

**CIENCIA DE DATOS:**

# **APRENDE LOS FUNDAMENTOS DE MANERA PRÁCTICA**



**SESION 05**

## **Programacion Spark**

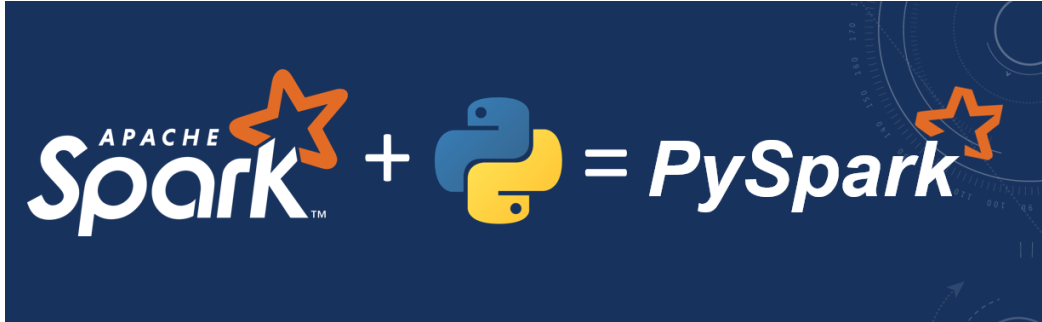
**Juan Chipoco**

mindquasar@gmail.com

# ÍNDICE

<b>OBJETIVO .....</b>	<b>4</b>
<b>PYSPARK.....</b>	<b>5</b>
<b>FEATURES.....</b>	<b>7</b>
<b>INSTALACION .....</b>	<b>9</b>
<b>EJEMPLO PYSPARK .....</b>	<b>12</b>
<i>Paso 1: Crear una SparkSession .....</i>	<i>13</i>
<i>Paso 2: crear el DataFrame.....</i>	<i>13</i>
<i>Paso 3: Análisis de datos exploratorios .....</i>	<i>14</i>
<i>Paso 4: Preprocesamiento de datos .....</i>	<i>16</i>
<i>Paso 5: Construcción del modelo de aprendizaje automático.....</i>	<i>21</i>
<i>Paso 6: Análisis de conglomerados .....</i>	<i>24</i>

## OBJETIVO



Apache Spark está escrito en lenguaje de programación Scala. Para admitir Python con Spark, la comunidad de Apache Spark lanzó una herramienta, PySpark. Con PySpark, también se puede trabajar con RDD en el lenguaje de programación Python. Es gracias a una biblioteca llamada Py4j que se puede lograr esto. Esta es una sesión introductoria que cubre los conceptos básicos de los documentos basados en datos y explica cómo manejar sus diversos componentes y subcomponentes.

## PySpark

### *¿Qué es PySpark?*

PySpark es una interfaz para Apache Spark en Python. Con PySpark, puede escribir comandos similares a Python y SQL para manipular y analizar datos en un entorno de procesamiento distribuido. Para aprender los conceptos básicos del lenguaje, puede tomar el curso Introducción a PySpark de Datacamp. Esta es una sesión para principiantes que te guiará a través de la manipulación de datos, la creación de canalizaciones de aprendizaje automático y el ajuste de modelos con PySpark.

### *¿Para qué se utiliza PySpark?*

La mayoría de los científicos y analistas de datos están familiarizados con Python y lo usan para implementar flujos de trabajo de aprendizaje automático. PySpark les permite trabajar con un lenguaje familiar en conjuntos de datos distribuidos a gran escala.

Apache Spark también se puede usar con otros lenguajes de programación de ciencia de datos como R. Si esto es algo que le interesa aprender, el curso Introducción a Spark con sparklyr en R es un excelente lugar para comenzar.

### *¿Por qué PySpark?*

Las empresas que recopilan terabytes de datos tendrán un gran marco de datos como Apache Spark. Para trabajar con estos conjuntos de datos a gran escala, el conocimiento de los marcos Python y R por sí solo no será suficiente.

Debe aprender un marco que le permita manipular conjuntos de datos sobre un sistema de procesamiento distribuido, ya que la mayoría de las organizaciones basadas en datos requerirán que lo haga. PySpark es un excelente lugar para comenzar, ya que su sintaxis es simple y puede aprenderse fácilmente si ya está familiarizado con Python.

