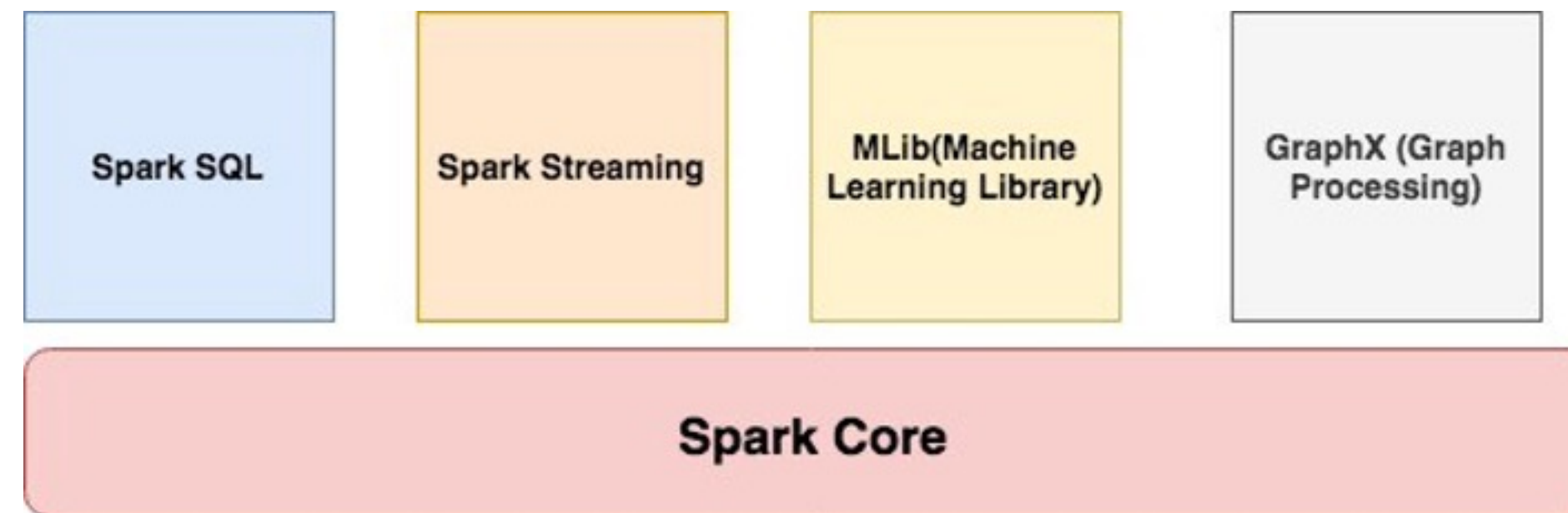


DATABRICKS

Inteligencia de Negocio
13 de octubre de 2023

SPARK



- Apache Spark es un framework de computación (entorno de trabajo) en clúster en código abierto. Proporciona una interfaz para la programación de clusters completos con Paralelismo de Datos implícito y tolerancia a fallos.
- Se puede considerar un sistema de computación en clúster de propósito general y orientado a la velocidad. Proporciona APIs en Java, Scala, Python y R. También proporciona un motor optimizado que soporta la ejecución de gráficos en general. También soporta un conjunto extenso y rico de herramientas de alto nivel entre las que se incluyen Spark SQL (para el procesamiento de datos estructurados basada en SQL), MLlib para implementar machine learning, GraphX para el procesamiento de graficos y Spark Streaming.

PYSPARK Y DATABRICKS


- Databricks es un entorno que permite utilizar notebooks en PySpark para definir los pipelines con el flujo de datos en los procesos ETL
- Usamos la Community Edition de Databricks, que permite crear clusters Spark gratuitos.
- Para poder conectar Databricks con herramientas de acceso como Power Bi de Microsoft necesitaríamos un conector que solamente está disponible en las ediciones de pago de Databricks
- <https://docs.databricks.com/en/getting-started/community-edition.html>


NUEVA CUENTA EN COMMUNITY EDITION


- <https://community.cloud.databricks.com/login.html>



Choose a cloud provider 2/2

 Amazon Web Services

 Microsoft Azure

 Google Cloud Platform

Continue

By clicking "Get Started," you agree to the [Privacy Policy](#) and [Terms of Service](#).

Don't have a cloud account?

