Completo tutorial de machine learning con PySpark

Ahora que ya tienes PySpark en funcionamiento, te mostraremos cómo ejecutar un proyecto de segmentación de clientes de principio a fin utilizando la biblioteca.

La segmentación de clientes es una técnica de marketing que utilizan las empresas para identificar y agrupar a los usuarios que presentan características similares. Por ejemplo, si visitas Starbucks solo en verano para comprar bebidas frías, se te puede segmentar como "comprador de temporada" y pueden seducirte con promociones especiales diseñadas para la época estival.

Los científicos de datos suelen crear algoritmos de machine learning no supervisado como K-medias o agrupamiento jerárquico para realizar la segmentación de clientes. Estos modelos son excelentes para identificar patrones similares entre grupos de usuarios que a menudo pasan desapercibidos al ojo humano.

En este tutorial, utilizaremos K-medias para realizar la segmentación de clientes en el conjunto de datos de comercio electrónico que descargamos anteriormente.

Al final de este tutorial, estarás familiarizado con los siguientes conceptos:

- Leer archivos .csv con PySpark
- Análisis exploratorio de datos con PySpark
- Agrupar y ordenar datos
- Realizar operaciones aritméticas
- Agregar conjuntos de datos
- Preprocesamiento de datos con PySpark
- Trabajar con valores fecha-hora
- Conversión de tipo
- Unir dos marcos de datos
- La función rank()
- Machine learning con PySpark
- Crear un vector de características
- Estandarizar los datos
- Crear un modelo K-medias
- Interpretar el modelo

Paso 1: Crear una SparkSession

Una SparkSession es un punto de entrada a toda la funcionalidad de Spark, y es necesaria si quieres crear un marco de datos en PySpark. Ejecuta las siguientes líneas de código para inicializar una SparkSession:

```
spark = SparkSession.builder.appName("Datacamp Pyspark Tutorial").config("spark.memory.offHeap.enabled","true").config("spark.memory.offHeap.size","10g").getOrCreate()
```

Este código crea un objeto SparkSession llamado "spark" utilizando el patrón de diseño builder de PySpark. El parámetro appName establece el nombre de la aplicación en "Datacamp Pyspark Tutorial". Los dos parámetros config activan la memoria off-heap y establecen el tamaño de la memoria off-heap en 10 gigabytes. Por último, el método getOrCreate() devuelve una SparkSession existente o crea una nueva si no existe ninguna.

Utilizando los códigos anteriores, creamos una spark session y establecemos un nombre para la aplicación. A continuación, los datos se almacenaban en caché en la memoria off-heap para evitar almacenarlos directamente en el disco, y se especificaba manualmente la cantidad de memoria.

Paso 2: Crear el DataFrame

Ahora podemos leer el conjunto de datos que acabamos de descargar:

```
df = spark.read.csv('datacamp_ecommerce.csv',header=True,escape="\"")
```

Este código lee un archivo CSV llamado "datacamp_ecommerce.csv" en un DataFrame de Spark llamado "df". El argumento header=True indica que la primera fila del archivo CSV contiene los nombres de las columnas. El argumento escape="\"" especifica que se utilice la barra invertida como carácter de escape para cualquier carácter especial del archivo CSV. Se supone que el objeto spark es un objeto SparkSession creado previamente.

Ten en cuenta que hemos definido un carácter de escape para evitar comas en el archivo .csv al analizarlo.

Echemos un vistazo a la cabecera del dataframe utilizando la función show():

```
df.show(5,0)
```

Este código está escrito en Python y utiliza el método show() para mostrar las 5 primeras filas de un DataFrame df. El segundo argumento, 0, especifica que el método no debe truncar las columnas mostradas. Esto es útil cuando se trabaja con grandes conjuntos de datos en los que el comportamiento por defecto es truncar las columnas para que quepan en la pantalla. Si estableces el segundo argumento en 0, se mostrarán todas las columnas sin truncar.

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	1/12/2010 8:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	1/12/2010 8:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	1/12/2010 8:26	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	1/12/2010 8:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	1/12/2010 8:26	3.39	17850	United Kingdom

El dataframe consta de 8 variables:

1. InvoiceNo: El identificador único de cada factura de cliente.
2. StockCode: El identificador único de cada artículo en stock.
3. Description: El artículo comprado por el cliente.
4. Quantity: La cantidad de cada artículo comprada por un cliente en una sola factura.
5. InvoiceDate: La fecha de compra.
6. UnitPrice: Precio de una unidad de cada artículo.
7. CustomerID: Identificador único asignado a cada usuario.
8. Country: El país desde el que se realizó la compra

Paso 3: Análisis exploratorio de datos

Ahora que hemos visto las variables presentes en este conjunto de datos, vamos a realizar algunos análisis exploratorios de datos para comprender mejor estos puntos de datos:

1. Empecemos contando el número de filas del dataframe:

```
df.count()  # Answer: 2,500
```

Este código está escrito en Python. La función df.count() se utiliza para contar el número de valores no nulos de cada columna de un DataFrame df. En este caso, devuelve el recuento de valores no nulos en todas las columnas de df, que es 2500.

1. ¿Cuántos clientes únicos hay en el dataframe?

```
df.select('CustomerID').distinct().count() # Answer: 95
```

Este código está escrito en Python y utiliza la biblioteca PySpark. El código seleccion la columna 'CustomerID' de un DataFrame de PySpark llamado "df". A continuación, aplica el método "distinct" para obtener solo los valores únicos de "CustomerID". Por último, utiliza el método "count" para contar el número de valores únicos. El resultado de este código es el número de valores únicos de "CustomerID" en el DataFrame "df", que es 95.

1. ¿De qué país proceden la mayoría de las compras?

