

Manolo Ramírez Pintor - A01706155

Módulo Big Data:

Utilización, procesamiento y visualización de grandes volúmenes de datos

1. Configurando el entorno de trabajo PySpark

Iniciamos revisando los recursos que tenemos disponibles en el sistema

Tenemos aproximadamente medio GB de RAM utilizado de los 24 GB que tenemos disponibles en total, así que tenemos suficientes recursos para trabajar con un dataset grande.

In [1]:

```
!free -m
```

	total	used	free	shared	buff/cache	available
Mem:	23988	11954	6001	5024	6031	6733
Swap:	0	0	0			

In [2]:

```
!whoami
```

ubuntu

Para evitar tener muchos mensajes de advertencia en el notebook, importamos warnings para filtrarlos todos

In [3]:

```
import warnings  
warnings.filterwarnings('ignore')
```

Ahora, ponemos las rutas de donde tenemos instalado Java 8 y Spark

In [4]:

```
#Estableciendo variable de entorno  
import os  
# import pandas as pd  
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-arm64"  
os.environ["SPARK_HOME"] = "/home/mc/spark/spark-3.2.2-bin-hadoop3.2"
```

Importamos findspark e inicializamos la instalación de Spark

In [5]:

```
#Buscando e inicializando la instalación de Spark  
import findspark  
findspark.init()  
findspark.find()
```

Out[5]: '/home/mc/spark/spark-3.2.2-bin-hadoop3.2'

Ahora importamos SparkSession y creamos una sesión para este trabajo, en mi caso se llama bigData_Manolo

```
In [6]: from pyspark.sql import SparkSession
spark_session = SparkSession.builder.appName('bigData_Manolo').getOrCreate()
spark_session
```

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/10/29 03:46:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform
m... using builtin-java classes where applicable

Out[6]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.2.2
Master	local[*]
AppName	bigData_Manolo

2. Seleccionando un dataset de gran tamaño

En mi caso, encontré y seleccioné el dataset de [Car Sales](#), este lo encontré en Kaggle usando los filtros de tamaño. El peso del dataset descomprimido llega a 2GB.

3. Generando un modelo inteligente de regresión con MLlib

a) EDA básico

Ya que en Kaggle se indica que los datos no están completamente limpios, voy a revisar qué columnas tienen datos faltantes y qué es lo que puede servirme para luego hacer ETL y comenzar a realizar predicciones.

Al momento de cargar el dataset, podemos ver que existen distintas columnas. Estos son datos obtenidos de una página parecida a un Mercado Libre ruso de automóviles.

Tenemos columnas como la marca, el modelo, el tipo de carrocería, color, tipo de combustible, año, kilometraje, tipo de transmisión, poder en caballos de fuerza, precio en rublos, el nombre del motor, la capacidad del motor (en litros), la fecha de publicación, la ubicación del automóvil, el link de la publicación, la descripción y el momento en el que se capturó la información desde la página.

```
In [7]: df_spark = spark_session.read.option("header",True).csv('region25.csv')
df_spark.columns
```

```
Out[7]: ['brand',
 'name',
 'bodyType',
 'color',
 'fuelType',
 'year',
 'mileage',
 'transmission',
 'power',
 'price',
 'vehicleConfiguration',
 'engineName',
 'engineDisplacement',
 'date',
 'location',
 'link',
 'description',
 'parse_date']
```

En base a la información que encontré en Kaggle, estableceremos el tipo de dato por columna.

```
In [8]: df_spark = df_spark.withColumn("date",df_spark.date.cast('string'))
df_spark = df_spark.withColumn("parse_date",df_spark.parse_date.cast('string'))
df_spark = df_spark.withColumn("year",df_spark.year.cast('int'))
df_spark = df_spark.withColumn("mileage",df_spark.mileage.cast('int'))
df_spark = df_spark.withColumn("power",df_spark.power.cast('int'))
df_spark = df_spark.withColumn("price",df_spark.price.cast('int'))
```

A continuación realizaré un describe, aparecerá roto por el gran número de columnas pero lo arreglaré con Markdown...

- Tenemos 1,513,200 filas totales en el dataset.
- El precio promedio de los automóviles es de 1,368,558.3 rublos.
- Hay columnas que son de datos numéricos pero que presentan valores nulos.

```
In [9]: df_spark.describe().show()
```

```
[Stage 1:===== (14 + 1) / 15]
```

