ANEXO

23 de octubre de 2021

Practica 4: k-means

Dataset disponible en:
https://www.kaggle.com/hellbuoy/online-retail-customer-clustering

1. Objetivo de la practica.

Usar un dataset de registros de transacciones para identificar a los mejores clientes (los más frecuentes), usando un modelo aprendizaje no supervizado. Esto, debido a que no existen etiquetas que indiquen que tipo de cliente hay en la base de datos.

2. Conceptos.

En la librería Scikit-learn existe el modulo sklearn.cluster.KMeans y tal como su nombre lo indica, es un modulo que ayuda a la creación de modelos de clustering no supervisado.
El dataset consiste en un archivo csv que contiene los registros de transacciones dentro de Reino Unido en un periodo de 01/12/2010 a 09/12/2011.

3. Herramientas a usar.

Computadora con acceso a internet.
Los siguientes programas y librerías:

a. Python versión 3.8.8

b. Anaconda versión 4.10.1

c. Jupyter-lab 3.0.14

d. Pandas versión 1.2.4

e. Numpy versión 1.20.2

f. Scikitlearn versión 0.24.1

g. Matplotlib versión 3.3.4

h. Seaborn versión 0.11.1

4. Desarrollo.

4.1. Entender el problema.

Este problema consiste en identificar cuales son los clientes más fieles (que realiza compras con más frecuencia) de una base de datos. Para esto, se propone usar clustering entre los clientes para identificar grupos cuyos integrantes compartan características y de esta forma identificar el grupo cuya frecuencia de compras sea alta.

4.2. Criterio de evaluación.

En este caso no existe una métrica exacta para evaluar el modelo, debido a la falta de etiquetas, pero se usarán métodos como la inercia y silhouette para medir que tan bien divididos están los clusters.

4.3. Preparar los datos.

a. Importe las librerías necesarias.

```
import os
import path
import zipfile
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
```

b. Descargue los datos del link que se encuentra al inicio de esta práctica.

c. Extraer los datos.
Use el método mostrado en la práctica 0 sección 3 inciso h para extraerlos datos con Python OPCIONAL
Cree una carpeta para extraer ahí los archivos.

d. Cargar los datos.
Cargue el archivo que en este caso se llama "OnlineRetail.csv.zip". El archivo original se llama "archive.zip", pero este se renombró para que fuese más fácil identificarlo.

```
o_retail = pd.DataFrame(pd.read_csv(Path+''OnlineRetail.csv"))
```

Código 1

e. Explorar los datos.
Use head() para ver el contenido de los datos. También use info() para ver si existen registros vacíos. [Ver Figura 1]

```
pozos.head()
pozos.info()
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Figura 1: Función head() y Función info()

En este caso existen registros vacíos en la columna *CustomerID*, pero esta columna no se puede eliminar debido a que este problema se enfoca precisamente en los clientes. Por lo tanto es mejor eliminar los registros que no contengan información de CustomerID. Para esto se usa la siguiente instrucción:

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

Ahora ya se puede trabajar de manera ordenada con los datos.

f. Preprocesamiento de los datos.
Según la información que se puede encontrar en el link proporcionado al

inicio de esta práctica, se dice que el dataset se enfoca en las transacciones hechas en Reino Unido sin embargo, en la Figura 1 se muestra que hay una columna llamada *Country*. Esto hace pensar que hay más de un país considerado, para verificarlo, se hace una gráfica con el número de usuarios que hay por país.

Los países que se incluyen en el dataset son los siguientes: *United Kingdom, Germany, France, EIRE, Spain, Netherlands, Belgium, Switzerland, Portugal, Australia, Norway, Italy, Channel Islands, Finland, Cyprus, Sweden, Unspecified, Austria, Denmark, Japan, Poland, Israel, USA, Hong-Kong, Singapore, Iceland, Canada, Greece, Malta, United Arab Emirates, European Community, RSA, Lebanon, Lithuania, Brazil, Czech Republic, Bahrain y Saudi Arabia*. Sin embargo, en la Figura 2 se muestra que la mayoria se concentran en United Kingdom, por lo tanto, esa columna no se considerará.