Para buscar el país desde el que se realizan la mayoría de las compras, tenemos que utilizar la cláusula groupBy() de PySpark:

```
from pyspark.sql.functions import *

from pyspark.sql.types import *

df.groupBy('Country').agg(countDistinct('CustomerID').alias('country_count')).show()
```

Este código está escrito en Python y utiliza la biblioteca PySpark para trabajar con big data. La primera línea importa dos módulos de la biblioteca PySpark: pyspark.sql.functions y pyspark.sql.types. Estos módulos contienen funciones y tipos de datos que se utilizan para trabajar con datos en PySpark. La segunda línea de código utiliza el método groupBy() para agrupar los datos del DataFrame df por la columna Country. A continuación, se utiliza el método agg() para aplicar una función de agregación a cada grupo. En este caso, se utiliza la función countDistinct() para contar el número de valores distintos de CustomerID en cada grupo. El método alias() se utiliza para cambiar el nombre de la columna resultante a country_count. Por último, se utiliza el método show() para mostrar el DataFrame resultante en formato tabular. Esto mostrará el número de clientes distintos de cada país en la columna Country del DataFrame original.

Tras ejecutar los códigos anteriores, aparecerá la siguiente tabla:

Country	country_count
Germany	2
France	1
EIRE	1
Norway	1
Australia	1
United Kingdom	88
Netherlands	1

Casi todas las compras de la plataforma se hicieron desde el Reino Unido, y solo unas pocas se hicieron desde países como Alemania, Australia y Francia.

Observa que los datos de la tabla anterior no se presentan por orden de compra. Para ordenar esta tabla, podemos incluir la cláusula orderBy():

```
df.groupBy('Country').agg(countDistinct('CustomerID').alias('country_count')).orderBy(desc('country_count')).show()
```

Este código está escrito en Python y utiliza la biblioteca PySpark. El código realiza una operación de agrupar por en un DataFrame llamado df basándose en la columna "Country". A continuación, agrega el número de valores distintos de "CustomerID" de cada país utilizando la función countDistinct. La columna resultante se renombra como "country_count" utilizando la función alias. Por último, el DataFrame resultante se ordena en orden descendente basándose en la columna "country_count" mediante la función orderBy y se muestra mediante la función show. En general, este código cuenta el número de clientes únicos de cada país en el DataFrame y muestra los resultados en orden descendente.

La salida mostrada se ordena ahora en orden descendente:

```
+--------------+-------------+
|       Country|country_count|
+--------------+-------------+
|United Kingdom|           88|
|       Germany|            2|
|        France|            1|
|          EIRE|            1|
|     Australia|            1|
|        Norway|            1|
|   Netherlands|            1|
+--------------+-------------+
```

1. ¿Cuándo fue la compra más reciente realizada por un cliente en la plataforma de comercio electrónico?

   Para averiguar cuándo se realizó la última compra en la plataforma, tenemos que convertir la columna "InvoiceDate" a un formato de marca de tiempo y utilizar la función max() de Pyspark:

```
spark.sql("set spark.sql.legacy.timeParserPolicy=LEGACY")

df = df.withColumn('date',to_timestamp("InvoiceDate", 'yy/MM/dd HH:mm'))

df.select(max("date")).show()
```

   Este código está escrito en Python y utiliza el módulo SQL de Apache Spark para manipular los datos. La primera línea establece la propiedad de configuración spark.sql.legacy.timeParserPolicy en LEGACY. Esta propiedad determina cómo analiza Spark las marcas de tiempo en las consultas SQL. Si se estableces enLEGACY, Spark utilizará el antiguo comportamiento de análisis de marcas de tiempo, que es más indulgente y puede manejar una gama más amplia de formatos de marcas de tiempo. La segunda línea crea una nueva columna llamada date en el DataFrame df. Los valores de esta columna se generan aplicando la función to_timestamp a la columna InvoiceDate de df. El segundo argumento de to_timestamp especifica el formato de la cadena de marca de tiempo en InvoiceDate. En este caso, el formato es yy/MM/dd HH:mm, lo que significa que la cadena de marca de tiempo debe tener un año de dos dígitos (yy), seguido de una barra (/), luego un mes de dos dígitos (MM), otra barra,

un día de dos dígitos (dd), un espacio, una hora de dos dígitos (HH) y dos puntos (:), y finalmente un minuto de dos dígitos (mm). La tercera línea selecciona el valor máximo de la columna date y lo muestra mediante el método show. Esta línea busca la última fecha de la columna InvoiceDate y la muestra en un formato legible para el ser humano.

Deberías ver la siguiente tabla después de ejecutar el código anterior:

```
+-------------------+
|          max(date)|
+-------------------+
|2012-01-10 17:06:00|
+-------------------+
```

1. ¿Cuándo fue la primera compra realizada por un cliente en la plataforma de comercio electrónico?

   De forma similar a lo que hicimos antes, se puede utilizar la función min() para buscar la fecha y hora de compra más tempranas:

```
df.select(min("date")).show()
```

Este código está escrito en Python y utiliza la biblioteca PySpark. Se supone que la variable df es un objeto DataFrame. El método select() se utiliza para seleccionar una columna o un conjunto de columnas específicos del DataFrame. En este caso, se utiliza la función min() para buscar el valor mínimo de la columna "date". A continuación, se utiliza el método show() para mostrar el resultado de la operación. Esto imprimirá el valor mínimo de la columna "date" en la consola.

```
+-------------------+
|          min(date)|
+-------------------+
|2012-01-10 08:26:00|
+-------------------+
```

Observa que las compras más recientes y las más tempranas se hicieron el mismo día, con solo unas horas de diferencia. Esto significa que el conjunto de datos que descargamos solo contiene información de las compras realizadas en un solo día.

Paso 4: Preprocesamiento de datos

Ahora que hemos analizado el conjunto de datos y tenemos una mejor comprensión de cada punto de datos, tenemos que preparar los datos para introducirlos en el algoritmo de machine learning.

