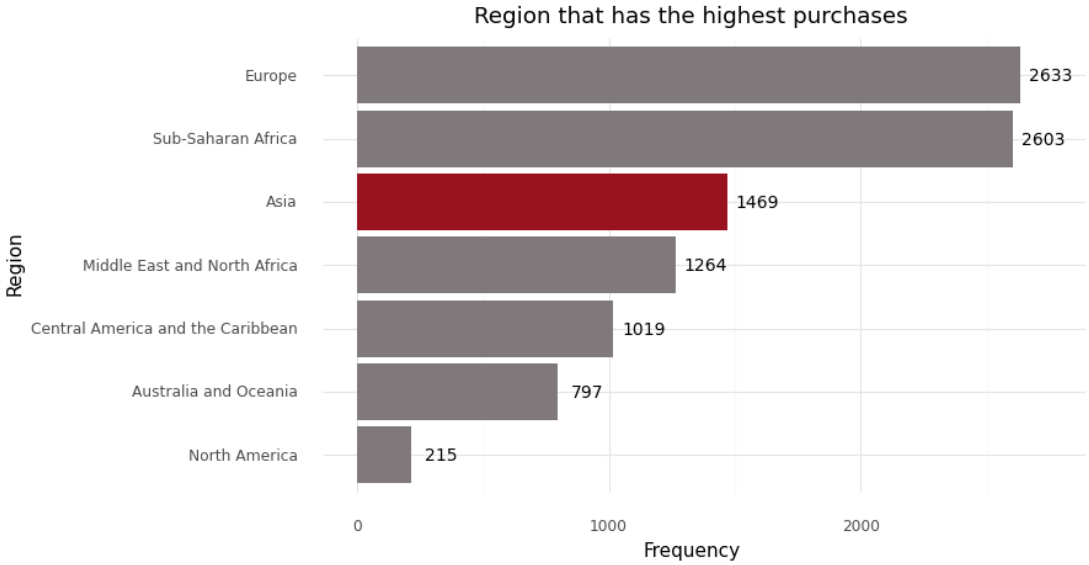
	Region	Total	Units Sold	Total Revenue	Total Cost	Total Profit
5	North America	215	5373.358140	1.559779e+06	1.097009e+06	462769.837767
1	Australia and Oceania	797	4986.769134	1.317192e+06	9.105785e+05	406613.816073
2	Central America and the Caribbean	1019	5081.062807	1.369509e+06	9.736721e+05	395836.947713
4	Middle East and North Africa	1264	5116.219146	1.357305e+06	9.538842e+05	403420.802634
0	Asia	1469	5015.488087	1.365082e+06	9.652160e+05	399866.097243
6	Sub-Saharan Africa	2603	4967.807530	1.287190e+06	9.031555e+05	384034.610642
3	Europe	2633	4920.384732	1.322207e+06	9.321582e+05	390049.226282

The sales distribution in each region in sales country data (Image by Author)

From the above result, we can conclude that North America is the region with the lowest number of sales but they outperform all regions in certain columns such as `Units Sold`, `Total Revenue`, `Total Cost`, and `Total Profit`. Unlike other regions,

means that the number of purchases is not having a strong linear correlation to the total profit.

```
# Data viz
plotnine.options.figure_size = (8, 4.8)
(
    ggplot(data = df_region)+
    geom_bar(aes(x = 'Region',
                 y = 'Total'),
             fill = np.where(df_region['Region'] == 'Asia',
'#981220', '#80797c'),
             stat = 'identity')+
    geom_text(aes(x = 'Region',
                  y = 'Total',
                  label = 'Total'),
              size = 10,
              nudge_y = 120)+
    labs(title = 'Region that has the highest purchases')+
    xlab('Region')+
    ylab('Frequency')+
    scale_x_discrete(limits = df_region['Region'].tolist())+
    theme_minimal()+
    coord_flip()
)
```



The number of sales each region in sales country data (Image by Author)

For the data exploration, we will create a cross-tabulation between `Region` and `Item Type` to look out for any patterns.

```
# distribution of item type
df_item = pd.crosstab(df['Region'], df['Item Type'], margins =
True).reindex(order_region, axis = 0).reset_index()
# Remove index name
df_item.columns.name = None
df_item
```

