

Tarea M52 – Bryan Alberto Coronado García

- Configuración de sesión de Spark en terminal

```
(base) bryancoronado@192 ~ % source ~/.bashrc
(base) bryancoronado@192 ~ % spark-shell
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

  ____
 /  _ \
/_/_/ \_/_/ version 4.0.0

Using Scala version 2.13.16 (OpenJDK 64-Bit Server VM, Java 17.0.16)
Type in expressions to have them evaluated.
Type :help for more information.
25/08/05 20:29:51 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable
Spark context Web UI available at http://192.168.1.23:4040
Spark context available as 'sc' (master = local[*], app id = local-1754447392107).
scala> █
```

Configuración de plataforma Spark

```
1 # Importación de librerías para la practica de Spark
2 import sys
3 from operator import add
4 from pyspark.sql import SparkSession
5 import warnings
6 warnings.filterwarnings("ignore")
7
8 # Configuración de Spark
9 spark = SparkSession.builder.appName("M52").getOrCreate()
```

Python

WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/07/31 17:00:07 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

- Importación de datos Housing en Spark y verificación del tipo de dataframe

Importación de datos de Housing a una estructura Spark

```
1 # Generación de variable para la ruta del archivo
2 path_archivo = '/Users/bryancoronado/Desktop/Housing.csv'
3
4 # Generación del DataFrame a partir del archivo CSV (con headers para facilitar la lectura)
5 df = spark.read.csv(path_archivo, header=True)
6
7 # Verificación de la carga del DataFrame
8 df.show(10)
```

Python

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
[13300000]	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished	
[12250000]	8960	4	4	4	yes	no	no	no	yes	3	no	furnished	
[12250000]	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished	
[12215000]	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished	
[11410000]	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished	
[10850000]	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished	
[10150000]	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished	
[10150000]	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished	
[9870000]	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished	
[9800000]	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished	

only showing top 10 rows

```
1 # Verificación del tipo de datos
2 print("Tipo de datos del DataFrame:")
3 type(df)
```

Python

Tipo de datos del DataFrame:

pyspark.sql.classic.dataframe.DataFrame

- Modificación del tipo de datos

```

1 # Transformación de los tipos de datos para asegurar que son correctos
2
3 from pyspark.sql.types import IntegerType, FloatType
4
5 df = df.withColumn("price", df.price.cast(IntegerType()))
6 df = df.withColumn("area", df.area.cast(IntegerType()))
7 df = df.withColumn("bedrooms", df.bedrooms.cast(IntegerType()))
8 df = df.withColumn("bathrooms", df.bathrooms.cast(IntegerType()))
9 df = df.withColumn("stories", df.stories.cast(IntegerType()))
10 df = df.withColumn("parking", df.parking.cast(IntegerType()))
11
12 df.printSchema()

```

```

root
|-- price: integer (nullable = true)
|-- area: integer (nullable = true)
|-- bedrooms: integer (nullable = true)
|-- bathrooms: integer (nullable = true)
|-- stories: integer (nullable = true)
|-- mainroad: string (nullable = true)
|-- guestroom: string (nullable = true)
|-- basement: string (nullable = true)
|-- hotwaterheating: string (nullable = true)
|-- airconditioning: string (nullable = true)
|-- parking: integer (nullable = true)
|-- prefarea: string (nullable = true)
|-- furnishingstatus: string (nullable = true)

```

- Listado completo ordenado por ZipCode (Area)

1) Listado completo de columnas ordenado por zipcode

```

1 import pyspark.sql.functions as F1
2
3 # Se toma la columna "area" como sustituto para "zipcode" ya que no se encuentra en el dataset
4 df.sort(F1.col("area")).show()

