Segmentación de clientes mediante análisis RFM

Aprenda a segmentar a su cliente utilizando el análisis RFM.

Introducción

La segmentación de clientes es la práctica de dividir una base de clientes en grupos de individuos que son similares en formas específicas relevantes para el marketing, como la edad, el género, los intereses y los hábitos de gasto.

Estos son algunos de los principales beneficios de la segmentación de clientes:

- Mensajes de marketing dirigidos a las personas adecuadas
- Se centra en los clientes más rentables
- Puede mejorar el servicio al cliente

En este post, repasaré paso a paso cómo realizar análisis RFM para segmentar a nuestros clientes.

¡Comencemos!

¿Qué es el análisis RFM?

El análisis de RFM es básicamente calificar a nuestros clientes en función de sus valores de actualidad, frecuencia y dinero.

Entendamos qué son la Actualidad, la Frecuencia y la Monetaria.

Novedad: Qué tan recientemente un cliente realizó una compra.

Por ejemplo, si compras algo el 1 de enero y hoy es 1 de febrero tu actualidad es de 30 días.

Frecuencia: Con qué frecuencia los clientes realizan una compra.

Este es el número de pedidos que realizó desde su primera fecha de compra.

Si compras una camiseta el 1 de enero y compras jeans el 15 de enero tu frecuencia es 2.

Valor monetario: Cuánto dinero gasta un cliente en compras.

Vamos a referirnos al ejemplo anterior donde realizaste 2 compras. La camiseta te cuesta 20 $ y los jeans cuestan 35 $. Su valor monetario es de 55 $.

Nota: A veces el valor monetario puede ser el gasto promedio de cada cliente.

Ahora que entendemos qué son la Actualidad, la Frecuencia y el Valor Monetario, podemos ir a nuestro proyecto.

Segmentación de clientes con RFM usando Python

Conjunto de datos

Utilizaremos el conjunto de datos de transacciones de una empresa minorista en línea y segmentaremos a los clientes utilizando RFM.

Puede encontrar el conjunto de datos y el bloc de notas en el Github del curso.

Importación de bibliotecas

```
import pandas as pd
import numpy as npimport matplotlib.pyplot as plt
import seaborn as sns
sns.set()import datetime as dt
import warnings
warnings.filterwarnings('ignore')
```

Lectura del conjunto de datos

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

Comprensión de los datos

```
def summary(df):
    display(df.head())
    print('-'*100)
    display(df.info())
    print('-'*100)
    display(df.describe([0.01,0.25,0.50,0.75,0.99]))summary(df)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

df. cabeza()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495478 entries, 0 to 495477
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    495478 non-null  object 
 1   StockCode    495478 non-null  object 
 2   Description  494024 non-null  object 
 3   Quantity     495478 non-null  int64  
 4   InvoiceDate  495478 non-null  object 
 5   UnitPrice    495478 non-null  float64
 6   CustomerID   361878 non-null  float64
 7   Country      495478 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 30.2+ MB
None
```

df.info()

	Quantity	UnitPrice	CustomerID
count	495478.000000	495478.000000	361878.000000
mean	8.605486	4.532422	15547.871368
std	227.588756	99.315438	1594.402590
min	-80995.000000	-11062.060000	12346.000000
1%	-2.000000	0.190000	12748.000000
25%	1.000000	1.250000	14194.000000
50%	3.000000	2.100000	15514.000000
75%	10.000000	4.130000	16931.000000
99%	100.000000	16.950000	18223.000000
max	80995.000000	38970.000000	18287.000000

df.describe()

Preparación de datos

- Convertir InvoiceDate al formato de fecha.

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

- Excluyendo los valores negativos en cantidad y precio unitario

Vemos valores negativos en Cantidad y Precio Unitario, lo cual es imposible. Necesitamos excluir estos valores.

```
df = df[(df.Quantity>0) & (df.UnitPrice> 0)]
```

- Eliminación de artículos devueltos

Artículos devueltos indicados con C para que podamos soltarlos filtrando

```
df = df[~df['StockCode'].str.contains('C')]
```

- Eliminación de valores duplicados

```
df = df.drop_duplicates()
```

- Valores que faltan

```
df.isnull().sum()
```

```
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     128535
Country            0
dtype: int64
```

df.isnull().sum()

```
df.dropna(inplace=True)
```

- Creación de la columna 'Precio total'

```
df['Total_Price'] = df['UnitPrice']*df['Quantity']
```

Creación de rfm dataframe

```
df['InvoiceDate'].max() # let's see the latest day Output:
Timestamp('2011-12-09 12:49:00')
```

Vemos que la última fecha de compra es 2011–12–09. Podemos determinar nuestra fecha para el análisis como 2011–12–10

```
now =  dt.datetime(2011,12,10)
```

Marco de datos RFM

```
rfm = df.groupby('CustomerID').agg({'InvoiceDate' : lambda day
: (now - day.max()).days,
                                    'InvoiceNo': lambda num :
len(num),
                                   'Total_Price': lambda price :
price.sum()


                                  })col_list =
['Recency','Frequency','Monetary']rfm.columns = col_list
```

Cálculo de puntajes de RFM

Tenga en cuenta que, en el siguiente código, invertimos los números de Recency porque una Recency más baja es mejor para nosotros.

```
rfm["R"] = pd.qcut(rfm["Recency"],5,labels=[5,4,3,2,1])rfm["F"]
= pd.qcut(rfm["Frequency"],5,labels=[1,2,3,4,5])rfm["M"] =
pd.qcut(rfm["Monetary"],5,labels=[1,2,3,4,5])rfm["RFM_Score"] =
rfm["R"].astype(str) +rfm["F"].astype(str) +
rfm["M"].astype(str)
```

CustomerID	Recency	Frequency	Monetary	R	F	M	RFM_Score
12346.0	325	1	77183.60	1	1	5	115
12747.0	2	100	4128.71	5	4	5	545
12748.0	0	4345	32509.54	5	5	5	555
12749.0	3	196	4014.18	5	5	5	555
12820.0	3	59	942.34	5	4	4	544

Conjunto de datos final

Segmentación de clientes por actualidad y frecuencia

Nuestro conjunto de datos está listo para segmentar a nuestros clientes.

```
seg_map = {
    r'[1-2][1-2]': 'Hibernating',
    r'[1-2][3-4]': 'At Risk',
    r'[1-2]5': 'Can\'t Loose',
    r'3[1-2]': 'About to Sleep',
    r'33': 'Need Attention',
    r'[3-4][4-5]': 'Loyal Customers',
    r'41': 'Promising',
    r'51': 'New Customers',
    r'[4-5][2-3]': 'Potential Loyalists',
    r'5[4-5]': 'Champions'
}
```

