SPARK

- Funciones de Spark

```
dataFrame = spark.read.json("path.json")
dataFrame.show()  // dataFrame.printSchema()  // dataFrame.select("name")
dataFrame.select(df['name'], df['age']+1).show()  // df.filter(...) //
df.groupBy("age").count()
```

Para registrar el dataFrame como una vista temporal de SQL: // También llamada tabla

```
--> df.createOrReplaceTempView("people") <--
--> df.createGlobalTempView("people") <--  Este caso es una vista temporal global
```

También podemos ejecutar SQL queries:

```
sqlDF = spark.sql("SELECT * FROM people")
```

Las vistas temporales globales se mantienen incluso entre sesiones

Hay una función en python para repetir una función en cada iteración.

```
map(function, iterables)
```

Es muy potente unido a ~~lambda~~ lambda:

```
x = lambda a: a+10
print(x(5)) // muestra 15
```

Los resultados de las consultas SQL devuelven objetos del tipo DataFrame

La función rdd de spark devuelve el contenido como "objeto. RDD"

Siempre se inician las funciones con:

```
sc = spark.sparkContext  // para cargar el schema de la map
```

Por ejemplo:

```
sc = spark.sparkContext  // Carga el schema
lines = sc.textFile("path.txt") // Carga el archivo y lo guarda como Rows (línea a línea)
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: (p[0], p[1].strip())) // lo convierte a tuples
schemaString = "name age"
fields [StructField(field_name, StringType(), True) for field_name in ...
-- schemaString.split()]
schema = StructType(fields)
schemaPeople = spark.createDataFrame(people, schema) // Apply the schema To RDD
```

→ Generar Vista → SQL query → Print

StructType = define el schema de los Spark DataFrames. StructType contiene una lista de StructField objects que define el nombre, tipo y etiqueta

---

spark.conf.set("spark.sql.execution.arrow.enabled", "true")

// Convierte pandas DataFrame a spark DataFrames con Arrow

de Spark a pandas = df.select("*").toPandas()

de pandas a Spark = ~~df~~ spark.createDataFrame(pdf)

Spark guarda y carga los archivos con formato ".parquet"

Con el método parallelize() podemos trabajar en paralelo

- Si no incluimos parametros por defecto obtendrá uno por CPU de nuestro cluster
- Podemos añadir manualmente la división: parallelize(data, 10)

El 80% - 90% del tiempo de un científico de datos consiste en limpiar los datos (eliminar duplicados, correos inexistentes, códigos de ttpp erroneos, ...)

⊙ Eliminar duplicados → El método distinct()

```
print('# {0}'.format(df.distinct().count()))
print('# {0}'.format(df.count()))
```

Si el resultado de los print son iguales NO HAY DUPLICADOS

Si hay duplicados, usamos el método df.dropDuplicates()

Ya hemos eliminado duplicados, pero puede haber duplicados en ciertas columnas como Id o DNI:

```
print('# {0}'.format(df.select([
    c for c in df.columns if c != 'id']).
... distinct().count()))
```

Podemos eliminar filas que tengan ~~mismo ID y dis~~ distinto ID pero mismos valores, con el código anterior. Sólo habría que cambiar select por subset

Como hemos visto la fila con el id=3 tiene 4 valores missing
Para comprobar los datos de esa fila, sería:

```
df.where('id==3').show()
```

Otra opción para decidir si eliminamos o no una columna, sería aplicar porcentajes con cuántos missing aparecen con el siguiente comando:

```
df.agg(*[
    (1-(fn.count(c)/fn.count('*'))).alias(c+'_missing')
    for c in df.columns]).show()
```

// * El comodín es para sumar todas las filas en el comando count()

Para eliminar las columnas lo que vamos a hacer es crear un conjunto sin esa columna (No me convence):

```
df_SinColumna = df.select([
    c for c in df.columns if c!='income'])
```

© Scenario: Tenemos una tabla con 10 columnas y solo queremos 4, pero quiero que me muestre toda la tabla con las filas que al menos tengan 4 valores excluyendo None, Missing o #N/A! con el siguiente comando:

```
df.dropna(thresh=4).show()
```

Sin embargo si lo que queremos es rellenar esos valores perdidos, en este caso será con la media (sólo para datos numéricos): * sexo es Categorico

```
means = df.agg(*[fn.mean(c).alias(c)
                 for c in df.columns if c!='sexo']
                ).toPandas().to_dict('records')[0]

means['sexo'] = 'missing'
df.fillna(means).show()
```

Para comprobar cuántos ID distintos tenemos sin bucles y con un sólo comando, sería así:

```
import pyspark.sql.functions as Fn
df.agg(
    Fn.count('id').alias('count'),
    Fn.countDistinct('id').alias('distinct')).show()
```