```
sns.displot(o_retail["Country"])
```

Código 2

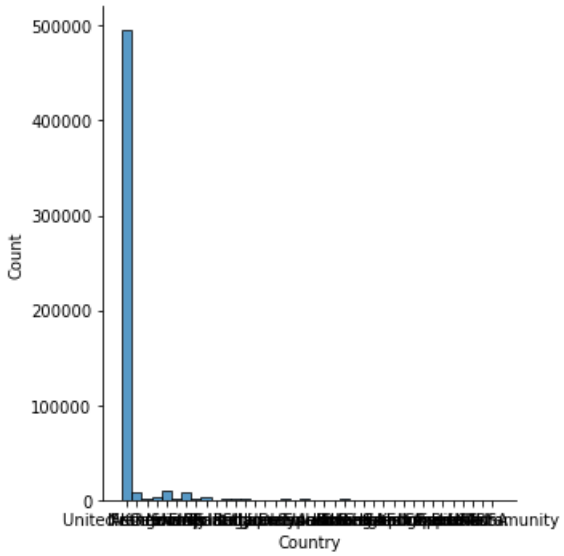

Figura 2: Gráfica de países vs clientes

Existen muchas compras hechas por adelantado, eso quiere decir que al consumidor se le DEBEN entregar n cantidad de artículos, por eso, en la columna *Quantity* hay valores negativos, los cuales hacen referencia a la cantidad de artículos que se deben. [Ver la Figura 3]

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
141	C536379	D	Discount	-1	01-12-2010 09:41	27.50	14527.0	United Kingdom
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	01-12-2010 09:49	4.65	15311.0	United Kingdom
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	01-12-2010 10:24	1.65	17548.0	United Kingdom
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	01-12-2010 10:24	0.29	17548.0	United Kingdom
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	01-12-2010 10:24	0.29	17548.0	United Kingdom

Figura 3: Valores negativos en Quantity

Para quitar eso, se agregará una nueva columna que nos diga si la compra es adelantada o no. Para hacer eso, primero se van a seleccionar las filas cuyos valores en *Quantity* sea negativos y con la siguiente función los transformarán en valores positivos y agregará la columna *Status*.

```
def status(df):
```

```
    #Seleccionamos los valores negativos en Quantity
    df[''Status"] = 1 #''Bought"
    aux = df[df[''Quantity"] < 0]
    for i in aux.index:
        df.loc[i, ''Quantity"] = -1*(df.loc[i, ''Quantity"])
        df.loc[i, ''Status"] = 0 #''Credit"
    return df
    status(o_retail)
```

Código 3

El resultado de esta función se muestra en la Figura 4

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

Figura 4: Nuevos valores en Quantity

El hacer esto ayuda a que los futuros cálculos no se vayan a ver afectados por los números negativos. Como ya no hay números menores a cero, ya es posible seguir con el preprocesamiento de datos.

La columna *CustomerID* tiene datos tipo int, pero no se realizarán cálculos con ella, se transformará en string. Posteriormente, se agrega una nueva columna que se llame *AmountTotal*, la cual se calcula multiplicando las columnas *Quantity* y *UnitPrice*. Esta ayuda a saber el monto total que cada cliente ha gastado en todas sus compras. [Ver la Figura 5]
Lo anterior se logra con el siguiente código:

```
o_retail['CustomerID'] = o_retail['CustomerID'].astype(str)
o_retail['AmountTotal'] = o_retail['Quantity']*o_retail['UnitPrice']
    
```

Código 4

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Status	AmountTotal
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	1	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	1	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34

Figura 5: Columna Amount Total añadida

El dataset incluye una columna llamada *InvoiceDate*, en la cual vienen datos de la fecha en la que se realizó la compra. El problema que presenta esta columna, es que la fecha vienen en formato str y esto hace que sea imposible realizar cálculos con ellos. Para corregir esto, se usará la librería datetime para que la fecha esté en el formato correcto.

Una vez que se hace esto, ahora se obtiene la fecha más reciente del dataset. Esta fecha se usará para contabilizar el número de días de diferencia que hay entre la fecha que hay en la columna *InvoiceDate* y la fecha más reciente. Los resultados se agregarán en una nueva columna llamada *LastInvoce*.[Ver la Figura 6]

El siguiente código realiza el procedimiento anteriormente mencionado.