Determinamos cómo nombrar nuestros segmentos usando regex y ahora estamos listos para crear nuestra columna de segmento.

```
rfm['Segment'] = rfm['R'].astype(str) + rfm['F'].astype(str)
rfm['Segment'] = rfm['Segment'].replace(seg_map, regex=True)
rfm.head()
```

CustomerID	Recency	Frequency	Monetary	R	F	M	RFM_Score	Segment
12346.0	325	1	77183.60	1	1	5	115	Hibernating
12747.0	2	100	4128.71	5	4	5	545	Champions
12748.0	0	4345	32509.54	5	5	5	555	Champions
12749.0	3	196	4014.18	5	5	5	555	Champions
12820.0	3	59	942.34	5	4	4	544	Champions

Vamos a agrupar nuestro conjunto de datos para ver los valores promedio de cada segmento.

```
rfm.groupby('Segment').mean().sort_values('Monetary')
```

Segment	Recency	Frequency	Monetary
About to Sleep	52.219178	15.267123	435.219486
Promising	22.428571	7.183673	445.777347
Hibernating	207.501042	12.985417	537.732126
Need Attention	51.410256	40.523077	815.458821
Potential Loyalists	15.753846	33.876923	879.101826
At Risk	167.796154	55.523077	884.872462
Can't Loose	145.901408	180.098592	2268.055930
Loyal Customers	33.410788	152.181189	2428.635339
New Customers	6.145833	7.062500	4111.827083
Champions	5.471971	275.752260	6129.229024

Agrupar por datos

Calculamos el puntaje de RFM y segmentamos a los clientes utilizando Recency y Frequency.

El análisis RFM es una forma fácil y efectiva de entender a nuestros clientes y comercializarlos de manera efectiva.

Segmentación de clientes paso a paso usando K-Means en Python

Segmenta a tus clientes para un mejor marketing.

Tabla de contenidos

1. Introducción — ¿Qué es la segmentación de clientes?
2. Escenario de negocio
3. Explore el conjunto de datos
4. Preprocesamiento de datos
5. K-Means para la segmentación
6. PCA con K-Means para una mejor visualización
7. Conclusión

Introducción

Digamos que decidiste comprar una camiseta de una marca en línea. ¿Alguna vez has pensado que quién más compró la misma camiseta?

Personas que tienen algo similar a ti, ¿verdad? Misma edad, mismas aficiones, mismo sexo, etc.

¡En marketing, las empresas básicamente intentan encontrar sus camisetas en otras personas!

¿Pero espera? ¿Cómo? ¡Por supuesto, con datos!

¡La segmentación de clientes es así de simple!

En realidad, tratamos de encontrar y agrupar clientes en función de características comunes como la edad, el género, el área de vida, el comportamiento de gasto, etc. Para que podamos comercializar a los clientes de manera efectiva.

¡Vamos a sumergirnos en nuestro proyecto de segmentación!

Escenario de negocio

Supongamos que estamos trabajando como científicos de datos para una empresa de gran consumo y queremos segmentar a nuestros clientes para ayudar al departamento de marketing para que lancen nuevos productos y ventas en función de la

segmentación. Por lo tanto, ahorraremos nuestro tiempo y dinero al comercializar un grupo específico de clientes con productos seleccionados.

¿Cómo recopilamos los datos por cierto?

Todos los datos se han recopilado a través de las tarjetas de fidelización que utilizan en el momento del pago :)

¡Utilizaremos algoritmos K-Means y PCA para este proyecto y veremos cómo definimos nuevos clientes agrupados!

¡Comprender los datos es importante!

Antes de comenzar cualquier proyecto, primero debemos comprender el problema comercial y el conjunto de datos.

Veamos las variables (características) en el conjunto de datos.

Descripción de la variable

ID: Muestra una identificación única de un cliente.

Sexo: Sexo biológico (género) de un cliente. En este conjunto de datos, solo hay 2 opciones diferentes.

0: masculino

1: femenino

Estado civil: Estado civil de un cliente.

0: individual

1: no soltero (divorciado / separado / casado / viudo)

Edad: La edad del cliente en años, calculada como año actual menos el año de nacimiento del cliente en el momento de la creación del conjunto de datos.

Valor mínimo de 18 minutos (la edad más baja observada en el conjunto de datos)

76 valor máximo (la edad más alta observada en el conjunto de datos)

Educación: Nivel de educación del cliente.

0: otro / desconocido

1: escuela secundaria

2: universidad

3: escuela de posgrado

Ingresos: Ingresos anuales autoinformados en pesos del cliente.

35832 Valor mínimo (el ingreso más bajo observado en el conjunto de datos)

309364 valor máximo (el ingreso más alto observado en el conjunto de datos)

Ocupación: Categoría de ocupación del cliente.

0: desempleados/no cualificados

1: empleado calificado / funcionario

2: directivo / autónomo / empleado altamente cualificado / funcionario

Tamaño del asentamiento: El tamaño de la ciudad en la que vive el cliente.

0: ciudad pequeña

1: ciudad de tamaño medio

2: gran ciudad

Tenemos conjuntos de datos y conocemos el problema del negocio. Ahora, ¡comencemos a codificar!

Importación de bibliotecas

¡En este proyecto, necesitaremos algunos amigos que te ayuden en el camino!

Permítanme presentarlos a continuación,

```
### Data Analysis and Manipulation
import pandas as pd
import numpy as np### Data Visualizationimport
matplotlib.pyplot as plt
import seaborn as snssns.set() ## this is for styling### Data
Standardization and Modeling with K-Means and PCAfrom
sklearn.preprocessing import StandardScalerfrom sklearn.cluster
import KMeans
from sklearn.decomposition import PCA
```

3. Explore el conjunto de datos

```
df= pd.read_csv('segmentation data.csv', index_col = 0)
```

Esta parte consiste en comprender los datos con la ayuda del análisis descriptivo y la visualización.

```
df.head()
```

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

Salida df.head()

También podemos aplicar el método de descripción para ver estadísticas descriptivas sobre las columnas.

```
df.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

Vemos la media de Edad e Ingresos 35.90 y 120954 respectivamente. El método Describe es muy útil para columnas numéricas.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000 entries, 100000001 to 100002000
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Sex              2000 non-null   int64
 1   Marital status   2000 non-null   int64
 2   Age              2000 non-null   int64
 3   Education        2000 non-null   int64
 4   Income           2000 non-null   int64
 5   Occupation       2000 non-null   int64
 6   Settlement size  2000 non-null   int64
dtypes: int64(7)
memory usage: 125.0 KB
```

df.info() devuelve información sobre DataFrame, incluidos el tipo de datos de índice y las columnas, los valores no nulos y el uso de memoria.