summary	brand	name	bodyType	color	fuelType	year	
mileage	transmission	power		price	vehicleConfiguration	engineName	engine
eDisplacement		date	location		link		description
parse_date							
count	1513200	1513200	1513200	1403466	1509640	1102226	
1498720	1510135	1492313		1513200		1102226	1101142
1092435		1513200	1513200		1513200	1477463	1
513200							
mean	null	1773.5805008944544		null	null	2010.3404338130292	134250.932
12874988		null	145.8111408263548	1368558.300035686	11.250605168465825		null
null		null	null		null	8.9146877521E10	3.3098591549295
775							
stddev	null	1015.7212254077098		null	null	7.568867638264089	85203.825
44809658		null	70.08857549984492	1573677.1363237544	60.40935143592945		null
null		null	null		null	0.0	2.0669060668646
346							
min	Acura	1-Series	Джип 3 дв.	Бежевый	Бензин	1943	
1000	АКПП	9		15000	1.0 CILQ G Packa...	10HM	
0.5 LTR	2022-08-19 00:00:00	Анучино	https://anuchino....	! ! ! ПЕРЕКУПОВ П...		ОТЛИЧНОЕ СОС	
ТОЯ...							
max	УАЗ	Хантер	Хэтчбек 5 дв.	Черный	Электро	2022	
1000000	Робот	1000		41500000		ГАЗ-М-21Л УМЗ-4218.10	
6.4 LTR	2022-09-26 00:00:00	Ярославский	https://zarubino....		 чем у автомат	
a.Ав...							

Ahora veremos si las columnas tienen el tipo de dato correcto.

```
In [10]: # Obtenemos el tipo de dato de las columnas
for col in df_spark.dtypes:
    print(col[0] + " , " + col[1])
```

```
brand , string
name , string
bodyType , string
color , string
fuelType , string
year , int
mileage , int
transmission , string
power , int
price , int
vehicleConfiguration , string
engineName , string
engineDisplacement , string
date , string
location , string
link , string
description , string
parse_date , string
```

Ya que ahora los tipos de dato son correctos, procederemos a ver qué columnas presentan valores nulos

In [11]:

```
# Ahora contaremos los valores nulos con isnan, when, count y col
from pyspark.sql.functions import isnan, when, count, col
df_spark.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_spark.columns])
```

```
[Stage 4:===== (13 + 2) / 15]
+-----+-----+-----+-----+-----+-----+
|brand|name|bodyType|color|fuelType|year|mileage|transmission|power|price|vehicleConfiguration|engineName|engineDisplacement|date|location|link|description|parse_date|
+-----+-----+-----+-----+-----+-----+
|0    |0    |0    |109734|3560  |410974|14480  |3065   |20887|0    |410974
|412058|420765|        |0    |0    |0    |35737  |0    |      |
+-----+-----+-----+-----+-----+-----+
```

Como la tabla no sale bien, pondré manualmente qué columnas presentan valores nulos y cuántos:

- Color: 109,734
- fuelType: 3,560
- Year: 410,974
- Mileage: 14,480
- Transmission: 3,065
- Power: 20,887
- vehicleConfiguration: 410,974
- engineName: 412,058
- engineDisplacement: 420,765
- description: 35,737

In [12]:

```
# Ahora para obtener los valores únicos, utilizaremos count_distinct
from pyspark.sql.functions import countDistinct
```

```
# Contaremos los valores únicos de cada columna  
df_spark.select([count_distinct(c).alias(c) for c in df_spark.columns]).show()
```

```
[Stage 7:===== (13 + 2) / 15]
+-----+-----+-----+-----+-----+-----+-----+
|brand|name|bodyType|color|fuelType|year|mileage|transmission|power|price|vehicleConfiguration|e
ngineName|engineDisplacement|date|location| link|description|parse_date|
+-----+-----+-----+-----+-----+-----+-----+
|    74|1026|      11|     16|      3|    59|     541|           5|   352|  2986|          7955|
1149|           55|    39|      71|50119|       63606|      2521|
+-----+-----+-----+-----+-----+-----+-----+
```

Como la tabla no sale bien, pondré manualmente los valores únicos de cada columna:

- brand: 67
 - name: 884
 - bodyType: 11
 - color: 16
 - fuelType: 3
 - year: 54
 - mileage: 483
 - transmission: 5
 - power: 323
 - price: 2037
 - vehicleConfiguration: 5620
 - engineName: 897
 - engineDisplacement: 53
 - date: 16
 - location: 69
 - link: 24757
 - description: 28790
 - parse_date: 307

b) ETL

Viendo un poco las columnas que tenemos y las descripciones cortas obtenidas de Kaggle, tenemos información que **no nos va a servir de inicio**, como `parse_date`, `description`, `link`, `location`, `date`, `engineDisplacement`, `engineName`, `vehicleConfiguration` y `year`.

La justificación de quitarlos es que presentan una gran cantidad de datos nulos, no son relevantes para la venta de un automóvil, existen automóviles viejos que pueden ser muy baratos y muy caros a la vez e incluso hay datos que son únicos por registro. Entonces procederemos a hacer un drop de las columnas.

```
In [13]: df_spark = df_spark.drop('year', 'vehicleConfiguration', 'engineName',
```

```
'engineDisplacement', 'link', 'description',
'parse_date', 'date', 'location')
```

Revisamos las columnas que nos quedan, ahora podremos trabajar mejor con estos datos.