```
o_retail['InvoiceDate'] = pd.to_datetime(o_retail['InvoiceDate'],
format='%d-%m-%Y %H:%M')
date = o_retail['InvoiceDate'].max()
o_retail['LastInvoce'] = date - o_retail['InvoiceDate']
o_retail['LastInvoce'] = o_retail['LastInvoce'].dt.days
```

Código 5

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Status	AmountTotal	LastInvoce
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	1	15.30	373
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34	373
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	1	22.00	373
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34	373
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	1	20.34	373

Figura 6: Columna LastInvoice muestra hace cuanto tiempo se realizó la compra (en días)

g. Dividir los datos en Train y test (X y y).

Como este es un problema de aprendizaje no supervisado, no se puede dividir los datos en train y test ya que el modelo no puede entrenar de esta manera. En cambio, se van a seleccionar columnas clave para realizar el aprendizaje.
En este caso se toman las columnas *InvoiceNo*, *AmountTotal* y *LastInvoce*, ya que estas contienen los datos numéricos de los pagos que se han hecho, la cantidad del pago y su frecuencia. Estas columnas se van a agrupar por *CustomerID*, de esta forma se tiene un DataFrame en el cual los consumidores son las filas y en las columnas está la información de sus compras.

Para crear este nuevo DataFrame, se usa el siguiente código

```
inv = pd.DataFrame(o_retail.groupby('CustomerID')['InvoiceNo'].count())
amount = pd.DataFrame(o_retail.groupby('CustomerID')['AmountTotal'].sum())
last = pd.DataFrame(o_retail.groupby('CustomerID')['LastInvoce'].min())

new_retail = pd.concat([inv, amount, last], axis=1)
```

Código 6

El nuevo DataFrame se muestra en la Figura 7

CustomerID	InvoiceNo	AmountTotal	LastInvoce
0.0	135080	2.018623e+06	0
12346.0	2	1.543672e+05	325
12347.0	182	4.310000e+03	1
12348.0	31	1.797240e+03	74
12349.0	73	1.757550e+03	18

Figura 7: Nuevo DF que se usará para entrenar el modelo

Ahora ya está listo para seguir trabajando con los datos finales que se usarán en el modelo de aprendizaje no supervisado.

h. Codificar y Escalar los datos en X.
Al igual que en la Práctica 2, se propone usar dos métodos para escalar los datos, MinMaxScaler y StandardScaler.

La siguiente función tiene como objetivo crear nuevos DataFrames los cuales contengan los datos ya escalados usando MinMaxScaler y StandardScaler.

```
scaler = MinMaxScaler()
scalerSS = StandardScaler()

def escalar(df, scal):
    ind = df.index
    retail_scaler = pd.DataFrame(scal.fit_transform(df), index=ind)
    retail_scaler.columns = df.columns
    return retail_scaler

df_MMS = escalar(new_retail, scaler)
df_SS = escalar(new_retail, scalerSS)
```

Código 7

Ahora ya se puede construir el modelo de k-means y pasarle los datos que se tienen en df_MMS y df_SS.

4.4. Construir el modelo.

En este caso se van a crear varios modelos, y se realizaran sus respectivas predicciones.

Usando la siguiente función, se puede hacer de manera más fácil el entrenamiento y la predicción, ya que solo se le pasa el modelo y el dataset.

```
def prediccion(df, model):
    model.fit(df)
    print(model.predict(df))
    print("Inercia = ", model.inertia_)
    pred = model.predict(df)
    return pred
```

Código 8

Ahora se aplica el Código 8 usando un modelo de k-means con *n_clusters*=2 por cada Dataset escalado

```
kmeansMMS = KMeans(n_clusters=2, init='k-means++')
predMMS = prediccion(df_MMS, kmeansMMS)

kmeansSS = KMeans(n_clusters=2, init='k-means++')
predSS = prediccion(df_SS, kmeansSS)
```

Código 9

Los resultados de cada modelo son los siguientes y se muestran en respectivo orden:
Inercia = 67.68068542075144
Inercia = 4891.799479100922

Si bien estos resultados no son malos, la inercia solo indica que tan bien separados están los clusters.
Sin embargo, existe una métrica más aceptada en ML llamada el método Silhouette, rl cual también se va a aplicar a cada modelo creado. En el caso de los primero dos modelos, se tiene el siguiente código.

```
silhouette_MMS = silhouette_score(df_MMS, kmeansMMS.labels_,
metric='euclidean')
print(''Silhouette =", silhouette_MMS)

silhouette_SS = silhouette_score(df_SS, kmeansSS.labels_,
metric='euclidean')
print(''Silhouette =", silhouette_SS)
```

Código 10

El resultado se muestra a continuación.
Silhouette = 0.7383086323753945
Silhouette = 0.8113704009700877

Este resultado es muy cercano a 1, es decir que tiene buena separación entre clusters, pero eso se debe a que solo hay 2 clusters, por lo tanto el modelo solo "adivina.ª que cluster debe poner los datos.

4.5. Análisis de errores.

Los modelos anteriores solamente contaban con 2 clusters, ahora se debe buscar el número óptimo de clusters para cada dataset escalado. Esto se logra a través del método del codo, en el cual se grafica la inercia del modelo usando diferentes números de clusters.

Lo anterior se puede implementar con el siguiente código.