Vemos que no falta ningún valor en el conjunto de datos y todas las variables son enteras.

Una buena manera de obtener una comprensión inicial de la relación entre las diferentes variables es explorar cómo se correlacionan.

Calculamos la correlación entre nuestras variables utilizando el método corr en la biblioteca de pandas.

```
plt.figure(figsize=(12,9))
sns.heatmap(df.corr(),annot=True,cmap='RdBu')
plt.title('Correlation Heatmap',fontsize=14)
plt.yticks(rotation =0)
plt.show()
```

Exploremos la correlación.

Vemos que existe una fuerte correlación entre Educación y Edad. En otras palabras, las personas mayores tienden a ser más altamente educadas.

¿Qué hay de los ingresos y la ocupación?

Su correlación es de 0,68. Eso significa que, si tiene un salario más alto, es más probable que tenga una ocupación de mayor nivel, como un gerente.

La matriz de correlación es una herramienta muy útil para analizar la relación entre características.

Ahora, entendemos nuestro conjunto de datos y tenemos una idea general de él.

La siguiente sección será la segmentación. Pero antes de eso, primero debemos escalar nuestros datos.

4. Preprocesamiento de datos

Necesitamos aplicar la estandarización a nuestras características antes de usar cualquier modelo de aprendizaje automático basado en la distancia, como K-Means, KNN.

En general, queremos tratar todas las características por igual y podemos lograrlo transformando las características de tal manera que sus valores caigan dentro del mismo rango numérico como [0:1].

Este proceso se conoce comúnmente como Estandarización.

Normalización

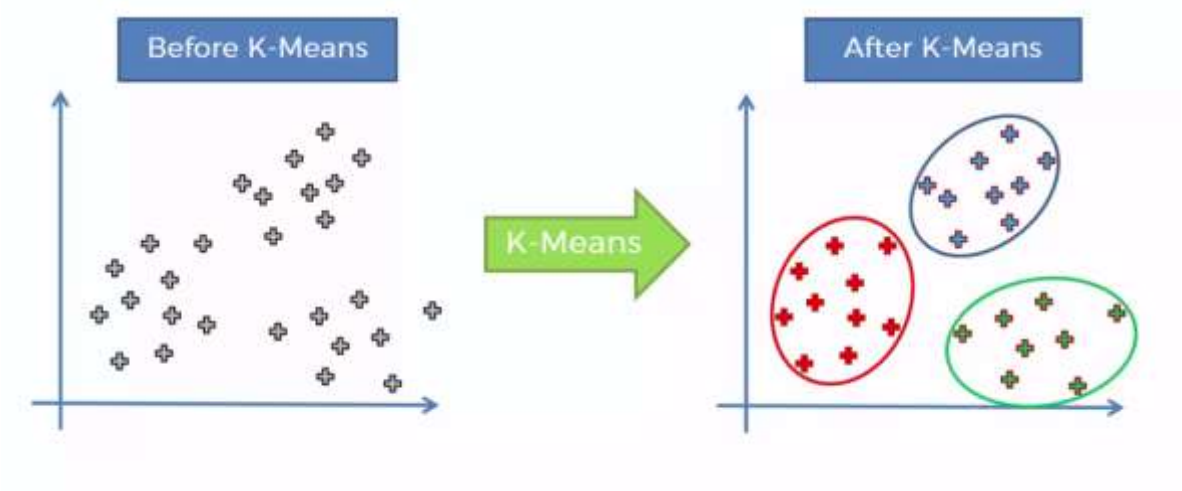
Ahora que lo hemos aclarado. Es hora de realizar la estandarización en Python.

```
scaler = StandardScaler()
df_std = scaler.fit_transform(df)
```

¡Ahora, estamos listos para comenzar a construir el modelo de segmentación K-Means!

```
df_std = pd.DataFrame(data = df_std,columns = df.columns)
```

Construyendo nuestro modelo de segmentación



Antes de aplicar el algoritmo K-Means necesitamos elegir cuántos clusters nos gustaría tener.

¿Pero cómo?

Hay dos componentes. *Dentro de Clusters Sum of Squares (WCSS)* y *Elbow Method.*

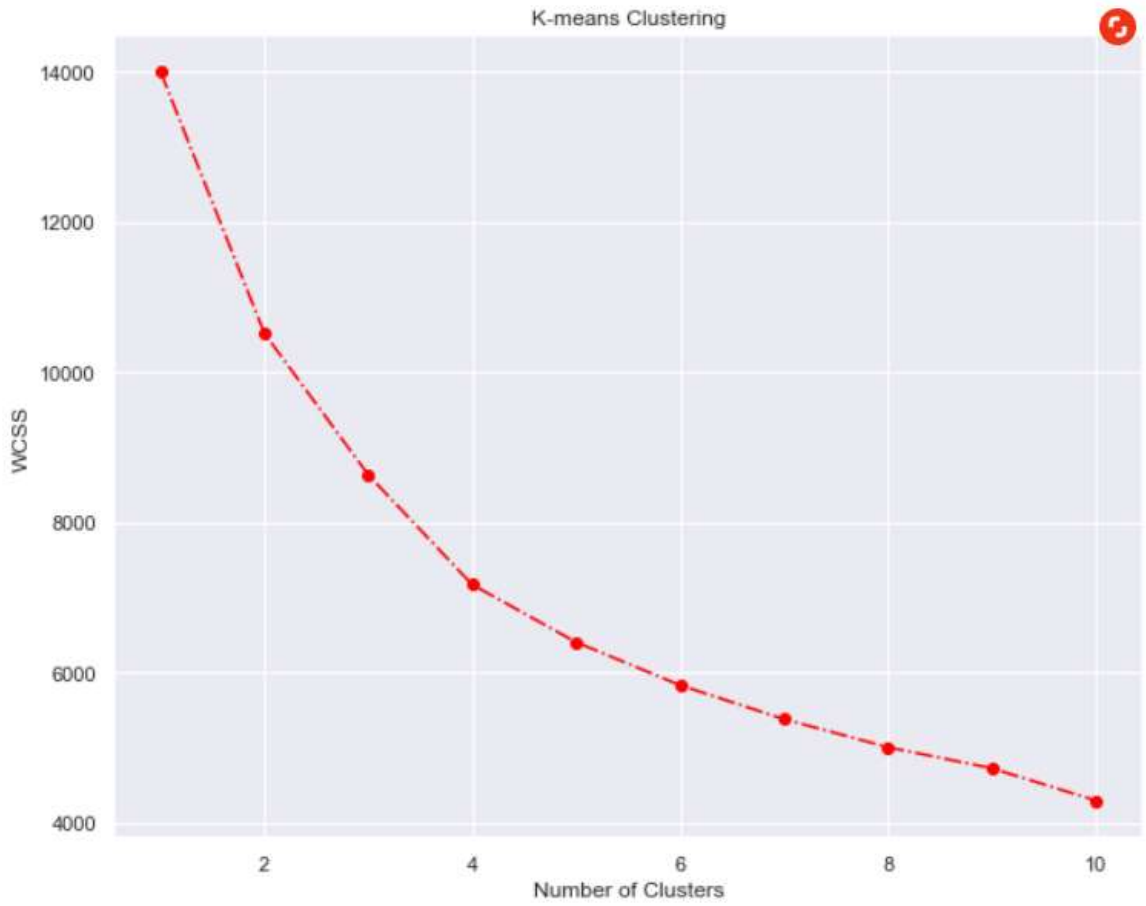
```
wcss = []
for i in range(1,11):
    kmeans_pca = KMeans(n_clusters = i, init = 'k-means++',
random_state = 42)
    kmeans_pca.fit(scores_pca)
    wcss.append(kmeans_pca.inertia_)
```