	Region	Baby Food	Beverages	Cereal	Clothes	Cosmetics	Fruits	Household	Meat	Office Supplies	Personal Care	Snacks	Vegetables	All
0	North America	21	20	16	21	20	15	20	17	20	17	16	12	215
1	Australia and Oceania	65	50	69	77	75	55	78	61	50	76	72	69	797
2	Central America and the Caribbean	74	92	77	84	77	81	104	75	94	82	89	90	1019
3	Middle East and North Africa	105	96	104	111	99	104	128	101	103	112	95	106	1264
4	Asia	132	108	121	116	125	111	116	114	132	137	120	137	1469
5	Sub-Saharan Africa	235	220	211	229	203	230	218	207	207	223	221	199	2603
6	Europe	210	196	227	234	235	199	211	223	231	241	203	223	2633
7	All	842	782	825	872	834	795	875	798	837	888	816	836	10000

The distribution of item type purchased by each region (Image by Author)

Data pre-processing aims to remove the unused columns which are `Country`, `Order Date`, `Order ID`, and `Ship Date`. They are irrelevant regarding the K-Prototype clustering algorithm. There are two reasons why we need to remove these columns as follows:

- `Country` — it has a lot of unique values that add to the computational load. The information is too much to process and becomes meaningless
- `Order Date` and `Ship Date` — the clustering algorithm needs the assumption that the rows in the data represent the unique observation observed in a certain time period
- `Order ID` — it has meaningless information to the cluster analysis

```
# Data pre-processing
df.drop(['Country', 'Order Date', 'Order ID', 'Ship Date'], axis =
1, inplace = True)
# Show the data after pre-processing
print('Dimension data: {} rows and {} columns'.format(len(df),
len(df.columns)))
df.head()
```

1	Europe	Beverages	Online	C	1075	47.45	31.79	51008.75	34174.25	16834.50
2	Middle East and North Africa	Vegetables	Offline	C	6515	154.06	90.93	1003700.90	592408.95	411291.95
3	Sub-Saharan Africa	Household	Online	C	7683	668.27	502.54	5134318.41	3861014.82	1273303.59
4	Europe	Beverages	Online	C	3491	47.45	31.79	165647.95	110978.89	54669.06

The country sales data without certain columns (Image by Author)

The K-Prototype clustering algorithm in `kmodes` module needs categorical variables or columns position in the data. This task aims to save those in a given variables `catColumnsPos`. It will be added for the next task in cluster analysis. The categorical column position is in the first four columns in the data.

```
# Get the position of categorical columns
catColumnsPos = [df.columns.get_loc(col) for col in
list(df.select_dtypes('object').columns)]
print('Categorical columns           :
{}'.format(list(df.select_dtypes('object').columns)))
print('Categorical columns position  : {}'.format(catColumnsPos))
```

```
Categorical columns           : ['Region', 'Item Type', 'Sales Channel', 'Order Priority']
Categorical columns position  : [0, 1, 2, 3]
```

The position of categorical variables or columns in the country sales data (Image by Author)

Next, convert the data from the data frame to the matrix. It helps the `kmodes` module running the K-Prototype clustering algorithm. Save the data matrix to the variable `dfMatrix`.

```
# Convert dataframe to matrix
dfMatrix = df.to_numpy()
dfMatrix
```

```
array([['Sub-Saharan Africa', 'Office Supplies', 'Online', ...,
        2920025.64, 2353920.64, 566105.0],
       ['Europe', 'Beverages', 'Online', ..., 51008.75, 34174.25,
        16834.5],
       ['Middle East and North Africa', 'Vegetables', 'Offline', ...,
        1003700.9, 592408.95, 411291.95],
       ...,
       ['Sub-Saharan Africa', 'Vegetables', 'Offline', ..., 388847.44,
```

```
dtype=object)
```

The matrix of country sales data for K-Prototype clustering algorithm (Image by Author)