La razón por la que las empresas eligen usar un marco como PySpark es por la rapidez con la que puede procesar grandes datos. Es más rápido que bibliotecas como Pandas y Dask, y puede manejar mayores cantidades de datos que estos marcos. Si tuviera más de petabytes de datos para procesar, por ejemplo, Pandas y Dask fallarían, pero PySpark podría manejarlos fácilmente.

Si bien también es posible escribir código Python sobre un sistema distribuido como Hadoop, muchas organizaciones eligen usar Spark en su lugar y usan la API de PySpark, ya que es más rápido y puede manejar datos en tiempo real. Con PySpark, puede escribir código para recopilar datos de una fuente que se actualiza continuamente, mientras que los datos solo se pueden procesar en modo por lotes con Hadoop.

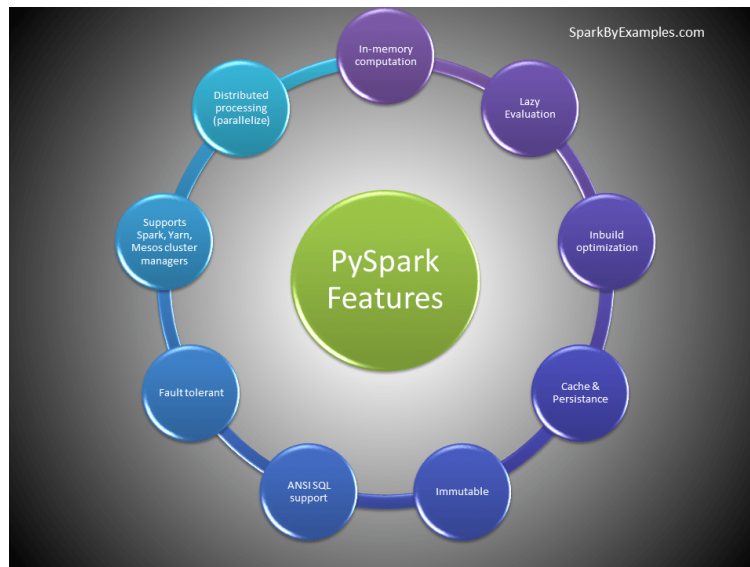
Apache Flink es un sistema de procesamiento distribuido que tiene una API de Python llamada PyFlink y, en realidad, es más rápido que Spark en términos de rendimiento. Sin embargo, Apache Spark existe desde hace más tiempo y cuenta con un mejor soporte de la comunidad, lo que significa que es más confiable.

Además, PySpark proporciona tolerancia a fallas, lo que significa que tiene la capacidad de recuperar pérdidas después de que ocurre una falla. El marco también tiene computación en memoria y se almacena en la memoria de acceso aleatorio (RAM). Puede ejecutarse en una máquina que no tenga un disco duro o SSD instalado.

### *¿Quién usa PySpark?*

PySpark se usa muy bien en la comunidad de ciencia de datos y aprendizaje automático, ya que hay muchas bibliotecas de ciencia de datos ampliamente utilizadas escritas en Python, incluidas NumPy, TensorFlow. También se utiliza debido a su procesamiento eficiente de grandes conjuntos de datos. PySpark ha sido utilizado por muchas organizaciones como Walmart, Trivago, Sanofi, Runtastic y muchas más.

## Features



- In-memory computation
- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
- Fault-tolerant
- Immutable
- Lazy evaluation
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

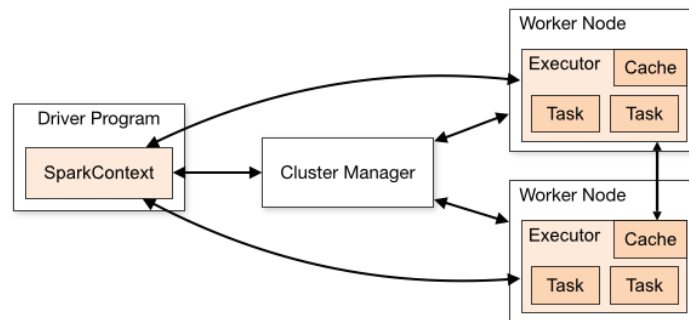
### Ventajas:

- PySpark es un motor de procesamiento distribuido, en memoria y de uso general que le permite procesar datos de manera eficiente y distribuida.
- Las aplicaciones que se ejecutan en PySpark son 100 veces más rápidas que los sistemas tradicionales.
- Obtendrás grandes beneficios al usar PySpark para canalizaciones de ingesta de datos.
- Con PySpark podemos procesar datos de Hadoop HDFS, AWS S3 y muchos sistemas de archivos.