Almacenamos a cada uno dentro de los clústeres la suma del valor cuadrado a la lista wcss.

Vamos a visualizarlos.

```
plt.figure(figsize = (10,8))
plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '-
.',color='red')
```

```
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('K-means Clustering')
plt.show()
```



El codo en el gráfico es la marca de cuatro grupos. Este es el único lugar hasta el cual el gráfico está disminuyendo abruptamente mientras se suaviza después.

Vamos a realizar el clustering de K-Means con 4 clusters.

```
kmeans = KMeans(n_clusters = 4, init = 'k-means++',
random_state = 42)
```

Ajuste de nuestro modelo al conjunto de datos

```
kmeans.fit(df_std)
```

Creamos un nuevo marco de datos con las características originales y agregamos una nueva columna con los clústeres asignados para cada punto.

```
df_segm_kmeans= df_std.copy()
df_std[‘Segment K-means’] = kmeans.labels_
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Segment K-means
0	-0.917399	-0.993024	2.653614	1.604323	0.097524	0.296823	1.552326	0
1	1.090038	1.007025	-1.187132	-0.063372	0.782654	0.296823	1.552326	2
2	-0.917399	-0.993024	1.117316	-0.063372	-0.833202	-1.269525	-0.909730	1
3	-0.917399	-0.993024	0.775916	-0.063372	1.328386	0.296823	0.321298	3
4	-0.917399	-0.993024	1.458716	-0.063372	0.736932	0.296823	0.321298	3

Ahora vemos los segmentos con nuestro conjunto de datos.

Vamos a agrupar a los clientes por clústeres y ver los valores medios de cada variable.

```
df_segm_analysis = df_std.groupby(['Segment K-means']).mean()
df_segm_analysis
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Segment K-means							
0	0.501901	0.692015	55.703422	2.129278	158338.422053	1.129278	1.110266
1	0.352814	0.019481	35.577922	0.746753	97859.852814	0.329004	0.043290
2	0.853901	0.997163	28.963121	1.068085	105759.119149	0.634043	0.422695
3	0.029825	0.173684	35.635088	0.733333	141218.249123	1.271930	1.522807

Es hora de interpretar nuestro nuevo conjunto de datos,

Comencemos con el primer segmento,

Tiene casi el mismo número de hombres y mujeres con una edad promedio de 56 años. En comparación con otros clústeres, nos damos cuenta de que este es el segmento más antiguo.

Para el segundo segmento, podemos decir,

Este segmento tiene los valores más bajos para el salario anual.

Viven casi exclusivamente en ciudades pequeñas.

Con bajos ingresos viviendo en ciudades pequeñas, parece que este es un segmento de personas con menos oportunidades.

Sigamos con el tercer segmento,

Este es el segmento más joven con una edad promedio de 29 años. Tienen un nivel medio de educación y un ingreso promedio.

También parecen promedio sobre cada parámetro que podemos etiquetar como promedio del segmento o estándar.

Finalmente, llegamos al cuarto segmento,

Está compuesto casi en su totalidad por hombres, menos del 20 por ciento de los cuales están en relaciones.

Al observar los números, observamos valores relativamente bajos para la educación, junto con valores altos para los ingresos y la ocupación.

La mayoría de este segmento vive en ciudades grandes o medianas.

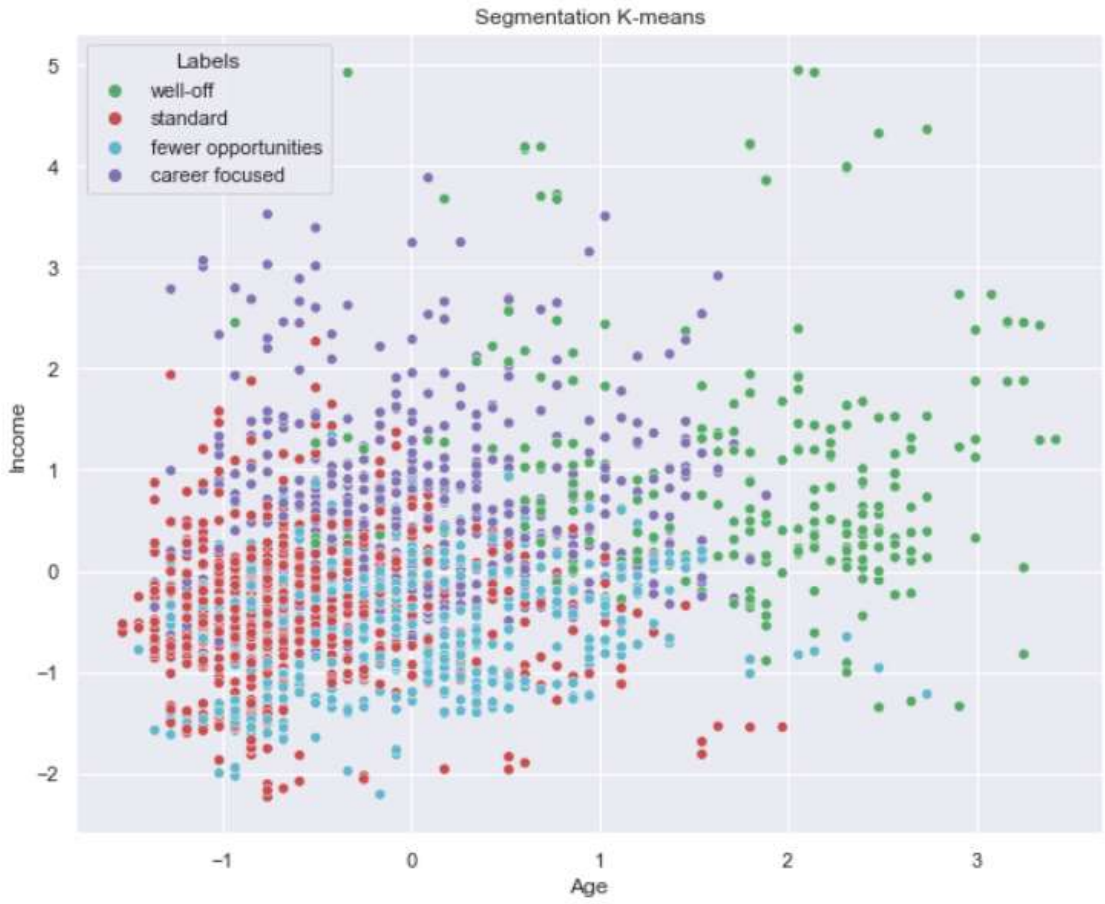
Vamos a etiquetar el segmento de acuerdo a su relevancia.

```
df_segm_analysis.rename({0:'well-off',
                         1:'fewer-opportunities',
                         2:'standard',
                         3:'career focused'})
```