Si nos interesa mantener todas esas filas, pero algunas tienen mismo ID y queremos que ese campo sea único, utilizamos:

- monotonically_increasing_id()

⊙ Campos vacíos

+ Por general se suelen diminas los campos vacíos pero hay que te cuidado de no diminas demasiadas. En el caso de reducir el datas a más de un 50%, habría que plantear otras decisiones cómo q campos son y si se podrían obtener.
+ Otra acción es la de cambias None por algún valor. Si los campos son booleanos o categóricos, podría añadirse otra categ (Missing)
+ Si los datos son numéricos o ordinales, podría aplicasse la med o la mediana o algún valor predefinido

Un comando para comprobar cuántos None aparecen por filas (Nones las columnas) sería:

```
df.rdd.map(
    lambda row: (row['id'], sum([c == None for c in row]
).collect()
```

Devolvería:

Output: [(1,0), (2,1), (3,4)...]

Esto quiere decir:
Fila 1 = No tiene None en sus columnas
Fila 2 = 1 None en sus colu
Fila 3 = 4 None....

Una vez que hemos limpiado los datos, nuestro siguiente paso sería interpretar los datos. y repetir la limpieza

⊙ **Estadística Descriptiva**

Mostrará cuantos no-missing values hay, la media, la desviación standard por columna y los valores máximos y mínimos.

Borraremos la cabecera del archivo .csv y convertiremos cada elemento a número entero.

```
fraud = sc.textFile('Fraud.csv')
header = fraud.first()
fraud = fraud \
        .filter(lambda row: row != header) \
        .map(lambda row: [int(elem) for elem in
                          row.split(',')])
```

A continuación crearemos el schema para nuestro DataFrame

```
fields = [
    *[
        typ.StructField(h[1:-1], typ.IntegerType(), True)
        for h in header.split(',')
    ]
]
schema = typ.StructType(fields)
```

A continuación crearemos el DataFrame, pero hay que tener en cuenta que tenemos campos categóricos

```
fraudP = spark.createDataFrame(fraud, schema)
fraudP.printSchema()
```

* Consejo: ver si los max y min son mucho más distintos que la media
  El coeficiente de variación (la media de desv. standard) es muy alto (cerca o mayor de 1), es decir, amplia difusión de observaciones

Valores Extremos (Outliers)

IQR = Interquartile range. Es calculado cómo la diferencia entre e
Q1 = 25% mayor y menor quartiles
Q3 = 75%

Se podría aceptar que no hay outliers si los valores cumplen que están entre Q1 - 1.5 IQR y Q3 + 1.5 IQR. Esto se podría calcular con el siguiente código.

```
cols = ['weight', 'height', 'age']
bounds = {}
for col in cols:
    quantiles = df.approxQuantile(
        col, [0.25, 0.75], 0.05)
    IQR = quantiles[1] - quantiles[0]
    bounds[col] = [
        quantiles[0] - 1.5 * IQR,
        quantiles[1] + 1.5 * IQR
]
// Para detectar outliers vamos a copiar la tabla pero con valores bodeanos para saber que filas son
outliers = df.select(*['id'] + [
    (
        (df[c] < bounds[c][0]) |
        (df[c] > bounds[c][1])
    ).alias(c + '_o') for c in cols
])
outliers.show()
// Para saber los valores y a qué id pertenece:
df = df.join(outliers, on='id')
df.filter('weight_o').select('id', 'weight').show()
df.filter('age_o').select('id', 'age').show
```

Correlaciones

Otro modelo de relación mutua entre características es la correlación. Incluyendo si son correlativas entre ellas. El método .corr() soporta el coeficiente de correlación de Pearson. Para cr una matriz de correlación, hay que seguir este script.

```
n_numerical = len(numerical)
corr = []
for i in range (0, n_numerical):
    temp = [None] * i
    for j in range (i, n_numerical):
        temp.append (fraudf. corr (numerical [i], numerical[j]
    corr. append (temp)
```

Si los datos no se acercan al 1 o al 0, quiere decir que la relación entre fraude y las características numericales es inexistente. No quiere decir que no debamos usarlas en nuestro modelo

Para visualización de datos, hay muchas opciones pero aquí se va a hablar de matplotlib que están preinstalados con Anaconda.

Primero hay que cargar los módulos

```
%matplotlib inline
import matplotlib. pyplot as plt
plt. style. use ('ggplot')

import bokeh. charts as chrt
from bokeh. io import output_notebook
output_notebook ()
```

Por general, los gráficos más usados son:

- Histogramas del balance
- Scatter con el género (categórico) de al menos 3 fracciones (0.02%, ...)

