Spark Dataframes(14/12/2018)

Carga de datos

In []:
```
tabla=[("Marta",25),
       ("Pedro",30),
       ("María",24)]
df=spark.createDataFrame(tabla,["nombre","edad"])
df.show()
```

Creación de un dataframe con encabezados

Out[]:
```
+------+----+
|nombre|edad|
+------+----+
| Marta|  25|
| Pedro|  30|
| María|  24|
+------+----+
```

Visualización del dataframe

In []:
```
df.printSchema()
```

Muestra la estructura

Out[]:
```
root
 |-- nombre: string (nullable = true)
 |-- edad: long (nullable = true)
```

Estructura del dataframe

In []:
```
from pyspark.sql.types import *

estructura=StructType([
    StructField("nombre",StringType()),
    StructField("edad",ByteType())
])

df2=spark.createDataFrame(tabla,estructura)
```

Creación de un dataframe con estructura

Datos externos

In []:
```
dfVentas=spark.read.json("ventas.json")
dfVentas.show()
```

Carga de un archivo json

Out[]:
```
+--------+----+-------+
|Producto| Año| Precio|
+--------+----+-------+
|    moto|null|1000.00|
|   coche|null|2000.00|
|    bici|null| 200.00|
|   coche|2015|3000.00|
+--------+----+-------+
```

In []:
```
dfVendedores=spark.read.csv("vendedores.csv",
                sep=";",header=True,inferSchema=True)
```

Carga de un archivo csv

Recuperación de información

In []:
```
df.summary().show()
```

Out[]:
```
+-------+------+------------------+
|summary|nombre|              edad|
+-------+------+------------------+
|  count|     3|                 3|
|   mean|  null|26.333333333333332|
| stddev|  null|3.2145502536643185|
|    min| Marta|                24|
|    25%|  null|                24|
|    50%|  null|                25|
|    75%|  null|                30|
|    max| Pedro|                30|
+-------+------+------------------+
```

In []:
```
df.collect()
```

Out[]:
```
[Row(nombre='Marta', edad=25),
 Row(nombre='Pedro', edad=30),
 Row(nombre='María', edad=24)]
```

Devuelve el conjunto como lista

In []:
```
df.count()
```

Registros en el DF

Out[]:
```
3
```

In []:
```
df.select("edad","nombre").show()
```

Selecciona columnas

Out[]:
```
+----+------+
|edad|nombre|
+----+------+
|  25| Marta|
|  30| Pedro|
|  24| María|
+----+------+
```

In []:
```
df.select("nombre",(df.edad+10)\
          .alias("edad ahora")).show()
```

Selecciona y modifica columnas

Out[]:
```
+----+------+
|edad|nombre|
+----+------+
|  25| Marta|
|  30| Pedro|
|  24| María|
+----+------+
```

In []:
```
df.where("edad=30").show()
#df.where(df.edad==30).show()
```

Selecciona filas en función de la condición

Out[]:
```
+------+----+
|nombre|edad|
+------+----+
| Pedro|  30|
+------+----+
```

In []:
```
grupo=dfVentas.groupBy("Producto")
grupo.sum("Precio").show()
```

Agrupa por producto y suma el precio

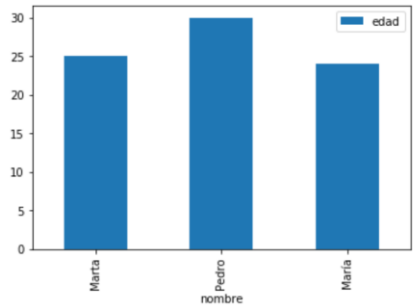
Out[]:
```
+--------+-----------+
|Producto|sum(Precio)|
+--------+-----------+
|   coche|    5000.00|
|    moto|    1000.00|
|    bici|     200.00|
+--------+-----------+
```

Pandas

In []:
```
dfp=df.toPandas()
dfp.plot.bar(x="nombre")
```

Cambia a pandas y realiza una gráfica

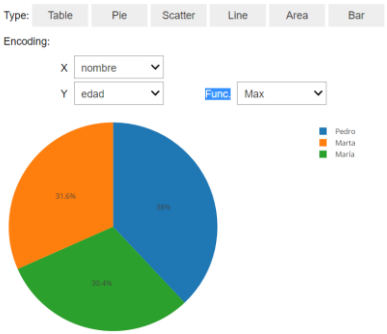
Out[]:



```
In []: from autovizwidget.widget.utils import display_dataframe
       display_dataframe(dfp)
```

Exploración de datos con autovizwidget

Out[]:



DATAFRAMES [doc]