Segment K-means	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
well-off	0.090136	0.391040	1.689452	1.819919	0.981226	0.499317	0.457039
fewer-opportunities	-0.209147	-0.954062	-0.028257	-0.485711	-0.606168	-0.754190	-0.856438
standard	0.796753	1.001351	-0.592830	0.050173	-0.398834	-0.276394	-0.389380
career focused	-0.857528	-0.645647	-0.023378	-0.508091	0.531869	0.722760	0.964888

Por último, podemos crear nuestra gráfica para visualizar cada segmento.

```
x_axis = df_std['Age']
y_axis = df_std['Income']
plt.figure(figsize = (10, 8))
sns.scatterplot(x_axis, y_axis, hue = df_std['Labels'], palette
= ['g', 'r', 'c', 'm'])
plt.title('Segmentation K-means')
plt.show()
```



Podemos ver que el segmento verde está claramente separado, ya que es más alto tanto en edad como en ingresos. Pero los otros tres están agrupados.

¡Podemos concluir que K-Means hizo un trabajo decente! Sin embargo, es difícil separar los segmentos entre sí.

En la siguiente sección, combinaremos PCA y K-Means para tratar de obtener un mejor resultado.

PCA con K-Means para una mejor visualización

Lo que haremos aquí es aplicar la reducción de dimensionalidad para simplificar nuestro problema.

Elegiremos componentes razonables para obtener una mejor solución de clustering que con el estándar K-Means. De modo que nuestro objetivo es ver una trama agradable y clara para nuestros grupos segmentados.

```
pca = PCA()
pca.fit(df_std)
```

Ahora, veamos la relación de varianza explicada por cada componente.

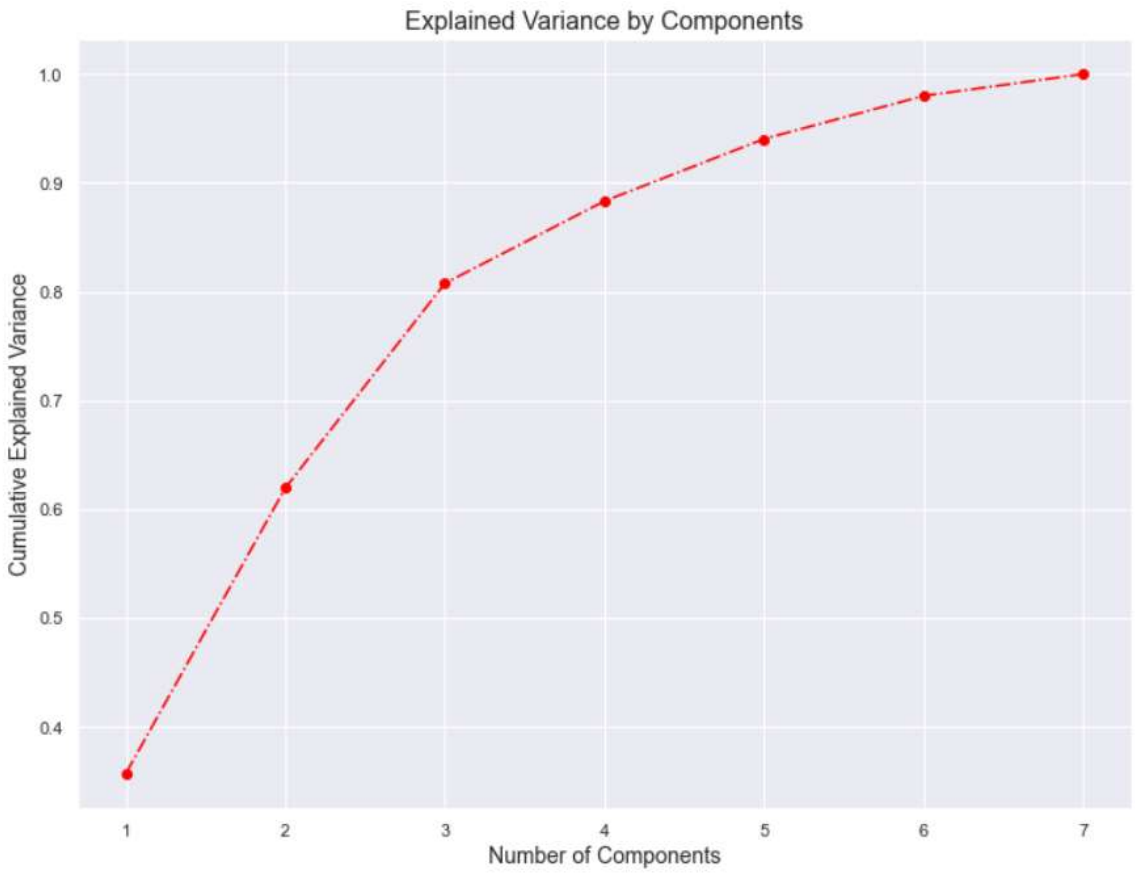
```
pca.explained_variance_ratio_
```

```
array([0.35696328, 0.26250923, 0.18821114, 0.0755775 , 0.05716512,
       0.03954794, 0.02002579])
```

Observamos que el primer componente explica alrededor del 36 % de la variabilidad de los datos. El segundo es del 26 % y así sucesivamente.