```
def buscar_k(df):
    SSE = []
    for cluster in range(1,20):
        kmeans = KMeans(n_clusters = cluster, init='k-means++')
        kmeans.fit(df)
        SSE.append(kmeans.inertia_)
    porc = 1
    for i in range(len(SSE)-1):
        dif = (SSE[i] - SSE[i+1])
```

```
        print('Punto',i+1,'a',i+2, dif/porc)
        porc = dif
        #print(dif, porc)
    plt.plot(range(1,20), SSE, 'o-')
```

Código 11

Para el caso del dataset *df_MMS* se tiene el siguiente resultado [Ver la Figura 8]

```
Punto 1 a 2 253.50295639670952
Punto 2 a 3 0.1363588769784553
Punto 3 a 4 0.3870901859675165
Punto 4 a 5 0.502706476735322
Punto 5 a 6 0.49069287691289304
Punto 6 a 7 0.6068910795738045
Punto 7 a 8 0.9235484149803169
Punto 8 a 9 0.744713443976982
Punto 9 a 10 0.7583704513911387
Punto 10 a 11 0.6688252582536596
Punto 11 a 12 0.7274044196677418
Punto 12 a 13 0.6583638422756075
Punto 13 a 14 0.8735324967938539
Punto 14 a 15 0.6437959478056088
Punto 15 a 16 0.9530647105747421
Punto 16 a 17 0.7373198727729307
Punto 17 a 18 0.869682384856029
Punto 18 a 19 0.8229602992308461
```

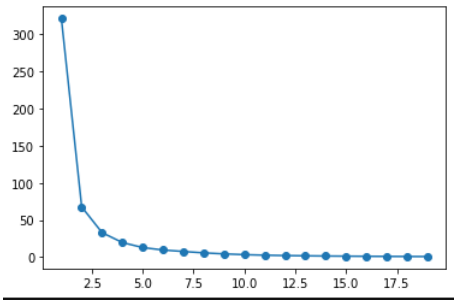

Figura 8: Distancia entre cada punto y gráfica de inercias de datos escalados con MinMaxScaler

Y en el caso del dataset *df_SS* se tiene el siguiente resultado [Ver la Figura 9]

```
Punto 1 a 2 8227.200520899114
Punto 2 a 3 0.42273016841899774
Punto 3 a 4 0.1366915390870341
Punto 4 a 5 0.6599224535685444
Punto 5 a 6 0.5910019150331535
Punto 6 a 7 0.5004578590657681
Punto 7 a 8 0.8427047562118207
Punto 8 a 9 0.5803274441636229
Punto 9 a 10 0.5394597338721148
Punto 10 a 11 0.8602875461136832
Punto 11 a 12 1.0646311208570927
Punto 12 a 13 0.7981447488301132
Punto 13 a 14 0.869304321226092
Punto 14 a 15 0.9215056176147315
Punto 15 a 16 0.5309031563692295
Punto 16 a 17 1.263309525637219
Punto 17 a 18 0.9194594463960396
Punto 18 a 19 0.7141956490575219
```

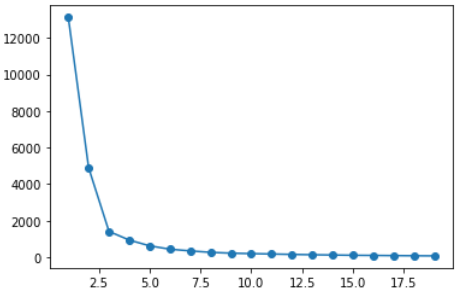

Figura 9: Distancia entre cada punto y gráfica de inercias de datos escalados con StandardScaler

Como se puede ver en el caso de *df_MMS* el número de cluster óptimo se encuentra entre 4 y 5, mientras que en *df_SS* es entre 8 y 9. Para comprobar si

son números óptimos, se hacen modelos nuevos con *n_clusters*= 5 y 8, respectivamente.
Use el Código 9 y el Código 10, cambiando los valores de *n_clusters* por los valores mostrados anteriormente, para crear nuevos modelos y obtener sus respectivas métricas.

Los resultados de inercia y silhouette para *df_MMS* son los siguientes:
Inercia = 12.998283407254766
Silhouette = 0.6143242841711527

Y para *df_SS* son los siguientes:
Inercia = 268.48616907818297
Silhouette = 0.5537561343887281

Ahora bien, se pueden graficar los datos, para comprobar visualmente si los clusters están bien separados o no.
El siguiente código genera 3 gráficas, una por cada columna en el DataFrame escalado. Compara cada columnas con las otras dos restantes.