.cache()	Almacena el Dataframe	**.randomSplit**(*weights, seed=None*)	Divide el dataframe aleatoriamente
.collect()	Lista de filas(Row)	**.rdd**	Devuelve RDD
.columns	Nombres de columnas	**.replace**(*to_replace,value=<novalue>, subset=None*)	Realiza reemplazos en el dataframe
.corr(*col1, col2*)	Correlación de Pearson	**.select**(**cols*)	Selecciona columnas
.count()	Número de registros	**.show**(*n=20, truncate=True, vertical=False*)	Muestra el dataframe
.crosstab(*col1, col2*)	Tabla de contingencia	**.stat**	Valores estadísticos
.distinct()	DF sin filas repetidas	**.summary**(**statistics*)	Sumario estadist.
.drop(**cols*)	Descarta columnas	**.take**(*num*)	
.filter(*condition*) **.where**(*condition*)	Selecciona columnas por condición	**.toDF**(**cols*)	Cambia el nombre de las columnas
.groupBy(**cols*)	Agrupa por columnas	**.toPandas**()	Convierte a Pandas
.join(*other, on=None, how=None*)	Combina Dataframes	**.union**(*other*)	Une dos dataframes
.na	Métodos para campos NULL	**.withColumn**(*colName, col*)	Añade o modifica columna
.orderBy(**cols,**kwargs*) **.sort**(**cols, **kwargs*)	Ordena por columnas	**.withColumnRenamed**(*existing, new*)	Cambia el nombre de una columna
.printSchema()	Muestra estructura	**.write**	Graba el DF

GRUPOS [doc] .groupBy(col)

.sum(**cols*)	Totales por grupo	**.min**(**cols*)	Mínimo por grupo
.avg(**cols*) **.mean**(**cols*)	Medias agrupadas	.agg(**exprs*)	Aplica funciones por grupo
.count()	Registros por grupo	.apply(*udf*)	Aplica una función de usuario a cada grupo
.max(**cols*)	Máximo por grupo	.pivot(*pivot_col, values=None*)	Reestructura el dataframe en función de la columna

COLUMNAS [doc]

Formas de indicar columnas: **"nombreCol", df.nombreCol, df["nombreCol"], col("nombreCol")**

.alias(**alias, **kwargs*)	Renombra columna	**.contains**(*other*)	Comprueba contenido
.asc() **.desc()**	Orden ascendente Orden descendente	**.isNotNull**() **.isNull**()	Comprueba no Null Comprueba si Null
.cast(*dataType*)	Convierte tipo de dato	**.like**(*other*) **.rlike**(*other*)	Coincidencia patrón SQL Coincidencia exp. regular
.between(*lower, upper*)	Comprueba intervalo	**.substr**(*start, length*)	Recupera subcadena
.endswith(*other*) **.startswith**(*other*)	Comprueba inicio cadena	**.when**(*condition, value*) **.otherwise**(*value*)	Devuelve un valor u otro si se cumple condición

TIPOS DE DATOS [doc]

DataType	BinaryType	**TimestampType**	**FloatType**	**LongType**	MapType
NullType	BooleanType	**DecimalType**	**ByteType**	**ShortType**	**StructField**
StringType	DateType	**DoubleType**	**IntegerType**	ArrayType	**StructType**

FUNCIONES [doc]

abs	acos	add_months	approxCountDistinctD
ascii	array	array_contains	asc
avg	asin	atan	atan2
broadcast	base64	bin	bitwiseNOT
coalesce	bround	cbrt	ceil
column	**col**	collect_list	collect_set
corr	**concat**	concat_ws	conv
countDistinct	cos	cosh	**count**
create_map	covar_pop	covar_samp	crc32
date_add	cume_dist	current_date	current_timestamp
datediff	date_format	date_sub	date_trunc
decode	dayofmonth	dayofweek	dayofyear
encode	degrees	dense_rank	**desc**
expm1	exp	**explode**	explode_outer
floor	expr	factorial	first
from_unixtime	**format_number**	format_string	from_json
grouping	from_utc_timestamp	get_json_object	greatest
hour	grouping_id	hash	hex
instr	hypot	initcap	input_file_name
kurtosis	isnan	**isnull**	json_tuple
lead	lag	last	last_day
lit	least	length	levenshtein
log1p	locate	log	log10
ltrim	log2	**lower**	lpad
md5	map_keys	map_values	**max**
next_day	**mean**	**min**	minute
posexplode	**month**	**months_between**	nanvl
radians	ntile	pandas_udfE	percent_rank
regexp_extract	posexplode_outer	pow	quarter
rint	rand	randn	rank
rtrim	**regexp_replace**	**repeat**	reverse
shiftLeft	round	row_number	rpad
sin	second	sha1	sha2
sort_array	shiftRight	shiftRightUnsigned	signum
sqrt	sinh	size	skewness
struct	soundex	spark_partition_id	**split**
sumDistinct	**stddev**	stddev_pop	stddev_samp
toRadiansD	substring	substring_index	sum
to_utc_timestamp	tan	tanh	toDegreesD
udf	to_date	to_json	**to_timestamp**
upper	**translate**	trim	trunc
unbase64	unhex	unix_timestamp	weekofyear
var_pop	var_samp	**variance**	when
window	**year**		