- PySpark también se usa para procesar datos en tiempo real usando Streaming y Kafka.
- Con la transmisión de PySpark, también puede transmitir archivos desde el sistema de archivos y también desde el socket.
- PySpark tiene bibliotecas gráficas y de aprendizaje automático de forma nativa.

## Arquitectura PySpark

Como ya vimos Apache Spark funciona en una arquitectura maestro-esclavo donde el maestro se llama "Driver" y los esclavos se llaman "Workers". Cuando ejecuta una aplicación Spark, Spark Driver crea un contexto que es un punto de entrada a su aplicación, y todas las operaciones (transformaciones y acciones) se ejecutan en los nodos trabajadores y los recursos son administrados por el Administrador de clústeres.



## Instalacion

### Cómo instalar PySpark

Requisitos previos:

Antes de instalar Apache Spark y PySpark, debe tener el siguiente software configurado en su dispositivo:

#### *Python*

Si aún no tiene instalado Python, siga nuestra guía de configuración para desarrolladores de Python para configurarlo antes de continuar con el siguiente paso.

#### *Java*

Se explicara como instalar Java en su computadora si está usando Windows.

#### *Jupyter Notebook*

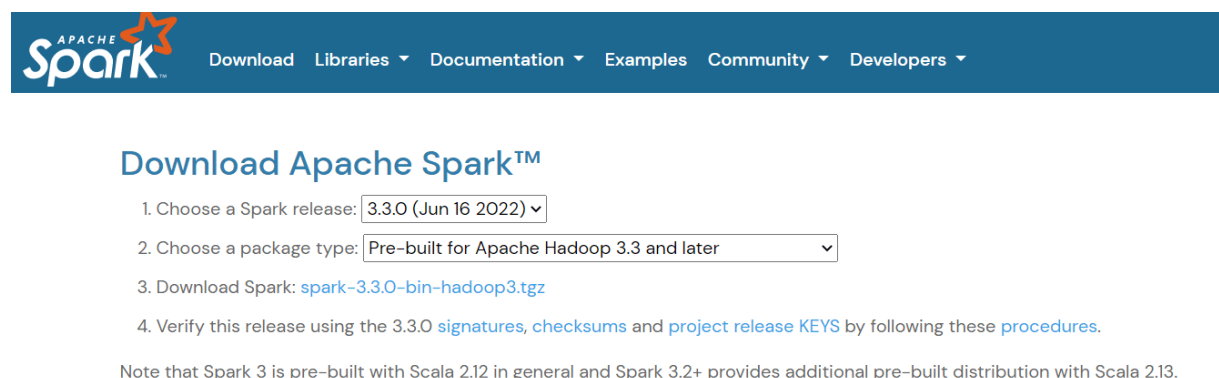
Un Jupyter Notebook es una aplicación web que puede usar para escribir código y mostrar ecuaciones, visualizaciones y texto. Es uno de los editores de programación más utilizados por los científicos de datos. Usaremos un Jupyter Notebook para escribir todo el código de PySpark en este tutorial, así que asegúrese de tenerlo instalado.

#### *Dataset*

Usaremos el Dataset de comercio electrónico de Datacamp para todos los análisis de esta sesión. Cambiamos el nombre del archivo a "datacamp\_ecommerce.csv" y lo guardamos en el directorio principal.

### Instalación de Apache Spark

Para configurar Apache Spark, vaya a la página de descarga y descargue el archivo .tgz que se muestra en la página:

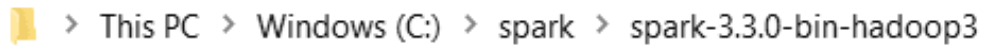


The screenshot shows the Apache Spark download page. At the top is the Apache Spark logo and a navigation bar with links: Download, Libraries, Documentation, Examples, Community, and Developers. Below the navigation bar is the heading "Download Apache Spark™". There are two dropdown menus: "1. Choose a Spark release:" with "3.3.0 (Jun 16 2022)" selected, and "2. Choose a package type:" with "Pre-built for Apache Hadoop 3.3 and later" selected. Below these is the text "3. Download Spark: [spark-3.3.0-bin-hadoop3.tgz](#)". Then it says "4. Verify this release using the 3.3.0 [signatures](#), [checksums](#) and [project release KEYS](#) by following these [procedures](#)." At the bottom, a note states: "Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13."