```
def resutados(df, pred):
    df_result = df.copy()
    df_result['cluster'] = pred
    at = df_result[df_result["AmountTotal"] < 75000]
    k = at['cluster'].nunique()
    fig, axes = plt.subplots(3, 1,figsize=(16,12))
    axes[0].set_title("AmountTotal vs LastInvoce")
    axes[1].set_title("AmountTotal vs InvoiceNo")
    axes[2].set_title("LastInvoce vs InvoiceNo")

    sns.scatterplot(at["AmountTotal"], at["LastInvoce"], hue = at["cluster"],
    palette = sns.color_palette('hls',k), ax=axes[0])
    sns.scatterplot(at["AmountTotal"], at["InvoiceNo"], hue = at["cluster"],
    palette = sns.color_palette('hls',k), ax=axes[1])
    sns.scatterplot(at["LastInvoce"], at["InvoiceNo"], hue = at["cluster"],
    palette = sns.color_palette('hls',k), ax=axes[2])
```

Los resultados de *df_MMS* y *df_SS* se muestran en las Figuras 10 y 11

4.6. Implementación

Ver el Notebook de Jupyter llamado *P4-k-means.ipynb*

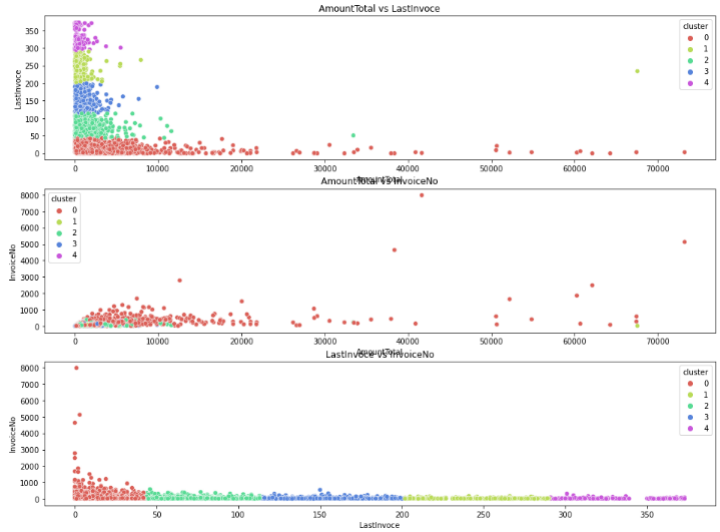

Figura 10: Gráfica de clusters usando datos escalados con MinMaxScaler

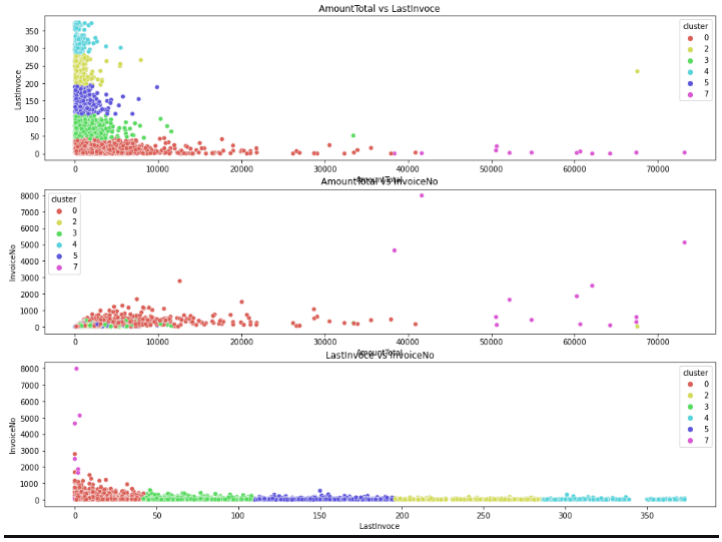

Figura 11: Gráfica de clusters usando datos escalados con StandardScaler