Echemos un vistazo a la cabecera del dataframe una vez más para entender cómo se realizará el preprocesamiento:

```
df.show(5,0)
```

Este código está escrito en Python y utiliza el método show() para mostrar las 5 primeras filas de un DataFrame df. El segundo argumento, 0, especifica que el método no debe truncar las columnas mostradas. Esto es útil cuando se trabaja con grandes conjuntos de datos en los que el comportamiento por defecto es truncar las columnas para que quepan en la pantalla. Si estableces el segundo argumento en 0, se mostrarán todas las columnas sin truncar.

```
+---------+---------+--------------------------------------+--------+--
|InvoiceNo|StockCode|Description                           |Quantity|In
+---------+---------+--------------------------------------+--------+--
|536365   |85123A   |WHITE HANGING HEART T-LIGHT HOLDER |6       |1/
|536365   |71053    |WHITE METAL LANTERN                   |6       |1/
|536365   |84406B   |CREAM CUPID HEARTS COAT HANGER        |8       |1/
|536365   |84029G   |KNITTED UNION FLAG HOT WATER BOTTLE|6       |1/
|536365   |84029E   |RED WOOLLY HOTTIE WHITE HEART.        |6       |1/
+---------+---------+--------------------------------------+--------+--
only showing top 5 rows
```

A partir del conjunto de datos anterior, tenemos que crear varios segmentos de clientes en función del comportamiento de compra de cada usuario.

Las variables de este conjunto de datos están en un formato que no se puede ingerir fácilmente en el modelo de segmentación de clientes. Estas características por separado no nos dicen mucho sobre el comportamiento de compra del cliente.

Por ello, utilizaremos las variables existentes para derivar tres nuevas características informativas: reciente, frecuencia y valor monetario (RFM).

El RFM se utiliza habitualmente en marketing para evaluar el valor de un cliente en función de su:

1. Reciente: ¿Hace cuánto que ha comprado cada cliente?
2. Frecuencia: ¿Con qué frecuencia ha comprado algo?
3. Valor monetario: ¿Cuánto dinero gasta por término medio cuando hace compras?

Ahora preprocesaremos el dataframe para crear las variables anteriores.

Reciente

En primer lugar, calculemos el valor de reciente: la última fecha y hora en que se realizó una compra en la plataforma. Esto puede conseguirse en dos pasos:

i) Asigna una puntuación de reciente a cada cliente

Restaremos todas las fechas del dataframe a la fecha más antigua. Esto nos dirá lo recientemente que se ha visto a un cliente en el dataframe. Un valor 0 indica lo menos reciente, ya que se asignará a la persona que hizo una compra en la fecha más temprana.

```
df = df.withColumn("from_date", lit("12/1/10 08:26"))
df = df.withColumn('from_date',to_timestamp("from_date", 'yy/MM/dd HH:mm'))

df2=df.withColumn('from_date',to_timestamp(col('from_date'))).withColumn('recency',col("date").cast("long") - col('from_date').cast("long"))
```

Este código está escrito en Python y utiliza la biblioteca PySpark para el procesamiento de datos. La primera línea crea una nueva columna llamada "from_date" en el DataFrame "df" de PySpark y establece todos los valores de esta columna en la cadena "12/1/10 08:26". La segunda línea convierte la columna "from_date" de una cadena a un formato de marca de tiempo utilizando la función "to_timestamp". El segundo argumento de esta función especifica el formato de la cadena de entrada. La tercera línea crea un nuevo DataFrame llamado "df2" aplicando dos transformaciones al DataFrame original "df". La primera transformación convierte la columna "from_date" a un formato de marca de tiempo utilizando la función "to_timestamp" sin argumentos. La segunda transformación crea una nueva columna llamada "recency" restando la marca de tiempo Unix de la columna "from_date" a la marca de tiempo Unix de la columna "date". La función "col" se utiliza para hacer referencia a columnas del DataFrame. En general, este fragmento de código se utiliza para convertir una columna de cadena a un formato de marca de tiempo y calcular la diferencia de tiempo entre dos columnas de un DataFrame PySpark.

ii) Selecciona la compra más reciente

Un cliente puede hacer varias compras en distintos momentos. Tenemos que seleccionar solo la última vez que compró un producto, ya que esto es indicativo de cuándo se hizo la compra más reciente:

```
df2= =
df2.join(df2.groupBy('CustomerID').agg(max('recency').alias('recency')),on='recency',how='leftsemi')
```