Luego, si está utilizando Windows, cree una carpeta en su directorio C llamada "Spark". Si usa Linux o Mac, puede pegar esto en una nueva carpeta en su directorio de inicio.

A continuación, extraiga el archivo que acaba de descargar y pegue su contenido en esta carpeta "Spark". Así es como debería verse la ruta de la carpeta:



Ahora, se necesita configurar sus variables de entorno. Hay dos maneras de hacer esto:

*Método 1:* Cambio de variables de entorno mediante Powershell

Si está utilizando una máquina con Windows, la primera forma de cambiar las variables de entorno es mediante Powershell:

Paso 1: haga clic en Inicio -> Windows Powershell -> Ejecutar como administrador

Paso 2: escriba la siguiente línea en Windows Powershell para configurar SPARK\_HOME:

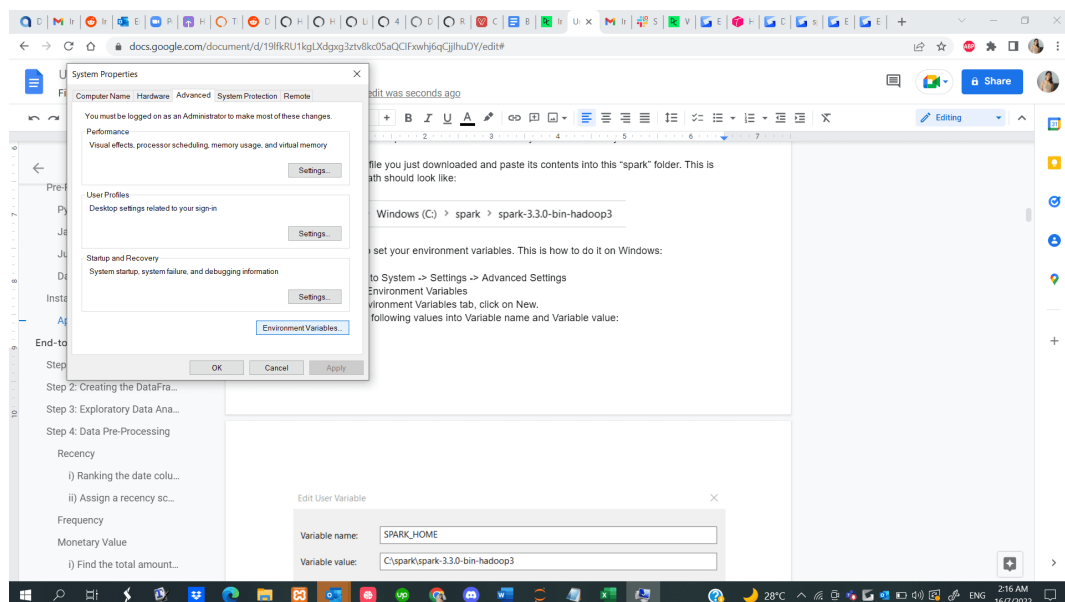
setx SPARK\_HOME "C:\spark\spark-3.3.0-bin-hadoop3" # change this to your path

Paso 3: A continuación, configure su directorio Spark bin como una variable de ruta:  
setx PATH "C:\spark\spark-3.3.0-bin-hadoop3\bin"

*Método 2:* cambiar las variables de entorno manualmente

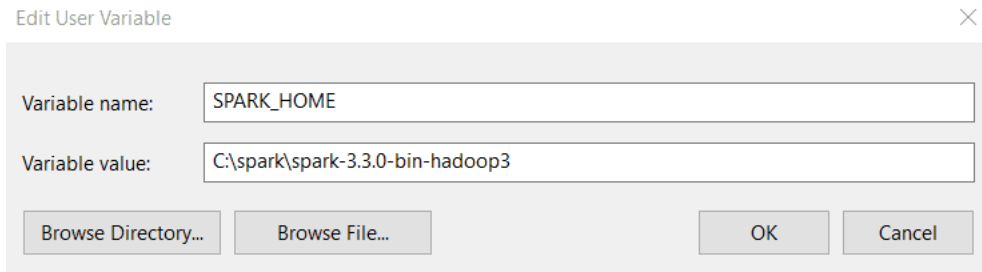
Paso 1: Vaya a Inicio -> Sistema -> Configuración -> Configuración avanzada

Paso 2: haga clic en Variables de entorno



Paso 3: en la pestaña Variables de entorno, haga clic en Nuevo.