Ahora podemos trazar la suma acumulativa de la varianza explicada.

```
plt.figure(figsize = (12,9))
plt.plot(range(1,8), pca.explained_variance_ratio_.cumsum(),
marker = 'o', linestyle = '--')
plt.title('Explained Variance by Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```



Bueno, ¿cómo elegimos el número correcto de componentes? La respuesta es que no hay una respuesta correcta o incorrecta para eso.

Pero, una regla general es mantener al menos del 70 al 80 por ciento de la varianza explicada.

El 80 % de la varianza de los datos se explica por los 3 primeros componentes. Mantengamos los primeros 3 componentes para nuestro análisis posterior.

```
pca = PCA(n_components = 3)
pca.fit(df_std)
pca.components_
```

```
array([[-0.31469524, -0.19170439,  0.32609979,  0.15684089,  0.52452463,
         0.49205868,  0.46478852],
       [ 0.45800608,  0.51263492,  0.31220793,  0.63980683,  0.12468314,
         0.01465779, -0.06963165],
       [-0.29301261, -0.44197739,  0.60954372,  0.27560461, -0.16566231,
        -0.39550539, -0.29568503]])
```

El resultado es una matriz de 3 por 7. Redujimos nuestro futuro a tres componentes de los siete valores originales que explican la forma en que los propios valores muestran las llamadas cargas.

Oye, solo un minuto, ¿qué se está cargando entonces?

Las cargas son correlaciones entre una variable original y el componente.

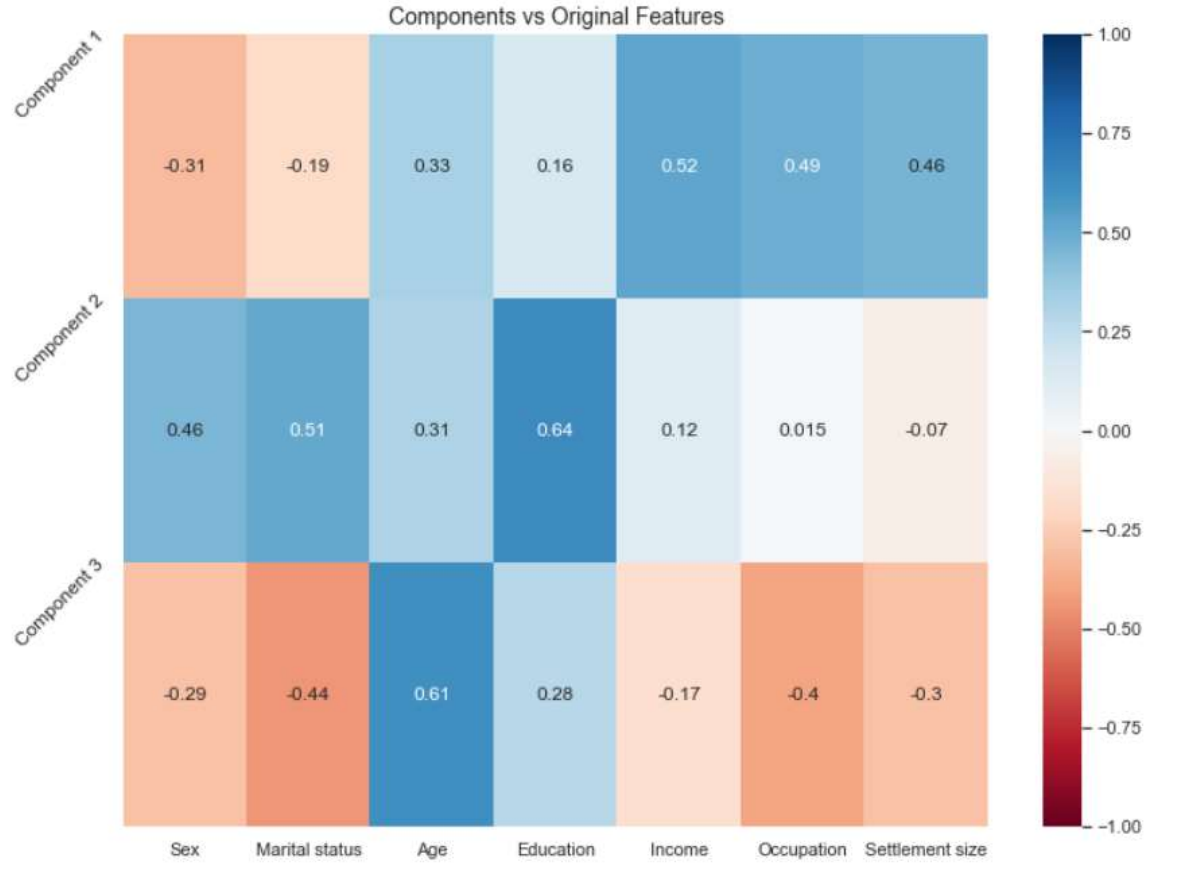
Por ejemplo, el primer valor de la matriz muestra la carga de la primera característica en el primer componente.

Pongamos esta información en un marco de datos de pandas para que podamos verlos bien. Las columnas son siete características originales y las filas son tres componentes que PCA nos dio.

```
df_pca_comp = pd.DataFrame(data = pca.components_,
                           columns = df.columns,
             index = ['Component 1', 'Component 2', 'Component
3'])
df_pca_comp
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Component 1	-0.314695	-0.191704	0.326100	0.156841	0.524525	0.492059	0.464789
Component 2	0.458006	0.512635	0.312208	0.639807	0.124683	0.014658	-0.069632
Component 3	-0.293013	-0.441977	0.609544	0.275605	-0.165662	-0.395505	-0.295685

```
plt.figure(figsize=(12,9))
sns.heatmap(df_pca_comp,
            vmin = -1,
            vmax = 1,
            cmap = 'RdBu',
            annot = True)
plt.yticks([0, 1, 2],
           ['Component 1', 'Component 2', 'Component 3'],
           rotation = 45,
           fontsize = 12)
plt.title('Components vs Original Features',fontsize = 14)
plt.show()
```



Vemos que existe una correlación positiva entre el Componente 1 y *la Edad, los Ingresos, la Ocupación y el tamaño del Asentamiento*. Estos están estrictamente relacionados con la carrera de una persona. Así que este componente muestra el enfoque profesional del individuo.

Para el segundo componente*, el sexo, el estado civil y la educación* son, con mucho, los determinantes más prominentes.