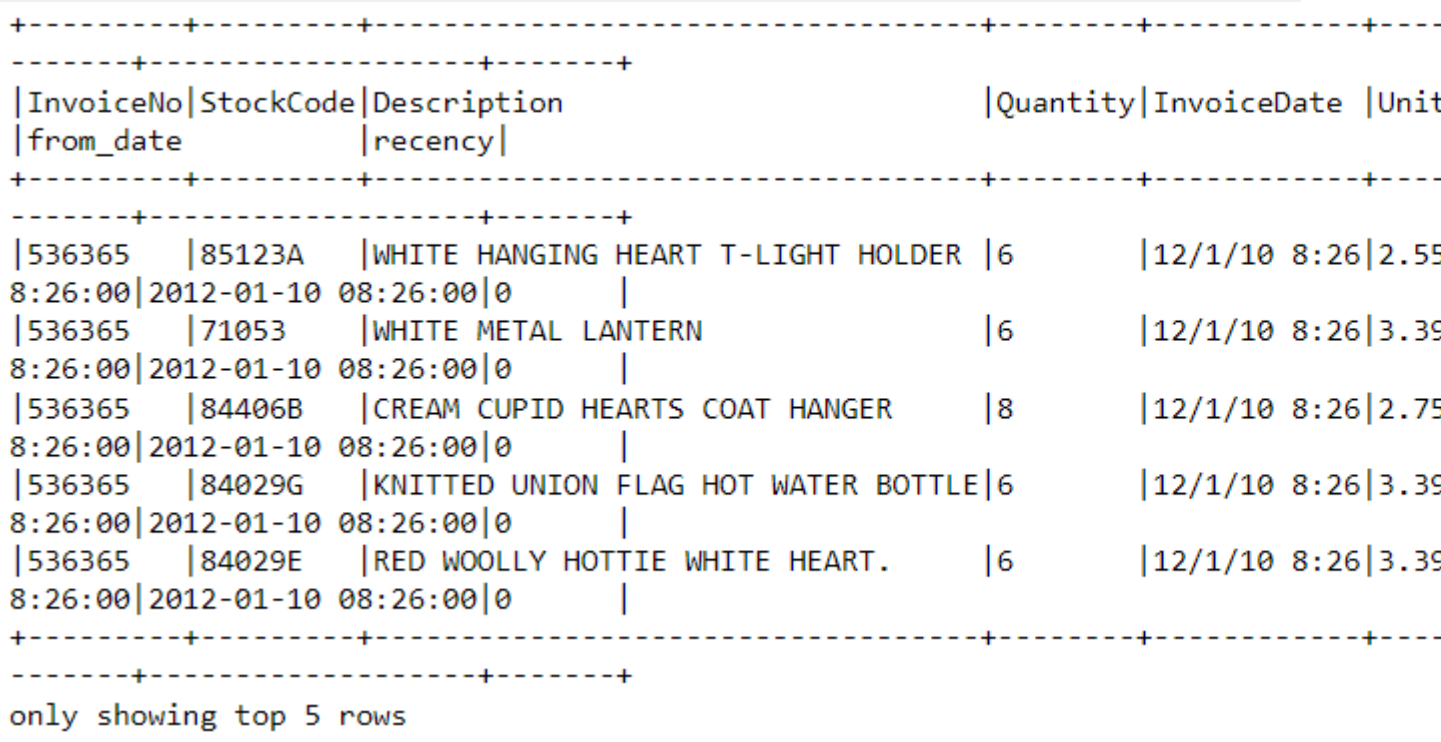
Este código está escrito en Python y utiliza la biblioteca PySpark. El código une el DataFrame df2 con el resultado de una operación groupBy sobre la columna "CustomerID". La operación groupBy busca el valor máximo de la columna "recency" para cada "CustomerID" único. La función agg se utiliza para aplicar la función max a la columna "recency" y asignar a la columna resultante el alias "recency". A continuación se utiliza la función join para unir df2 con el resultado de la operación groupBy. El parámetro on especifica que la unión debe realizarse en la columna "recency". El parámetro how se establece en leftsemi, lo que significa que solo se incluirán en el DataFrame final las filas de df2 que tengan un valor de "recency" coincidente en el resultado de la operación groupBy. En general, este código busca el valor de "recency" más reciente para cada "CustomerID" único y luego une esa información de nuevo al DataFrame original.

Veamos la cabecera del nuevo dataframe. Ahora tiene añadida una variable llamada "recency":

```
df2.show(5,0)
```

Este código está escrito en Python y utiliza el método show() para mostrar las 5 primeras filas del DataFrame df2. El segundo argumento, 0, especifica que no se debe aplicar ningún truncamiento a las columnas mostradas. Esto resulta útil cuando se trabaja con grandes conjuntos de datos en los

que los nombres o valores de las columnas pueden ser demasiado largos para mostrarse en su totalidad.



```
+---------+---------+-----------------------------------+--------+-----------+----
-------+-------------------+-------+
|InvoiceNo|StockCode|Description                        |Quantity|InvoiceDate |Unit
|from_date          |recency|
+---------+---------+-----------------------------------+--------+-----------+----
-------+-------------------+-------+
|536365   |85123A   |WHITE HANGING HEART T-LIGHT HOLDER |6       |12/1/10 8:26|2.55
8:26:00|2012-01-10 08:26:00|0      |
|536365   |71053    |WHITE METAL LANTERN                |6       |12/1/10 8:26|3.39
8:26:00|2012-01-10 08:26:00|0      |
|536365   |84406B   |CREAM CUPID HEARTS COAT HANGER     |8       |12/1/10 8:26|2.75
8:26:00|2012-01-10 08:26:00|0      |
|536365   |84029G   |KNITTED UNION FLAG HOT WATER BOTTLE|6       |12/1/10 8:26|3.39
8:26:00|2012-01-10 08:26:00|0      |
|536365   |84029E   |RED WOOLLY HOTTIE WHITE HEART.     |6       |12/1/10 8:26|3.39
8:26:00|2012-01-10 08:26:00|0      |
+---------+---------+-----------------------------------+--------+-----------+----
-------+-------------------+-------+
only showing top 5 rows
```

Una forma más sencilla de ver todas las variables presentes en un dataframe PySpark es utilizar su función printSchema(). Es el equivalente de la función info() en Pandas:

```
df2.printSchema()
```

Este código está escrito en Python e invoca el método printSchema() del objeto df2. El método printSchema() es una función proporcionada por la API Apache Spark DataFrame que imprime el esquema del DataFrame en la consola. El esquema de un DataFrame describe la estructura de los datos, incluidos los nombres y tipos de datos de cada columna. Esto puede ser útil para comprender los datos y para depurar cualquier problema del DataFrame.

El resultado obtenido debería tener este aspecto:

```
root
 |-- recency: long (nullable = true)
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: string (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: string (nullable = true)
 |-- CustomerID: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- from_date: timestamp (nullable = true)
```

Frecuencia

Calculemos ahora el valor de la frecuencia: la frecuencia con la que un cliente compró algo en la plataforma. Para ello, solo tenemos que agrupar por cada ID de cliente y contar el número de artículos que ha comprado:

```
df_freq = df2.groupBy('CustomerID').agg(count('InvoiceDate').alias('frequency'))
```

Este código está escrito en Python y utiliza la biblioteca PySpark. El código crea un nuevo DataFrame llamado df_freq agrupando las filas de un DataFrame existente df2 por los valores de la columna CustomerID. El método groupBy se utiliza para agrupar las filas por la columna especificada. A continuación, se utiliza el método agg para agregar los datos de cada grupo. En este caso, se utiliza la función count para contar el número de filas de cada grupo que tienen un valor no nulo en la columna InvoiceDate. El método alias se utiliza para cambiar el nombre de la columna resultante por frequency. Así, el DataFrame df_freq resultante tendrá una fila por cada valor único de la columna CustomerID, y la columna frequency contendrá el número de valores no nulos de la columna InvoiceDate de cada grupo.