Paso 4: Ingrese los siguientes valores en Nombre de variable y Valor de variable. Tenga en cuenta que la versión que instale puede ser diferente de la que se muestra a continuación, así que copie y pegue la ruta a su directorio de Spark.



Dialog box titled "Edit User Variable" with a close button (X) in the top right corner. It contains two input fields: "Variable name:" with the text "SPARK\_HOME" and "Variable value:" with the text "C:\spark\spark-3.3.0-bin-hadoop3". Below the fields are four buttons: "Browse Directory...", "Browse File...", "OK", and "Cancel".

Paso 5: A continuación, en la pestaña Variables de entorno, haga clic en Ruta y seleccione Editar.

Paso 6: haga clic en Nuevo y pegue la ruta a su directorio Spark bin. Aquí hay un ejemplo de cómo se ve el directorio bin:

C:\spark\spark-3.3.0-bin-hadoop3\bin

## Instalación de PySpark

Ahora que has instalado correctamente Apache Spark y todos los demás requisitos previos necesarios, abra un archivo de Python en su Jupyter Notebook y ejecute las siguientes líneas de código en la primera celda:

```
!pip install pyspark
```

## Ejemplo PySpark

Ahora que tiene PySpark en funcionamiento, te mostraremos cómo ejecutar un proyecto de segmentación de clientes de extremo a extremo 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 visita Starbucks solo durante el verano para comprar bebidas frías, puede ser segmentado como "comprador de temporada" y atraerse con promociones especiales seleccionadas para la temporada de verano.

Los científicos de datos suelen crear algoritmos de aprendizaje automático no supervisados, como el agrupamiento de K-Means o el 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 para el ojo humano.

En esta sección, utilizaremos la agrupación en clústeres de K-Means para realizar la segmentación de clientes en el conjunto de datos de comercio electrónico que descargamos anteriormente.

Al final de esta sección, estará familiarizado con los siguientes conceptos:

- Lectura de archivos csv con PySpark
- Análisis exploratorio de datos con PySpark
- Agrupación y clasificación de datos.
- Realización de operaciones aritméticas
- Agregar conjuntos de datos
- Preprocesamiento de datos con PySpark
- Trabajar con valores de fecha y hora
- Tipo de conversión
- Uniendo dos marcos de datos
- La función de rango()
- Aprendizaje automático de PySpark
- Crear un vector de características
- Estandarización de datos
- Construcción de un modelo de agrupamiento de K-Means
- Interpretando el modelo

## Paso 1: Crear una SparkSession

Una SparkSession es un punto de entrada a todas las funciones de Spark y es necesaria si desea crear un marco de datos en PySpark. Ejecute 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()
```

Usando los códigos anteriores, construimos una sesión Spark y establecimos un nombre para la aplicación. Luego, los datos se almacenaron en caché en la memoria fuera del montón para evitar almacenarlos directamente en el disco, y la cantidad de memoria se especificó manualmente.

## 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="\")
```

Tenga en cuenta que definimos un carácter de escape para evitar comas en el archivo .csv al analizar.

Echemos un vistazo al encabezado del dataframe usando la función show():

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

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 marco de datos 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. Descripción: El artículo comprado por el cliente.
4. Cantidad: El número de cada artículo comprado 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. País: El país desde donde se realizó la compra

### Paso 3: Análisis de datos exploratorios

Ahora que hemos visto las variables presentes en este conjunto de datos, realicemos un análisis exploratorio de datos para comprender mejor estos puntos de datos:

1. Comencemos contando el número de filas en el marco de datos:

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

1. ¿Cuántos clientes únicos están presentes en el marco de datos?