```

price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
3150000	1650	3	1	2	no	no	yes	no	no	0	no	unfurnished
1890000	1700	3	1	2	yes	no	no	no	no	0	no	unfurnished
2275000	1836	2	1	1	no	no	yes	no	no	0	no	semi-furnished
4340000	1905	5	1	2	no	no	yes	no	no	0	no	semi-furnished
3430000	1950	3	2	2	yes	no	yes	no	no	0	yes	unfurnished
2240000	1950	3	1	1	no	no	no	yes	no	0	no	unfurnished
2660000	2000	2	1	2	yes	no	no	no	no	0	no	semi-furnished
3850000	2015	3	1	2	yes	no	yes	no	no	0	yes	semi-furnished
3500000	2135	3	2	2	no	no	no	no	no	0	no	unfurnished
4200000	2145	3	1	3	yes	no	no	no	no	1	yes	unfurnished
3920000	2145	4	2	1	yes	no	yes	no	no	0	yes	unfurnished
3465000	2145	3	1	3	yes	no	no	no	no	0	yes	furnished
3430000	2145	3	1	3	yes	no	no	no	no	0	yes	furnished
3332000	2145	3	1	2	yes	no	yes	no	no	0	yes	furnished
3290000	2145	3	1	2	yes	no	yes	no	no	0	yes	furnished
3080000	2160	3	1	2	no	no	yes	no	no	0	no	semi-furnished
4270000	2175	3	1	2	no	yes	yes	no	yes	0	no	unfurnished
3850000	2176	2	1	2	yes	yes	no	no	no	0	yes	semi-furnished
3640000	2275	3	1	3	yes	no	no	yes	yes	0	yes	semi-furnished
4200000	2325	3	1	2	no	no	no	no	no	0	no	semi-furnished

only showing top 20 rows

- Listado con mayor número de casas, precio promedio, y ordenado por ZipCode (Area)

```
1 df.groupBy("area").agg( \
2     F1.count("*").alias("num_casas"),
3     F1.avg("price").alias("precio_promedio")) \
4     .orderBy(F1.desc("num_casas")).show()
5
6 # NOTA: No es posible realizar una columna del tamaño de la casa, ya que no existe la data correspondiente.
7 # El punto actualizado es "Listado de zipcodes ordenados por el número de casas, y el promedio de precio"
```

area	num_casas	precio_promedio
6000	24	7051479.166666667
3000	14	3309000.0
4500	13	4031192.3076923075
4000	11	4040272.727272727
5500	9	5762555.555555556
6600	9	6443111.111111111
3600	8	3360437.5
3180	7	3530000.0
4040	7	4139000.0
6360	7	5604000.0
3640	7	3542000.0
3630	7	3515000.0
3500	6	4275833.333333333
5400	6	4631666.666666667
2145	6	3606166.6666666665
3450	5	3680600.0
3850	5	3136000.0
3480	5	3227000.0
4800	5	5742800.0
3520	5	4107600.0

only showing top 20 rows

- Listado con mayor número de habitaciones, de baños, precio promedio, y ordenado por ZipCode (Area)

```
1 df.groupBy("area").agg( \
2     F1.count("bedrooms").alias("num_habitaciones"),
3     F1.count("bathrooms").alias("num_baños"),
4     F1.avg("price").alias("precio_promedio")) \
5     .orderBy(F1.desc("num_habitaciones"), F1.desc("num_baños")).show(20)
```

area	num_habitaciones	num_baños	precio_promedio
6000	24	24	7051479.166666667
3000	14	14	3309000.0
4500	13	13	4031192.3076923075
4000	11	11	4040272.727272727
5500	9	9	5762555.555555556
6600	9	9	6443111.111111111
3600	8	8	3360437.5
3180	7	7	3530000.0
4040	7	7	4139000.0
6360	7	7	5604000.0
3640	7	7	3542000.0
3630	7	7	3515000.0
3500	6	6	4275833.333333333
5400	6	6	4631666.666666667
2145	6	6	3606166.6666666665
3450	5	5	3680600.0
3850	5	5	3136000.0
3480	5	5	3227000.0
4800	5	5	5742800.0
3520	5	5	4107600.0

only showing top 20 rows