Mira la cabecera de este nuevo dataframe que acabamos de crear:

```
df_freq.show(5,0)
```

Este código está escrito en Python y utiliza el método show() para mostrar las 5 primeras filas de un DataFrame llamado df_freq. El segundo argumento, 0, especifica que no se debe aplicar ningún truncamiento a las columnas mostradas.

```
+----------+---------+
|CustomerID|frequency|
+----------+---------+
|16250     |14       |
|15100     |1        |
|13065     |14       |
|12838     |59       |
|15350     |5        |
+----------+---------+
only showing top 5 rows
```

A cada cliente del dataframe se le añade un valor de frecuencia. Este nuevo dataframe solo tiene dos columnas, y tenemos que unirlo con el anterior:

```
df3 = df2.join(df_freq,on='CustomerID',how='inner')
```

Este código está escrito en Python y une dos dataframes, df2 y df_freq, en la columna CustomerID mediante una unión interna. El dataframe resultante se asigna a la nueva variable df3. Una unión interna devuelve solo las filas que tienen valores coincidentes en ambos dataframes en la columna especificada. La función join() es un método del objeto dataframe de la biblioteca pandas y toma los siguientes parámetros: - df_freq: el dataframe que se debe unir con df2 - on: el nombre de la columna que se debe unir - how: el tipo de unión que se debe realizar; en este caso, una unión interna. En general, este código se utiliza para combinar dos dataframes basados en una columna común y mantener solo las filas que tienen valores coincidentes en ambos dataframes.

Vamos a imprimir el esquema de este dataframe:

```
df3.printSchema()
```

Este código está escrito en Python e invoca el método printSchema() del objeto df3. El método printSchema() es una función proporcionada por la API Apache Spark DataFrame que imprime el esquema del DataFrame en la consola. El esquema de un DataFrame describe la estructura de los datos, incluidos los nombres y tipos de datos de cada columna. Esto puede ser útil para comprender los datos y para depurar cualquier problema del DataFrame.

```
root
 |-- CustomerID: string (nullable = true)
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: string (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- from_date: timestamp (nullable = true)
 |-- recency: long (nullable = true)
 |-- frequency: long (nullable = false)
```

Valor monetario

Por último, calculemos el valor monetario: el importe total gastado por cada cliente en el dataframe. Hay dos pasos para conseguirlo:

i) Halla el importe total gastado en cada compra:

Cada ID de cliente tiene variables llamadas "Quantity" y "UnitPrice" para una sola compra:

```
+----------+--------+---------+
|CustomerID|Quantity|UnitPrice|
+----------+--------+---------+
|17850     |6       |2.55     |
|17850     |6       |3.39     |
|17850     |8       |2.75     |
|17850     |6       |3.39     |
|17850     |6       |3.39     |
+----------+--------+---------+
```

Para obtener el importe total gastado por cada cliente en una compra, tenemos que multiplicar "Quantity" por "UnitPrice":

```
m_val = df3.withColumn('TotalAmount',col("Quantity") * col("UnitPrice"))
```

Este código está escrito en Python y utiliza la biblioteca PySpark. El código crea una nueva columna llamada "TotalAmount" en el DataFrame PySpark "df3". Los valores de esta columna se calculan multiplicando los valores de la columna "Quantity" por los valores de la columna "UnitPrice" de cada fila del DataFrame. La función "withColumn" se utiliza para añadir una nueva columna al DataFrame. El primer argumento es el nombre de la nueva columna, y el segundo argumento es el cálculo que

debe realizarse en cada fila para rellenar la nueva columna. La función "col" se utiliza para hacer referencia a una columna del DataFrame. Toma el nombre de la columna como argumento y devuelve un objeto Columna que puede utilizarse en los cálculos. En general, este código calcula el importe total de cada artículo del DataFrame multiplicando la cantidad por el precio unitario, y añade esta información como nueva columna al DataFrame.

ii) Halla el importe total gastado por cada cliente:

Para hallar el importe total gastado por cada cliente en conjunto, solo tenemos que agrupar por la columna CustomerID y sumar el importe total gastado:

```
m_val = m_val.groupBy('CustomerID').agg(sum('TotalAmount').alias('monetary_value'))
```

Este código está escrito en Python y utiliza la biblioteca PySpark. El método groupBy se utiliza para agrupar los datos del DataFrame m_val por la columna CustomerID. A continuación, se utiliza el método agg para agregar los datos sumando los valores de la columna TotalAmount de cada grupo. El método alias se utiliza para cambiar el nombre de la columna resultante por monetary_value. El DataFrame resultante tendrá una fila por cada valor único de CustomerID, y la columna monetary_value contendrá la suma de los valores de TotalAmount de ese cliente.

Fusiona este dataframe con el resto de variables:

```
finaldf = m_val.join(df3,on='CustomerID',how='inner')
```

Este código está escrito en Python. El código crea un nuevo DataFrame llamado finaldf uniendo dos DataFrames existentes, m_val y df3, en la columna común CustomerID. El parámetro how se establece en 'inner', lo que significa que solo se incluirán en el resultado final las filas con valores de CustomerID coincidentes en ambos DataFrames. En resumen, este código realiza una operación de unión interna en dos DataFrames basada en una columna común y crea un nuevo DataFrame con los datos resultantes.

Ahora que hemos creado todas las variables necesarias para crear el modelo, ejecuta las siguientes líneas de código para seleccionar solo las columnas necesarias y eliminar las filas duplicadas del dataframe:

```
finaldf = finaldf.select(['recency','frequency','monetary_value','CustomerID']).distinct()
```

Este código está escrito en Python. El código selecciona columnas concretas de un DataFrame llamado finaldf y conserva solo las filas distintas. Las columnas seleccionadas son 'recency', 'frequency', 'monetary_value' y 'CustomerID'. El DataFrame resultante, finaldf, solo contendrá combinaciones únicas de estas cuatro columnas.