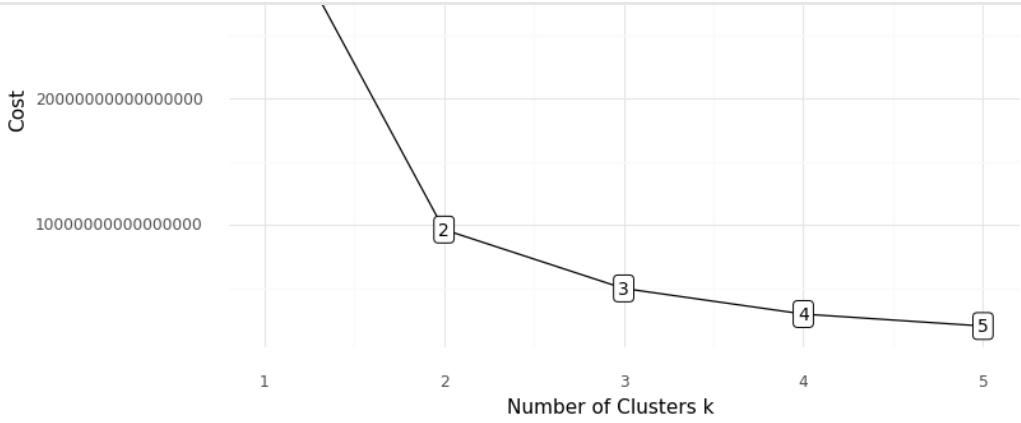
We are using the Elbow method to determine the optimal number of clusters for K-Prototype clusters. Instead of calculating the within the sum of squares errors (WSSE) with Euclidian distance, K-Prototype provides the cost function that combines the calculation for numerical and categorical variables. We can look into the Elbow to determine the optimal number of clusters.

```
# Choose optimal K using Elbow method
cost = []
for cluster in range(1, 10):
    try:
        kprototype = KPrototypes(n_jobs = -1, n_clusters = cluster,
init = 'Huang', random_state = 0)
        kprototype.fit_predict(dfMatrix, categorical =
catColumnsPos)
        cost.append(kprototype.cost_)
        print('Cluster initiation: {}'.format(cluster))
    except:
        break

# Converting the results into a dataframe and plotting them
df_cost = pd.DataFrame({'Cluster':range(1, 6), 'Cost':cost})

# Data viz
plotnine.options.figure_size = (8, 4.8)
(
    ggplot(data = df_cost)+
    geom_line(aes(x = 'Cluster',
                  y = 'Cost'))+
    geom_point(aes(x = 'Cluster',
                   y = 'Cost'))+
    geom_label(aes(x = 'Cluster',
                   y = 'Cost',
                   label = 'Cluster'),
               size = 10,
               nudge_y = 1000) +
    labs(title = 'Optimal number of cluster with Elbow Method')+
    xlab('Number of Clusters k')+
    ylab('Cost')+
    theme_minimal()
)
```

Optimal number of cluster with Elbow Method



The scree plot of a cost function using Elbow Method (Image by Author)

Important! Read more *HERE*

According to the scree plot of cost function above, we consider choosing the number of cluster `k = 3`. It will be the optimal number of clusters for K-Prototype cluster analysis. Read more about the Elbow method ***HERE***.

```
# Fit the cluster
kprototype = KPrototypes(n_jobs = -1, n_clusters = 3, init =
'Huang', random_state = 0)
kprototype.fit_predict(dfMatrix, categorical = catColumnsPos)
```

The algorithm has 7 iterations to converge and it has a cost of 4,960,713,581,025,175.0 (it's quite big right?!). We can print the centroids of cluster using `kprototype.cluster_centroids_`. For the numerical variables, it will be using the average while the categorical using the mode.

```
# Cluster centorid
kprototype.cluster_centroids_
# Check the iteration of the clusters created
kprototype.n_iter_
# Check the cost of the clusters created
kprototype.cost_
```

```
[array([[7.90436555e+03, 5.93526513e+02, 4.57785496e+02, 4.62276055e+06,
         3.55912112e+06, 1.06363942e+06],
        [4.04666949e+03, 1.63259107e+02, 1.05796869e+02, 4.67709452e+05,
         2.81142955e+05, 1.86566497e+05],
```

```
['Europe', 'Cosmetics', 'Online', 'H']], dtype='<U18')]
```