Para el componente final, nos damos cuenta de que la edad, el estado civil y la ocupación son las características más importantes. Observamos que el estado civil y la ocupación cargan negativamente, pero siguen siendo importantes.

Ahora, tenemos una idea sobre nuestras nuevas variables (componentes). Podemos ver claramente la relación entre componentes y variables.

Transformemos nuestros datos y guardémoslos scores_pca.

```
pca.transform(df_std)
scores_pca = pca.transform(df_std)
```

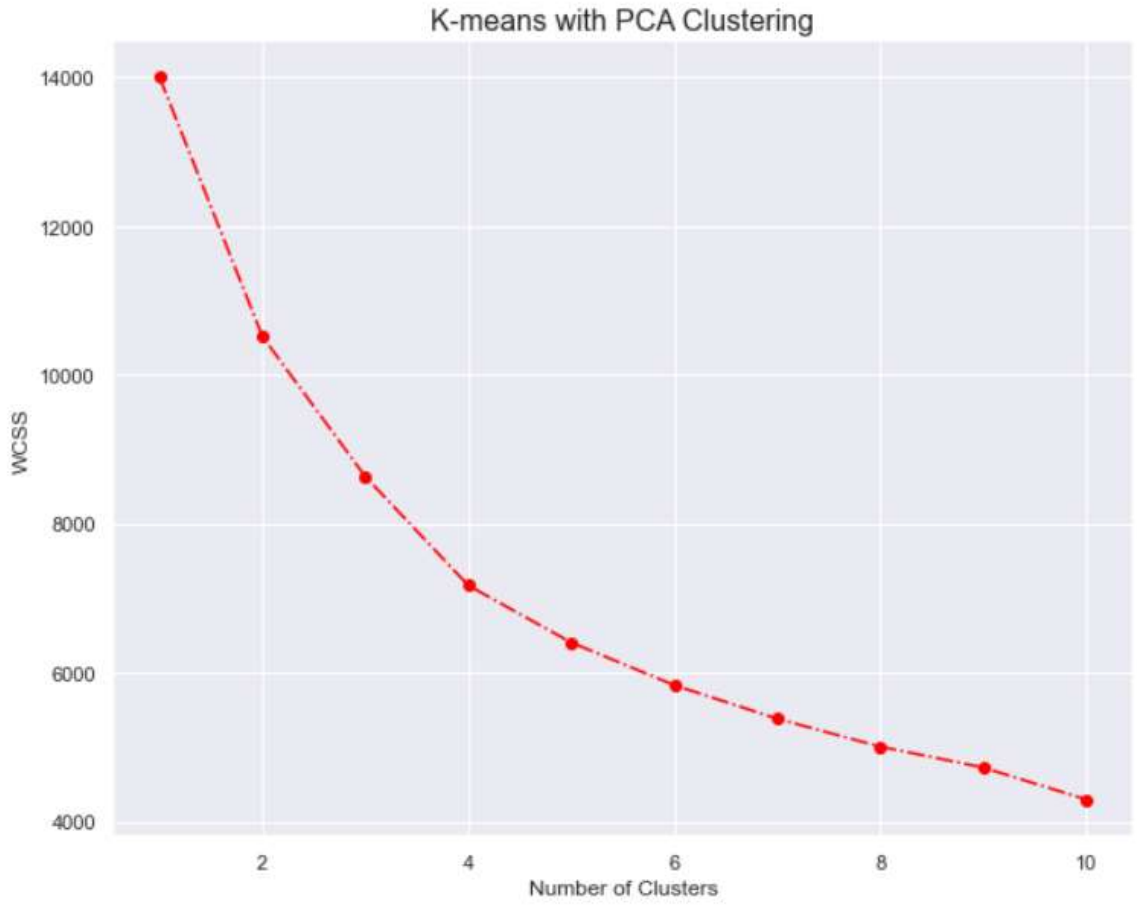
```
array([[ 2.51474593,  0.83412239,  2.1748059 ],
       [ 0.34493528,  0.59814564, -2.21160279],
       [-0.65106267, -0.68009318,  2.2804186 ],
       ...,
       [-1.45229829, -2.23593665,  0.89657125],
       [-2.24145254,  0.62710847, -0.53045631],
       [-1.86688505, -2.45467234,  0.66262172]])
```

K-means clustering con PCA

¡Nuestro nuevo conjunto de datos está listo! Es hora de aplicar K-Means a nuestro *nuevo conjunto de datos* con 3 componentes.

¡Es tan simple como antes! Seguimos los mismos pasos con K-Means estándar.

```
wcss = []
for i in range(1,11):
    kmeans_pca = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans_pca.fit(scores_pca)
    wcss.append(kmeans_pca.inertia_)
```



Vemos que el número de clúster óptimo dentro de la suma del cuadrado es 4.

```
kmeans_pca = KMeans(n_clusters = 4, init = 'k-means++',
random_state = 42)
kmeans_pca.fit(scores_pca)
```

El algoritmo K-Means ha aprendido de nuestros nuevos componentes y ha creado 4 clusters. Me gustaría ver conjuntos de datos antiguos con nuevos componentes y etiquetas.

```
df_segm_pca_kmeans = pd.concat([df.reset_index(drop = True),
pd.DataFrame(scores_pca)], axis =
1)df_segm_pca_kmeans.columns.values[-3: ] = ['Component 1',
'Component 2', 'Component 3']
df_segm_pca_kmeans['Segment K-means PCA'] =
kmeans_pca.labels_df_segm_pca_kmeans.head()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3	Segment K-means PCA
0	0	0	67	2	124670	1	2	2.514746	0.834122	2.174806	3
1	1	1	22	1	150773	1	2	0.344935	0.598146	-2.211603	0
2	0	0	49	1	89210	0	0	-0.651063	-0.680093	2.280419	2
3	0	0	45	1	171565	1	1	1.714316	-0.579927	0.730731	1
4	0	0	53	1	149031	1	1	1.626745	-0.440496	1.244909	1

```
# We calculate the means by segments.
df_segm_pca_kmeans_freq = df_segm_pca_kmeans.groupby(['Segment
K-means PCA']).mean()
df_segm_pca_kmeans_freq
```

Segment K-means PCA	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3
0	0.900289	0.965318	28.878613	1.060694	107551.500000	0.677746	0.440751	-1.107019	0.703776	-0.781410
1	0.027444	0.168095	35.737564	0.734134	141525.826758	1.267581	1.480274	1.372663	-1.046172	-0.248046
2	0.306522	0.095652	35.313043	0.760870	93692.567391	0.252174	0.039130	-1.046406	-0.902963	1.003644
3	0.505660	0.690566	55.679245	2.128302	158019.101887	1.120755	1.101887	1.687328	2.031200	0.844039

Arriba vemos nuestros datos agrupados por segmento K-Means. También podemos convertir los números de segmento a la etiqueta y ver el número de observación y las proporciones de cada segmento por observación total.