Mira la cabecera del dataframe final para asegurarte de que el preprocesamiento se haya realizado correctamente:

```
+-------+---------+------------------+----------+
|recency|frequency|monetary_value    |CustomerID|
+-------+---------+------------------+----------+
|5580   |14       |226.14            |16250     |
|2580   |1        |350.4             |15100     |
|30360  |14       |205.85999999999999|13065     |
|12660  |59       |390.78999999999985|12838     |
|18420  |5        |115.65            |15350     |
+-------+---------+------------------+----------+
only showing top 5 rows
```

Estandarización

Antes de crear el modelo de segmentación de clientes, estandaricemos el dataframe para asegurarnos de que todas las variables estén en torno a la misma escala:

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
assemble=VectorAssembler(inputCols=[
   'recency','frequency','monetary_value'
], outputCol='features')
assembled_data=assemble.transform(finaldf)
scale=StandardScaler(inputCol='features',outputCol='standardized')
data_scale=scale.fit(assembled_data)
data_scale_output=data_scale.transform(assembled_data)
```

Este código utiliza la biblioteca de machine learning de PySpark para realizar ingeniería de características en un conjunto de datos. En primer lugar, importa los módulos necesarios para la ingeniería de características: VectorAssembler y StandardScaler. A continuación, crea un objeto VectorAssembler llamado assemble que toma tres columnas de entrada (recency, frequency y monetary_value) y da como salida una única columna llamada features. Esto combina las tres columnas de entrada en un único vector que puede utilizarse como entrada para los algoritmos de machine learning. A continuación, aplica la transformación assemble al conjunto de datos finaldf utilizando el método transform, y almacena el resultado en assembled_data. A continuación, crea un objeto StandardScaler llamado scale que toma la columna features de assembled_data y genera una nueva columna llamada standardized. Esto escala la columna features de modo que cada característica tenga una media de 0 y una desviación típica de 1. Esto es importante para algunos algoritmos de machine learning que requieren una entrada estandarizada. Por último, aplica la transformación scale a assembled_data utilizando los métodos fit y transform y almacena el resultado en data_scale_output. Esto nos da un nuevo conjunto de datos en el que se ha estandarizado la columna features.

Ejecuta las siguientes líneas de código para ver qué aspecto tiene el vector de características estandarizado:

```
data_scale_output.select('standardized').show(2,truncate=False)
```

Este código está escrito en Python y utiliza la biblioteca PySpark. Se supone que la variable data_scale_output es un DataFrame de PySpark. El método select() se utiliza para seleccionar una columna concreta del DataFrame: en este caso, la columna standardized. A continuación, se invoca el método show() en el DataFrame resultante para mostrar las dos primeras filas de la columna standardized'. El parámetro truncate=False` garantiza que se muestre el contenido completo de la columna sin truncar.

```
+-----------------------------------------------------------+
|standardized                                               |
+-----------------------------------------------------------+
|[0.6860448646904733,0.6848507976304103,0.45968090513788246] |
|[0.3172035395880683,0.048917914116457885,0.7122675738936677]|
+-----------------------------------------------------------+
only showing top 2 rows
```

Estas son las características escaladas que se introducirán en el algoritmo de clustering.

Paso 5: Crear el modelo de machine learning

Ahora que hemos completado todo el análisis y la preparación de los datos, vamos a construir el modelo K-medias.

El algoritmo se creará utilizando la **API de machine learning** de PySpark.

i) Encontrar el número de clústeres que se va a utilizar

Al crear un modelo K-medias, primero tenemos que determinar el número de clústeres o grupos que queremos que devuelva el algoritmo. Si nos decidimos por tres grupos, por ejemplo, tendremos tres segmentos de clientes.

La técnica más utilizada para decidir cuántos clústeres utilizar en K-medias se denomina "método del codo".

Esto se hace simplemente ejecutando el algoritmo K-medias para una amplia gama de clústeres y visualizando los resultados del modelo para cada clúster. El gráfico tendrá un punto de inflexión parecido a un codo, y solo tenemos que elegir el número de clústeres de este punto.

Vamos a ejecutar las siguientes líneas de código para crear un algoritmo de K-medias de 2 a 10 clústeres:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import numpy as np
cost = np.zeros(10)
evaluator = ClusteringEvaluator(predictionCol='prediction',
featuresCol='standardized',metricName='silhouette', distanceMeasure='squaredEuclidean')
for i in range(2,10):
```

```
    KMeans_algo=KMeans(featuresCol='standardized', k=i)
    KMeans_fit=KMeans_algo.fit(data_scale_output)
    output=KMeans_fit.transform(data_scale_output)
    cost[i] = KMeans_fit.summary.trainingCost
```

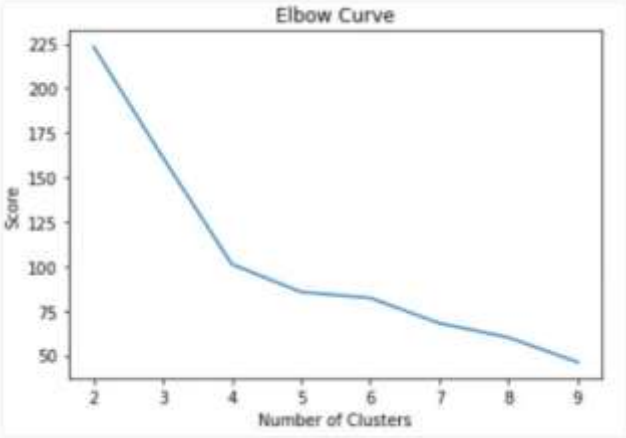
Este código importa las bibliotecas necesarias para ejecutar K-medias utilizando PySpark. A continuación, inicializa una matriz de ceros de longitud 10 para almacenar el coste de cada modelo K-medias. A continuación, crea una instancia de la clase ClusteringEvaluator, que se utilizará para evaluar la calidad de los resultados del clustering. El evaluador está configurado para utilizar la métrica silhouette con distancia euclidiana al cuadrado. A continuación, el código entra en un bucle que itera sobre los valores de k de 2 a 9. Para cada valor de k, se crea un algoritmo K-media con el número especificado de clústeres y la columna de características normalizadas. A continuación, el algoritmo se ajusta a los datos y se utiliza para transformarlos y obtener las asignaciones de clústeres. Por último, el coste de entrenamiento del modelo K-medias se almacena en la matriz de costes en el índice correspondiente al valor actual de k. El coste de entrenamiento es una medida de cómo se ajusta el modelo a los datos, y se calcula como la suma de las distancias al cuadrado entre cada punto y su centro de clúster asignado.