Cluster centroids by K-Prototype (Image by Author)

The interpretation of clusters is needed. The interpretation is using the centroids in each cluster. To do so, we need to append the cluster labels to the raw data. Order the cluster labels will be helpful to arrange the interpretation based on cluster labels.

```
# Add the cluster to the dataframe
df['Cluster Labels'] = kprototype.labels_
df['Segment'] = df['Cluster Labels'].map({0:'First', 1:'Second',
2:'Third'})
# Order the cluster
df['Segment'] = df['Segment'].astype('category')
df['Segment'] =
df['Segment'].cat.reorder_categories(['First','Second','Third'])
```

	Region	Item Type	Sales Channel	Order Priority	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit	Cluster Labels	Segment
0	Sub-Saharan Africa	Office Supplies	Online	L	4484	651.21	524.96	2920025.64	2353920.64	566105.00	2	Third
1	Europe	Beverages	Online	C	1075	47.45	31.79	51008.75	34174.25	16834.50	1	Second
2	Middle East and North Africa	Vegetables	Offline	C	6515	154.06	90.93	1003700.90	592408.95	411291.95	1	Second
3	Sub-Saharan Africa	Household	Online	C	7683	668.27	502.54	5134318.41	3861014.82	1273303.59	0	First
4	Europe	Beverages	Online	C	3491	47.45	31.79	165647.95	110978.89	54669.06	1	Second

The country sales data with cluster information (Image by Author)

To interpret the cluster, for the numerical variables, it will be using the average while the categorical using the mode. But there are other methods that can be implemented such as using median, percentile, or value composition for categorical variables.

```
# Cluster interpretation
df.rename(columns = {'Cluster Labels':'Total'}, inplace = True)
df.groupby('Segment').agg(
    {
        'Total':'count',
        'Region': lambda x: x.value_counts().index[0],
        'Item Type': lambda x: x.value_counts().index[0],
        'Sales Channel': lambda x: x.value_counts().index[0],
        'Order Priority': lambda x: x.value_counts().index[0],
        'Units Sold': 'mean',
        'Unit Price': 'mean',
        'Total Revenue': 'mean',
```

```
).reset_index()
```

	Segment	Total	Region	Item Type	Sales Channel	Order Priority	Units Sold	Unit Price	Total Revenue	Total Cost	Total Profit
0	First	1190	Europe	Household	Offline	L	7904.365546	593.526513	4.622761e+06	3.559121e+06	1.063639e+06
1	Second	6381	Sub-Saharan Africa	Personal Care	Online	C	4046.669488	163.259107	4.677095e+05	2.811430e+05	1.865665e+05
2	Third	2429	Europe	Cosmetics	Online	H	6093.275422	384.264504	1.995888e+06	1.380540e+06	6.153486e+05

The centroid of clusters in the country sales data (Image by Author)

The complete example is listed below.

K-prototype Cluster Algorithm

Import module

In [1]:
```
# Import module for data manipulation
import pandas as pd
# Import module for linear algebra
import numpy as np
# Import module for data visualization
from plotnine import *
import plotnine

# Import module for k-protoype cluster
from kmodes.kprototypes import KPrototypes
```

In [2]:
```
# Ignore warnings
import warnings
warnings.filterwarnings('ignore', category = FutureWarning)
```

kprototype.ipynb hosted with ♡ by GitHub view raw

Complete Python script for K-Prototype clustering algorithm

Conclusion

The K-Prototype is the clustering algorithm which is the combination of K-Means and K-Mode developed by Huang. For the implementation of its algorithm, the researcher needs to filter the columns carefully especially for the categorical variables. The

(cluster initialization) and how the algorithm processes the data to get the converged result. As the researcher, for the final task, interpretation, we need to consider the metrics to use for both numerical and categorical variables.

References

[1] Z. Huang. *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values* (1998). Data Mining and Knowledge Discovery. 2(3): 283–304.