```
df_segm_pca_kmeans_freq['N Obs'] = df_segm_pca_kmeans[['Segment
K-means PCA','Sex']].groupby(['Segment K-means PCA']).count()
df_segm_pca_kmeans_freq['Prop Obs'] =
df_segm_pca_kmeans_freq['N Obs'] / df_segm_pca_kmeans_freq['N
Obs'].sum()
df_segm_pca_kmeans_freq =
df_segm_pca_kmeans_freq.rename({0:'standard',
                                1:'career focused',
                                2:'fewer opportunities',
```

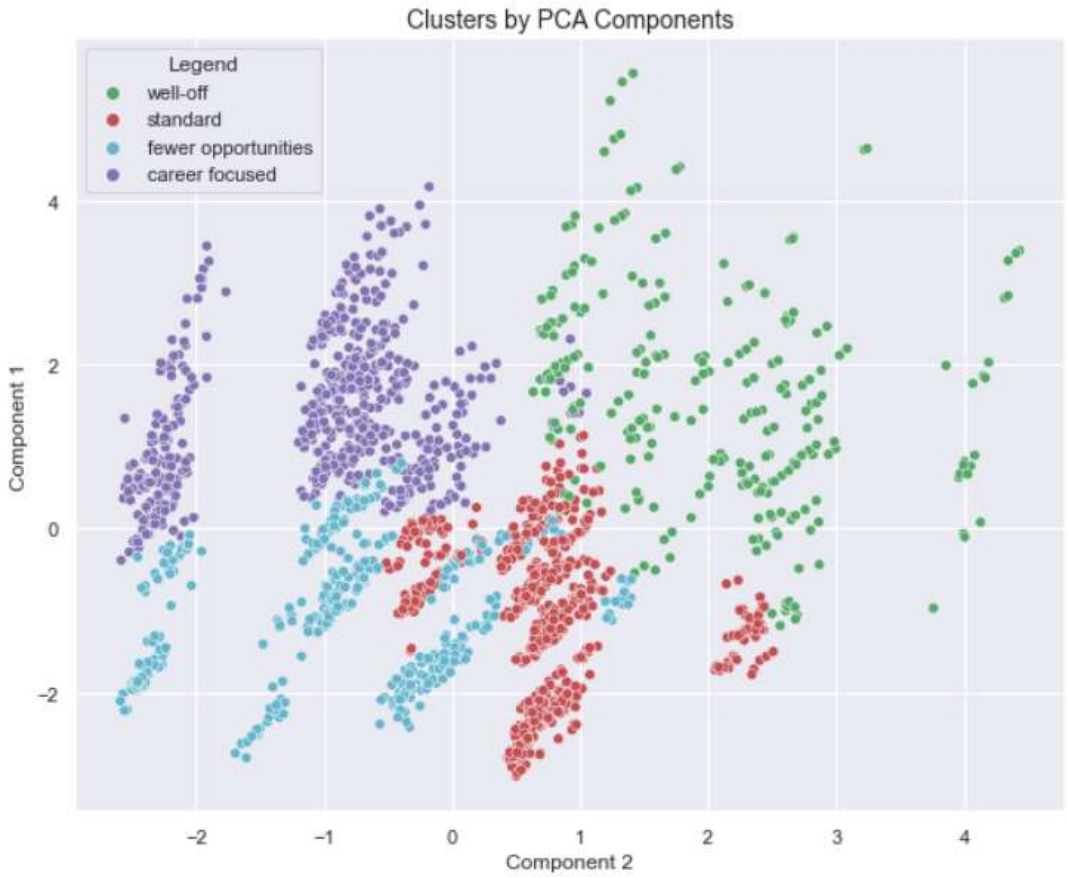
```
                                        3:'well-off'})
df_segm_pca_kmeans_freq
```

Segment K-means PCA	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3	N Obs	Prop Obs
standard	0.900289	0.965318	28.878613	1.060694	107551.500000	0.677746	0.440751	-1.107019	0.703776	-0.781410	692	0.3460
career focused	0.027444	0.168096	35.737564	0.734134	141525.826758	1.267581	1.480274	1.373663	-1.046172	-0.248046	583	0.2915
fewer opportunities	0.306522	0.095652	35.313043	0.760870	93692.567391	0.252174	0.039130	-1.046406	-0.902963	1.003644	460	0.2300
well-off	0.505660	0.690566	55.679245	2.128302	158019.101887	1.120755	1.101887	1.687328	2.031200	0.844039	265	0.1325

Obtuvimos columnas y cambiamos el nombre con unas pocas líneas de códigos. Ahora, vamos a trazar nuestros nuevos segmentos y ver las diferencias.

Como probablemente pueda recordar, nuestros cuatro grupos anteriores eran estándar, centrados en la carrera, menos oportunidades.

```
df_segm_pca_kmeans['Legend'] = df_segm_pca_kmeans['Segment K-
means PCA'].map({0:'standard',
           1:'career focused',
           2:'fewer opportunities',
           3:'well-off'})
x_axis = df_segm_pca_kmeans['Component 2']
y_axis = df_segm_pca_kmeans['Component 1']
plt.figure(figsize = (10, 8))
sns.scatterplot(x_axis, y_axis, hue =
df_segm_pca_kmeans['Legend'], palette = ['g', 'r', 'c', 'm'])
plt.title('Clusters by PCA Components')
plt.show()
```



Cuando trazamos la solución de agrupamiento de medios K sin PCA, solo pudimos distinguir el segmento verde, pero la división basada en los componentes es mucho más pronunciada.

Ese fue uno de los mayores objetivos de PCA para reducir el número de variables combinándolas en otras más grandes.

"No encuentres clientes para tus productos, encuentra productos para tus clientes".

- Seth Godín

Conclusión

Segmentamos a nuestros clientes en 4 grupos. ¡Estamos listos para comenzar a elegir nuestros grupos en función de nuestros objetivos y comercializarlos!

La segmentación ayuda a los especialistas en marketing a ser más eficientes en términos de tiempo, dinero y otros recursos.

Obtienen una mejor comprensión de las necesidades y deseos del cliente y, por lo tanto, pueden adaptar las campañas a los segmentos de clientes con más probabilidades de comprar productos.