Con los códigos anteriores, hemos creado y evaluado correctamente un modelo K-medias con 2-10 clústeres. Los resultados se han colocado en una matriz, y ahora se pueden visualizar en un gráfico de líneas:

```
import pandas as pd
import pylab as pl
df_cost = pd.DataFrame(cost[2:])
df_cost.columns = ["cost"]
new_col = range(2,10)
df_cost.insert(0, 'cluster', new_col)
pl.plot(df_cost.cluster, df_cost.cost)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()
```

Este código importa las bibliotecas pandas y pylab y crea un DataFrame llamado df_cost utilizando la función pd.DataFrame(). La variable cost se pasa como argumento a esta función, con el corte [2:] indicando que solo deben incluirse en el DataFrame los valores a partir del índice 2. El DataFrame df_cost tiene una columna llamada "cost". La variable new_col se crea mediante la función range(), que genera una secuencia de enteros del 2 al 9. A continuación, esta secuencia se inserta como nueva columna llamada "cluster" al principio del DataFrame df_cost mediante el método df_cost.insert(). La función pl.plot() se utiliza para crear un gráfico de líneas de la columna "cluster" en el eje x y la columna "cost" en el eje y. Las funciones pl.xlabel(), pl.ylabel() y pl.title() se utilizan para añadir etiquetas al eje x, al eje y y al título del gráfico, respectivamente. Por último, se invoca la función pl.show() para mostrar el gráfico. En general, este código genera un gráfico de curva del codo para ayudar a determinar el número óptimo de clústeres de un algoritmo de clustering. El eje x representa el número de clústeres, y el eje y representa el coste o puntuación asociados a cada número de clústeres. El "codo" de la curva representa el punto de rendimiento decreciente, en el que añadir más clústeres no mejora significativamente el rendimiento del algoritmo de clustering.

Los códigos anteriores mostrarán el siguiente gráfico:



ii) Crear el modelo K-medias

En el gráfico anterior, podemos ver que hay un punto de inflexión que parece un codo en el cuatro. Por ello, procederemos a crear el algoritmo K-medias con cuatro clústeres:

```
KMeans_algo=KMeans(featuresCol='standardized', k=4)
KMeans_fit=KMeans_algo.fit(data_scale_output)
```

Este código utiliza el algoritmo K-medias de la biblioteca ML de Apache Spark para agrupar datos. La primera línea crea una instancia del algoritmo K-medias con el parámetro featuresCol establecido en 'standardized' y el parámetro k establecido en 4. El parámetro featuresCol especifica la columna del DataFrame de entrada que contiene las características que se utilizarán para el clustering, y 'standardized' se refiere probablemente a una columna que se ha preprocesado para tener una media de 0 y una desviación típica de 1. El parámetro k especifica el número de clústeres que se deben crear. La segunda línea ajusta el algoritmo K-medias a los datos de entrada, que se almacenan en el DataFrame data_scale_output. Esto crea un objeto KMeansModel que puede utilizarse para hacer previsiones sobre nuevos datos o para analizar los resultados del clustering.

iii) Hacer previsiones

Utilicemos el modelo que hemos creado para asignar clústeres a cada cliente del conjunto de datos:

```
preds=KMeans_fit.transform(data_scale_output)
preds.show(5,0)
```

Este fragmento de código está escrito en Python y utiliza el algoritmo K-medias de la biblioteca scikit-learn para agrupar datos. La primera línea de código aplica el objeto KMeans_fit (que ya se ha entrenado con algunos datos) a los datos escalados de la variable data_scale_output. Esto genera un conjunto de previsiones para cada punto de los datos de entrada. La segunda línea de código muestra las 5 primeras filas de las previsiones resultantes utilizando el método show(). El primer argumento de show() especifica el número de filas que se deben mostrar (en este caso, 5), y el segundo argumento especifica el número de caracteres que se deben mostrar por cada fila (en este caso, 0, lo que significa mostrar toda la fila).

Observa que en este dataframe tenemos la columna "prediction", que nos dice a qué clúster pertenece cada CustomerID:

```
+-------+---------+------------------+----------+--------------------+--------------------+----------+
|recency|frequency|    monetary_value|CustomerID|            features|        standardized|prediction|
+-------+---------+------------------+----------+--------------------+--------------------+----------+
|   5580|       14|            226.14|     16250|[5580.0,14.0,226.14]|[0.68604486469047...|         0|
|   2580|        1|             350.4|     15100|  [2580.0,1.0,350.4]|[0.31720353958806...|         0|
|  30360|       14|205.85999999999999|     13065|[30360.0,14.0,205...|[3.73267421003633...|         1|
|  12660|       59|390.78999999999985|     12838|[12660.0,59.0,390...|[1.55651039193214...|         1|
|  18420|        5|            115.65|     15350|[18420.0,5.0,115.65]|[2.26468573612876...|         0|
```

Paso 6: Análisis de clústeres

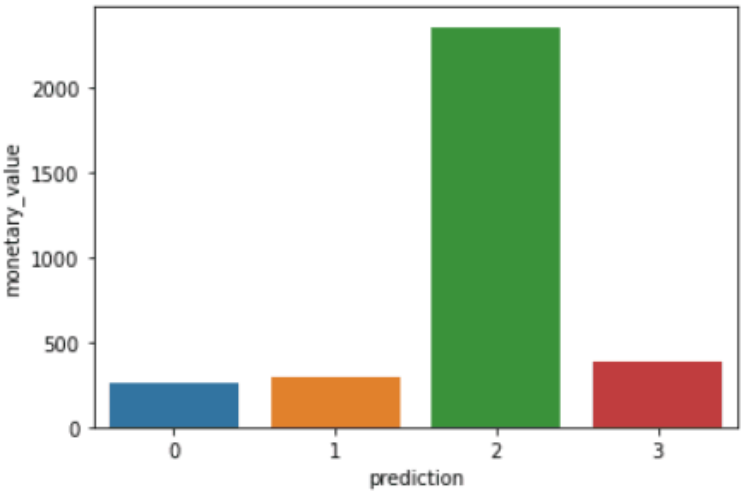
El último paso de todo este tutorial es analizar los segmentos de clientes que acabamos de crear.