Community Edition is a limited Databricks environment for personal use and training.

Get started with Community Edition →

By clicking "Get started with Community Edition," you agree to the [Privacy Policy](#) and [Terms of Service](#) undefined.

SUBIDA DE DATOS

- Subimos archivo "game_skater_stats.csv"

Databricks

You're using Databricks Community Edition. Upgrade for unlimited clusters and collaboration features.Upgrade

Data Science & Engineering

Notebook

Create a new notebook for querying, data processing, and machine learning.

Create a notebook

AutoML

Quickly train ML models for discovery and iteration.

Start AutoML

Transform data

dbt Core

Data import

Quickly import data, preview its schema, create a table, and query it in a notebook.

Browse files

Guide: Quickstart tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5...
Start tutorial

Create New Table

Data source ?

Upload File

S3Other Data Sources

DBFS Target Directory ?

/FileStore/tables/

Select

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

Files ?

game_skater_s

30.4 MB

Cancel upload

NOTEBOOK

- Creamos un notebook con acceso a la table
- Cambiamos infer_schema y first_row_is_header a True
- Borramos celdas de la tercera en en adelante

2023-09-27 - DBFS ExamplePython

FileEditViewRunHelpLast edit was 41 minutes agoProvide feedback

Run allConnectSharePublish

NEW

Overview

This notebook will show you how to create and query a table or DataFrame that you uploaded to DBFS. **DBFS** is a Databricks File System that allows you to store data for querying inside of Databricks. This notebook assumes that you have a file already inside of DBFS that you would like to read from.

This notebook is written in **Python** so the default cell type is Python. However, you can use different languages by using the `%LANGUAGE` syntax. Python, Scala, SQL, and R are all supported.

Cmd 1

Python

```
1 # File location and type
2 file_location = "/FileStore/tables/game_skater_stats.csv"
3 file_type = "csv"
4
5 # CSV options
6 infer_schema = "false"
7 first_row_is_header = "false"
8 delimiter = ","
9
10 # The applied options are for CSV files. For other file types, these will be ignored.
11 df = spark.read.format(file_type) \
12     .option("inferSchema", infer_schema) \
13     .option("header", first_row_is_header) \
14     .option("sep", delimiter) \
15     .load(file_location)
16
17 display(df)
```

Cmd 2

Python

```
1 # Create a view or table
2
3 temp_table_name = "game_skater_stats_csv"
4
5 df.createOrReplaceTempView(temp_table_name)
```