Y empezamos con la chicha ¡¡

Spark tiene la librería MLlib para machine learning.

Para ML en alto nivel tenemos 3 funcionalidades:

- Data preparation: Extracción, transformación, selección, hashing of categorical features y algunos métodos de NLP
- Algoritmos de ML
- Utilidades: Métodos estadísticos como estadística descriptiva, chi-square testing, linear algebra (sparse and dense matrices and vectors) y ~~modelos~~ métodos de evaluación de modelos

Raramente, nuestro modelo debería de ser lo mejor que podemos hacer.

El concepto de parameter hyper-tuning es encontrar los mejores parametros del modelo: cómo puede ser el máximo de iteraciones en regresión o la profundidad máxima de un árbol de decisiones

- Grid Search: Es un modelo exhaustivo que busca a través de una lista de valores parametrizados, estima modelos separados y elige el mejor dando una métrica de evaluación. Dependiendo de los parametros si son demasiados tomará mucho tiempo.

Leer el libro de PySpark para coger los códigos

- Train-validation splitting: Selecciona el mejor modelo, divide entre valores aleatorios creando un train dataset que a su vez está dividido en smaller training and validation subsets.

Latent Dirichlet Allocation (LDA) es un modelo de NLP

⑥ Streaming Spark

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
⋮
```

```
:
sc = SparkContext("local[2]", "NetworkWordCount")
// la linea de arriba trabajará con 2 hilos
ssc = StreamingContext(sc, 1) // creará un batch interval de
lines = ssc.socketTextStream("localhost", 9999)
// la linea de arriba crea DStream que conecta a localhost:99
```

El siguiente código consiste en contar palabras:

```
words = lines.flatMap(lambda line: line.split(" "))
pairs = words.map(lambda word: (word, 1))   // cada palabra pos ba
wordsCounts = pairs.reduceByKey(lambda x, y: x+y)
wordsCounts.pprint()   // Muestra los 10 primeros de cada RDD
ssc.start()            // Empieza
ssc.awaitTermination() // Espera hasta que acabe
```

Para este caso cada segundo va ir mostrando palabra a palabra (por linea) pero si nosotros lo queremos agregado, es decir, que por cada linea se vaya agregando tenemos que modificar el TIME WINDOW.

En el mismo libro (pag 317 approx.) encontramos el código para hacer lo anterior, pero básicamente sería crear checkpoints y al actualizar datos con una suma entre valor nuevo y antiguo o seguir añadiendo modificando el anterior.

Todo esto ha sido solucionado con estructuras de Streaming con el siguiente código

```
lines = spark\
        .readStream\
        .format('socket')\
        .option('host', 'localhost')\
        .option('port', 9999)\
        .load()

words = lines.select(
   explode(
        split(lines.value, ' ')
   ).alias('word')
   )
wordCounts = words.groupBy('word').count()
```

Solo nos quedaría hacer la llamada

```
query = wordsCounts\
        .writeStream\
        .outputMode('complete')\
        .format('console')\
        .start()
query.awaitTermination()
```

Basilea I (Relacionado con bancos)

- Riesgos: garantizar solvencia del banco

Basilea II: Acuerdo internacional para garantizar ~~[illegible]~~ la solvencia del sistema financiero. Es una normativa de cumplimiento obligado para todos los bancos. Enfocado para la gestión de riesgos.

Para ello hay que guardar un capital (unas reservas) por cada operación que realiza el banco.

P. eje. Hay que guardar el 8% y x el APR

→ APR = Activos ponderados por riesgo

Capital regulatorio es el dinero que tiene que guardar el banco para evitar la quiebra

BIII = 11%

Lo que el banco presta se llama exposición

Capital regulatorio = exposición × 8% × APR

APR = % de la exposición definido según el perfil de riesgos de cliente y operación

- Enfoque del banco

Ejemplo

	Hipoteca	Tarj. Crédito	Préstamo	Estado	Particulares
APR	Bajo	Muy Alto	Alto	Bajo	Alto
	Por su garantía real (Te quitan la casa)	No garantía	Garantía Personal		

Capital Santander = Σ capital de cada cartera

Cada cartera = $\sum_{\text{operación}}$ "x" × 8 × APR

Agregación del capital tiene en cuenta la granularidad y la diversificación y la correlación

- Granularidad = Muchas ops. pequeñas por grupos heterogeneos entre sí y cada componente de ese grupo es homogeneo con los demás componentes

Los llaman buckets

Banca mayorista = ~~Grandes~~ clientes, pocas operaciones con mucho volumen

Banca retail = Clientes muchas operaciones con poco volumen