In [14]: df_spark.columns

```
Out[14]: ['brand',
'name',
'bodyType',
'color',
'fuelType',
'mileage',
'transmission',
'power',
'price']
```

Procederemos a revisar las columnas con pocos valores únicos y que coincidan con los nulos para ver lo que contienen

In [15]: # Obtenemos los colores de los automóviles
df_spark.select('color').distinct().collect()

```
Out[15]: [Row(color='Бордовый'),
Row(color='Белый'),
Row(color='Золотистый'),
Row(color='Коричневый'),
Row(color=None),
Row(color='Оранжевый'),
Row(color='Серебристый'),
Row(color='Розовый'),
Row(color='Фиолетовый'),
Row(color='Бежевый'),
Row(color='Красный'),
Row(color='Голубой'),
Row(color='Серый'),
Row(color='Желтый'),
Row(color='Зеленый'),
Row(color='Черный'),
Row(color='Синий')]
```

In [16]: # Obtenemos los tipos de combustión de los automóviles
df_spark.select('fuelType').distinct().collect()

```
Out[16]: [Row(fuelType=None),
Row(fuelType='Дизель'),
Row(fuelType='Электро'),
Row(fuelType='Бензин')]
```

In [17]: # Transmisión
df_spark.select('transmission').distinct().collect()

```
Out[17]: [Row(transmission='Механика'),  
Row(transmission='Робот'),  
Row(transmission=None),  
Row(transmission='Вариатор'),  
Row(transmission='Автомат'),  
Row(transmission='АКПП')]
```

Después de un análisis rápido, pude ver que es posible generalizar datos nulos en vez de borrarlos directamente y comenzar a perder información.

Por ejemplo, el color más común de los automóviles es blanco y podemos partir de ahí para llenar los strings nulos o vacíos con Белый .

```
In [18]: df_spark = df_spark.na.fill('Белый', 'color')
```

Ahora, el tipo de combustible que más se uses es la gasolina, siendo la más común de los automóviles, así que llenaré el tipo de combustible con Бензин .

```
In [19]: df_spark = df_spark.na.fill('Бензин', 'fuelType')
```

Revisando el progreso de los valores nulos, ahora sólo quedan las columnas de kilometraje, poder y transmisión.

```
In [20]: df_spark.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_spark.columns])
```

```
[Stage 22:===== (14 + 1) / 15]  
+---+---+---+---+---+---+---+  
|brand|name|bodyType|color|fuelType|mileage|transmission|power|price|  
+---+---+---+---+---+---+---+  
|0 |0 |0 |0 |0 |14480 |3065 |20887|0 |  
+---+---+---+---+---+---+---+
```

Las columnas de poder y kilometraje las podemos reemplazar con valores de la media, el promedio o la moda ya que son columnas con el tipo de dato entero.

```
In [21]: from pyspark.ml.feature import Imputer  
  
imputer = Imputer(  
    inputCols = ['power', 'mileage'],  
    outputCols = ['power', 'mileage'])  
    .setStrategy('mean')
```

```
In [22]: df_spark = imputer.fit(df_spark).transform(df_spark)
```

Quizá sea bueno poner los tipos de transmisión faltantes con la moda por esta ocasión.

Usaré consultas SQL ya que no encontré otra manera de hallar la moda y el Imputer tristemente me da error si trato de hacerlo con tipo de dato de string.

```
In [25]: df_aux = df_spark.where(col('transmission').isNotNull())

df_aux.createOrReplaceTempView('table')
df_aux_2 = spark_session.sql(
    'SELECT transmission, COUNT(transmission) AS count FROM table GROUP BY transmission ORDER BY
')

df_aux_2.show()
```

transmission	count
Вариатор	677023
АКПП	645355
Механика	73331
Автомат	59233
Робот	55193

Podemos observar que el tipo de transmisión más común es el de CVT, (o transmisión continuamente variable), así que se lo asignaremos a los valores nulos.

```
In [26]: df_spark = df_spark.na.fill('Вариатор', 'transmission')
```

Ahora revisaré si ya no tenemos valores nulos en nuestro dataset...

```
In [27]: df_spark.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_spark.columns])
```

Stage 33:=====	(13 + 2) / 15
brand name bodyType color fuelType mileage transmission power price	0 0 0 0 0 0 0 0

Ya que no tenemos ningún dato nulo y puedo decir que ahora mis datos están limpios, guardaré una copia y la insertaré dentro de Tableau más adelante para observar datos de forma visual y darme una idea de cómo están las cosas. 😊

```
In [ ]: # df.write.option("header",true).csv('region25_Less.csv')
```

c) Generando el modelo con MLlib

Ahora vamos a generar un modelo inteligente de clasificación de regresión con el objetivo de predecir el precio de los automóviles en el mercado libre ruso de automóviles.

Primero importamos las herramientas para entrenar con regresión lineal.

```
In [28]: from pyspark.mllib.linalg import SparseVector
from pyspark.mllib.regression import LabeledPoint
```

```
In [31]: from pyspark import SparkContext  
sc = SparkContext.getOrCreate()
```

Ahora separamos los datos de entrenamiento y de prueba

```
In [29]: (trainingData, testData) = df_spark.randomSplit([0.8, 0.2])
```

Procedemos a usar la función de regresión lineal con gradiente descendiente usando los datos de entrenamiento.

```
In [ ]: lrm = LinearRegressionWithSGD.train(sc.parallelize(trainingData), iterations=10,  
initialWeights=np.array([1.0]))
```

4. Evaluando el modelo con PySpark

5. Generando un tablero de visualización con Tableau