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

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

Para encontrar el país desde el que se realizan la mayoría de las compras, necesitamos usar la cláusula `groupBy()` en PySpark:

```
from pyspark.sql.functions import *  
  
from pyspark.sql.types import *  
  
df.groupBy('Country').agg(countDistinct('CustomerID').alias('country_count')).show()
```

La siguiente tabla se representará después de ejecutar los códigos anteriores:

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

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

Tenga en cuenta que los datos de la tabla anterior no se presentan en el orden de las compras. Para ordenar esta tabla, podemos incluir la cláusula `orderBy()`:

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

La salida que se muestra ahora está ordenada 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 saber cuándo se realizó la última compra en la plataforma, debemos convertir la columna "Fecha de la factura" a un formato de marca de tiempo y usar la función `max()` en 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()
```

Deberías ver aparecer 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?

Similar a lo que hicimos anteriormente, la función `min()` se puede usar para encontrar la fecha y hora de compra más tempranas:

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

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

Tenga en cuenta que las compras más recientes y más antiguas se realizaron el mismo día con solo unas pocas horas de diferencia. Esto significa que el conjunto de datos que descargamos contiene información de solo 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, debemos preparar los datos para alimentar el algoritmo de aprendizaje automático.

Echemos un vistazo al encabezado del marco de datos una vez más para comprender cómo se realizará el preprocesamiento:

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

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

only showing top 5 rows

A partir del conjunto de datos anterior, necesitamos crear múltiples segmentos de clientes basados en el comportamiento de compra de cada usuario.

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

Debido a esto, utilizaremos las variables existentes para derivar tres nuevas características informativas: actualidad, frecuencia y valor monetario (RFM).

**RFM** se usa comúnmente en marketing para evaluar el valor de un cliente en función de su:

1. Actualidad: ¿Qué tan recientemente ha realizado una compra cada cliente?
2. Frecuencia: ¿Con qué frecuencia han comprado algo?
3. Valor Monetario: ¿Cuánto dinero gastan en promedio al realizar compras?

Ahora preprocesaremos el DataFrame para crear las variables anteriores.

## 1 Recency

Primero, calculemos el valor de la actualidad: la última fecha y hora en que se realizó una compra en la plataforma. Esto se puede lograr en dos pasos:

### 2 i) Asignar una puntuación de actualidad a cada cliente

Restaremos cada fecha en el marco de datos desde la fecha más antigua. Esto nos dirá qué tan recientemente se vio un cliente en el marco de datos. Un valor de 0 indica la antigüedad más baja, ya que se asignará a la persona que fue vista realizando 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"))
```

### 3 ii) Seleccione la compra más reciente

Un cliente puede realizar varias compras en diferentes momentos. Necesitamos seleccionar solo la última vez que fueron vistos comprando un producto, ya que esto es indicativo de cuándo se realizó la compra más reciente:

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

Veamos el encabezado del nuevo marco de datos. Ahora tiene una variable llamada "actualidad" adjunta:

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



```

+-----+-----+-----+-----+-----+-----+-----+-----+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|date|
|from_date|recency|
+-----+-----+-----+-----+-----+-----+-----+-----+
|536365|85123A|WHITE HANGING HEART T-LIGHT HOLDER|6|12/1/10 8:26|2.55|17850|United Kingdom|2012-01-10 0
8:26:00|2012-01-10 08:26:00|0|
|536365|71053|WHITE METAL LANTERN|6|12/1/10 8:26|3.39|17850|United Kingdom|2012-01-10 0
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|17850|United Kingdom|2012-01-10 0
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|17850|United Kingdom|2012-01-10 0
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|17850|United Kingdom|2012-01-10 0
8:26:00|2012-01-10 08:26:00|0|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

Una forma más fácil de ver todas las variables presentes en un marco de datos de PySpark es usar su función `printSchema()`. Este es el equivalente de la función `info()` en Pandas:

```
df2.printSchema()
```

La salida renderizada debería verse así:

```

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)

```

## 4 Frecuencia

Ahora calculemos el valor de la frecuencia: con qué frecuencia un cliente compra algo en la plataforma. Para hacer esto, solo necesitamos agrupar por ID de cada cliente y contar la cantidad de artículos que compraron:

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

Mire el encabezado de este nuevo marco de datos que acabamos de crear:

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

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

Hay un valor de frecuencia adjunto a cada cliente en el marco de datos. Este nuevo marco de datos solo tiene dos columnas, y debemos unirlo con el anterior:

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

Imprimamos el esquema de este marco de datos:

```
df3.printSchema()
```

```
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)
```

## 5 Valor monetario

Finalmente, calculemos el valor monetario: la cantidad total gastada por cada cliente en el marco de datos. Hay dos pasos para lograr esto:

### 6 i) Encuentra el monto total gastado en cada compra:

Cada ID de cliente viene con variables llamadas "Cantidad" y "Precio unitario" 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 monto total gastado por cada cliente en una compra, debemos multiplicar "Cantidad" por "Precio unitario":

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

## 7 ii) Encuentre el monto total gastado por cada cliente:

Para encontrar el monto total gastado por cada cliente en general, solo necesitamos agrupar por la columna CustomerID y sumar el monto total gastado:

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

Combine este marco de datos con todas las demás variables:

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

Ahora que hemos creado todas las variables necesarias para construir el modelo, ejecute las siguientes líneas de código para seleccionar solo las columnas requeridas y suelte las filas duplicadas del marco de datos:

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

Mire el encabezado del marco de datos final para asegurarse de que el preprocesamiento se haya realizado con precisión:

```
+-----+-----+-----+-----+
|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
```

## 8 Estandarización

Antes de construir el modelo de segmentación de clientes, estandaricemos el marco de datos para asegurarnos de que todas las variables estén en 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)
```

Ejecute las siguientes líneas de código para ver cómo se ve el vector de características estandarizado:

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

```
+-----+  
| 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 incorporarán al algoritmo de agrupamiento.

## Paso 5: Construcción del modelo de aprendizaje automático

Ahora que hemos completado todo el análisis y la preparación de datos, construyamos el modelo de agrupación en clústeres de K-Means.

El algoritmo se creará utilizando la [API de aprendizaje automático](#) de PySpark .

### 9 i) Encontrar el número de clústeres a utilizar

Al construir un modelo de agrupación en clústeres de K-Means, primero debemos determinar la cantidad de clústeres o grupos que queremos que devuelva el algoritmo. Si nos decidimos por tres clústeres, por ejemplo, tendremos tres segmentos de clientes.

La técnica más popular utilizada para decidir cuántos conglomerados usar en K-Means se denomina "método del codo".

Esto se hace simplemente ejecutando el algoritmo K-Means para una amplia gama de conglomerados y visualizando los resultados del modelo para cada conglomerado. El gráfico tendrá un punto de inflexión que se parece a un codo, y solo elegimos la cantidad de grupos en este punto.

Lea este tutorial de agrupación en clústeres de Datacamp [K-Means](#) para obtener más información sobre cómo funciona el algoritmo.

Ejecutemos las siguientes líneas de código para construir un algoritmo de agrupación en clústeres de K-Means 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
```

Con los códigos anteriores, hemos construido y evaluado con éxito un modelo de agrupación en clústeres de K-Means con 2 a 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)

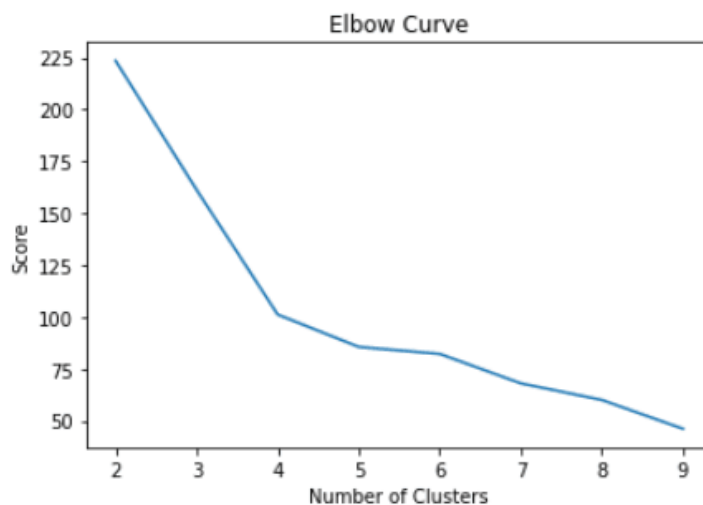
pl.xlabel('Number of Clusters')

pl.ylabel('Score')

pl.title('Elbow Curve')

pl.show()
```

Los códigos anteriores generarán el siguiente gráfico:



## 10 ii) Construyendo el Modelo de Agrupamiento de K-Means

En el diagrama de arriba, podemos ver que hay un punto de inflexión que parece un codo en cuatro. Debido a esto, procederemos a construir el algoritmo K-Means con cuatro clusters:

```
KMeans_algo=KMeans(featuresCol='standardized', k=4)

KMeans_fit=KMeans_algo.fit(data_scale_output)
```

## 11 iii) Hacer predicciones

Usemos el modelo que creamos para asignar clústeres a cada cliente en el conjunto de datos:

```
preds=KMeans_fit.transform(data_scale_output)

preds.show(5,0)
```

Tenga en cuenta que hay una columna de "predicción" en este marco de datos que nos dice a qué grupo pertenece cada ID de cliente:

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 conglomerados

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

Ejecute las siguientes líneas de código para visualizar la actualidad, la frecuencia y el valor monetario de cada ID de cliente en el marco de datos:

```
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()

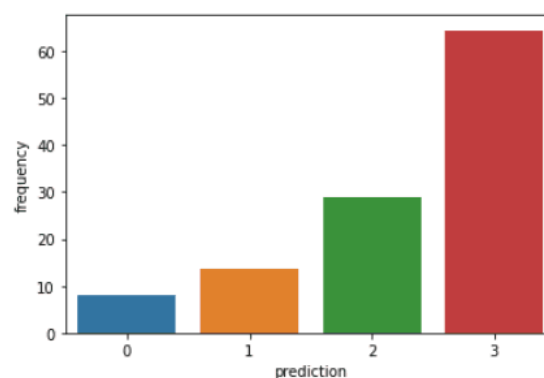
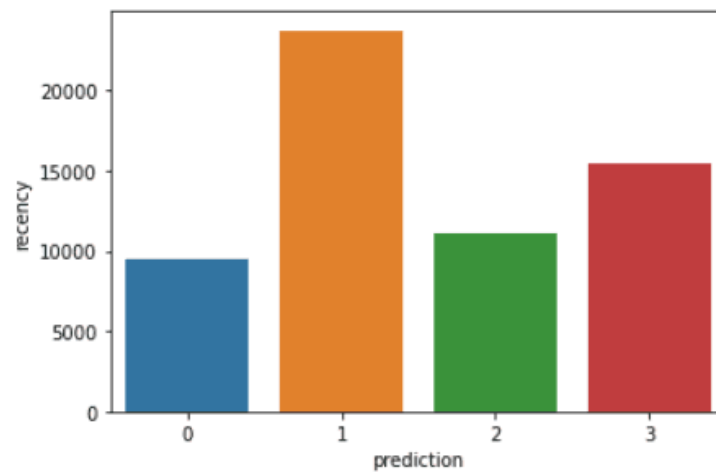
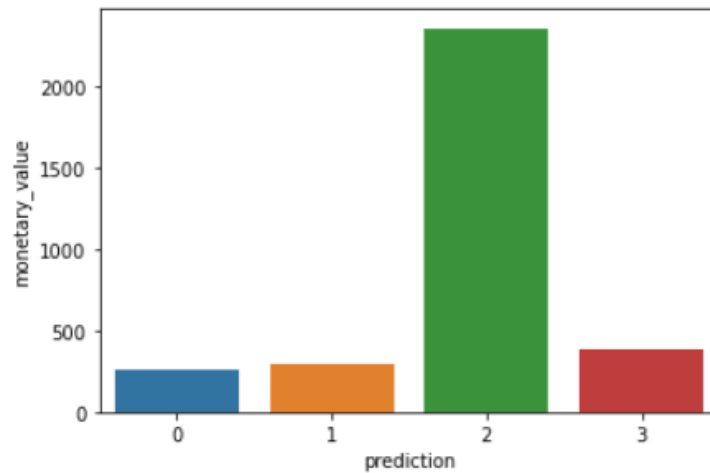
list1 = ['recency','frequency','monetary_value']

for i in list1:

    sns.barplot(x='prediction',y=str(i),data=avg_df)

plt.show()
```

Los códigos anteriores generarán los siguientes gráficos:



Aquí hay una descripción general de las características mostradas por los clientes en cada grupo:

- Grupo 0: los clientes de este segmento muestran poca actualidad, frecuencia y valor monetario. Rara vez compran en la plataforma y son clientes de bajo potencial que probablemente dejarán de hacer negocios con la empresa de comercio electrónico.



- Grupo 1: los usuarios de este grupo muestran una alta antigüedad, pero no se les ha visto gastar mucho en la plataforma. Tampoco visitan el sitio con frecuencia. Esto indica que podrían ser clientes más nuevos que acaban de comenzar a hacer negocios con la empresa.
- Grupo 2: los clientes de este segmento muestran una frecuencia y actualidad medias y gastan mucho dinero en la plataforma. Esto indica que tienden a comprar artículos de alto valor o hacer compras al por mayor.
- Grupo 3: el segmento final comprende a los usuarios que muestran una alta actualidad 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.