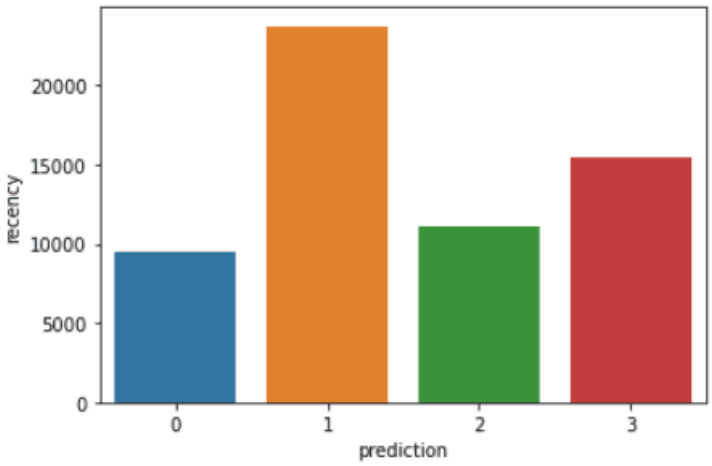
Ejecuta las siguientes líneas de código para visualizar Reciente, Frecuencia y Valor monetario en cada customerID en el dataframe:

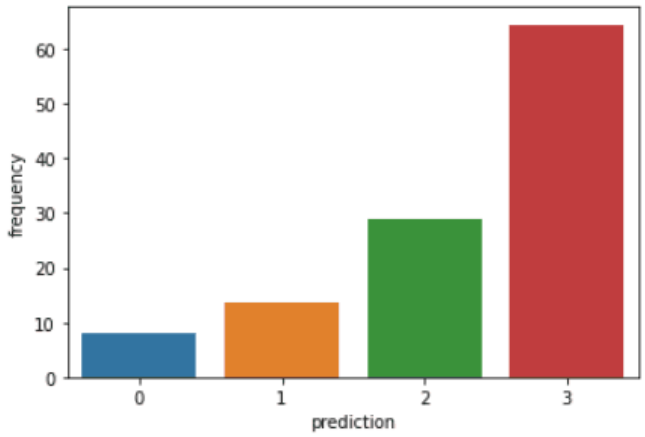
```
import matplotlib.pyplot as plt
import seaborn as sns
df_viz = preds.select('recency','frequency','monetary_value','prediction')
df_viz = df_viz.toPandas()
avg_df = df_viz.groupby(['prediction'], as_index=False).mean()
list1 = ['recency','frequency','monetary_value']
for i in list1:
  sns.barplot(x='prediction',y=str(i),data=avg_df)
  plt.show()
```

Este código importa las bibliotecas necesarias para la visualización de datos, concretamente matplotlib.pyplot y seaborn. A continuación, selecciona columnas concretas de un dataframe llamado preds y lo convierte en un dataframe Pandas llamado df_viz. A continuación, se utiliza la función groupby para agrupar los datos por la columna prediction y calcular la media de las demás columnas. El dataframe resultante se almacena en avg_df. A continuación, el código crea una lista de nombres de columnas llamada list1 e itera en cada columna mediante un bucle for. Para cada columna, se crea un gráfico de barras utilizando sns.barplot con el eje x establecido en prediction, el eje y establecido en el nombre de la columna actual y los datos establecidos en avg_df. Por último, se invoca plt.show() para mostrar el gráfico. En general, este código crea gráficos de barras para visualizar los valores medios de determinadas columnas agrupadas por una columna específica en un dataframe.

Los códigos anteriores mostrarán los siguientes gráficos:







Aquí tienes un resumen de las características que muestran los clientes de cada clúster:

- Clúster 0: Los clientes de este segmento muestran valores bajos en Reciente, Frecuencia y Valor monetario. Rara vez compran en la plataforma y son clientes de bajo potencial que probablemente dejen de hacer negocios con la empresa de comercio electrónico.

- Clúster 1: Los usuarios de este clúster muestran un valor alto en Reciente, pero no han gastado mucho en la plataforma. Tampoco visitan el sitio a menudo. Esto indica que pueden ser clientes nuevos que acaban de empezar a hacer negocios con la empresa.
- Clúster 2: Los clientes de este segmento muestran valores medios en Reciente y Frecuencia y gastan mucho dinero en la plataforma. Esto indica que tienden a comprar artículos de gran valor o a hacer compras al por mayor.
- Clúster 3: El segmento final comprende a los usuarios que muestran valores altos en Reciente y realizan compras frecuentes en la plataforma. Sin embargo, no gastan mucho en la plataforma, lo que podría significar que tienden a seleccionar artículos más baratos en cada compra.

Aprender PySpark desde cero. Próximos pasos:

Si has conseguido seguir todo este tutorial de PySpark, ¡enhorabuena! Ya has instalado correctamente PySpark en tu dispositivo local, has analizado un conjunto de datos de comercio electrónico y has creado un algoritmo de machine learning utilizando el marco.

Una advertencia sobre el análisis anterior es que se realizó con 2500 filas de datos de comercio electrónico recogidos en un solo día. El resultado de este análisis puede solidificarse si tenemos una mayor cantidad de datos con los que trabajar, ya que técnicas como el modelado RFM suelen aplicarse sobre meses de datos históricos.

Sin embargo, puedes tomar los principios aprendidos en este artículo y aplicarlos a una amplia variedad de conjuntos de datos mayores en el espacio del machine learning no supervisado.