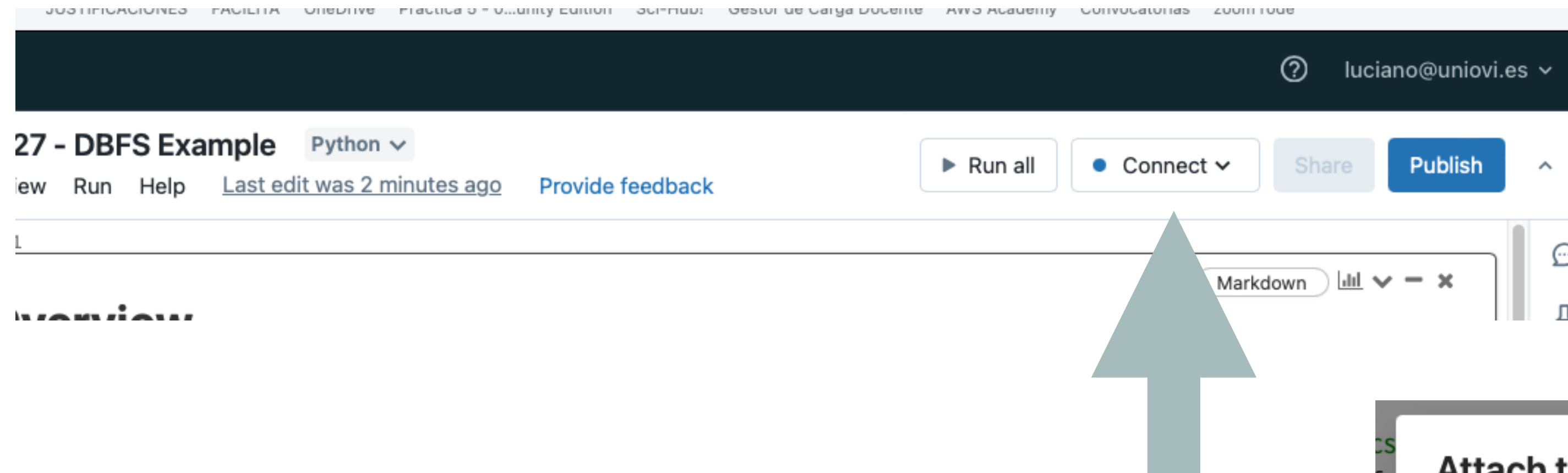
Cmd 3

Python

Cmd 4

Python

SOLICITAMOS CLUSTER DE CÓMPUTO



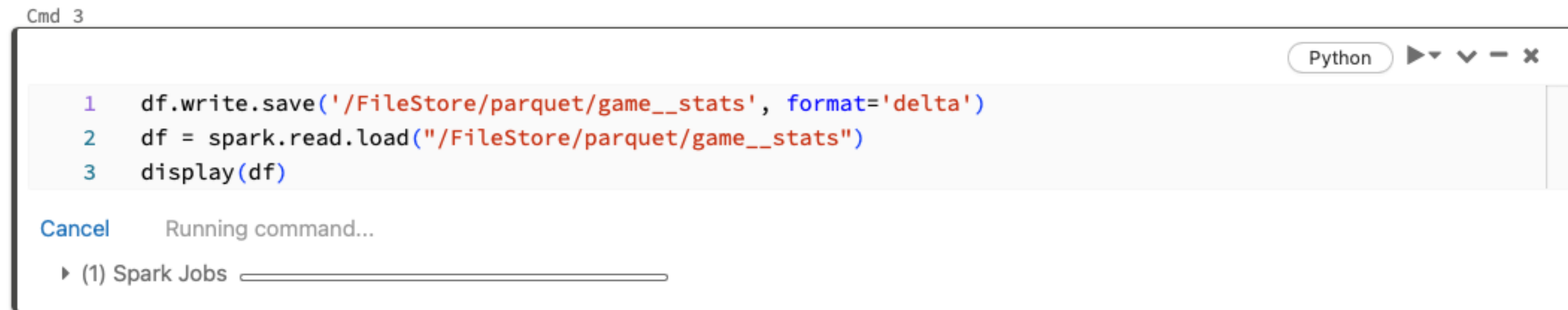
- Conectamos con un cluster (o creamos uno si es necesario)

A modal dialog box titled 'Attach to a compute resource' with a close button (X) in the top right corner. The dialog contains the text: 'Your notebook must be attached to a compute resource to execute commands. Create new compute to run your command.' Below this, there are two main sections: 'Name' with a text input field containing 'visualizacion', and 'Runtime' with a dropdown menu showing 'Runtime: 12.2 LTS (Scala 2.12, Spark 3.3.2)'. At the bottom left, there is a link for 'Advanced Configuration'. At the bottom right, there are two buttons: 'Cancel' and 'Create, Attach, & Run'.

EJECUTAMOS CELDAS Y AÑADIMOS CONTENIDO

- Ver archivo `ParteGuiada.py` con el contenido de las celdas que iremos pegando en el notebook
- Las celdas no se ejecutan en un único servidor, sino que se paraleliza el trabajo entre las máquinas del cluster
- Tampoco usamos Dataframes en memoria, sino que se accede a un sistema de archivos que también está paralelizado
- Es posible convertir una vista de la base de datos en un Dataframe Pandas, pero en ese momento se restringe la ejecución a un único nodo (el que tiene el Dataframe en el memoria) y se pierde la capacidad de Spark de trabajar en paralelo con grandes volúmenes de datos

CARGAR CSV EN LA BASE DE DATOS DISTRIBUIDA




```
Cmd 3
Python ▶ ▼ - ✕
1 df.write.save('/FileStore/parquet/game__stats', format='delta')
2 df = spark.read.load("/FileStore/parquet/game__stats")
3 display(df)
Cancel Running command...
▶ (1) Spark Jobs
```

- Se graba el contenido del objeto df en formato delta
- <https://docs.databricks.com/en/introduction/delta-comparison.html>
- df no es un Dataframe pandas, sino un objeto pySpark con un interfaz de acceso similar (pero no exactamente igual)
- df tampoco está en memoria en un nodo; es un enlace a la base de datos distribuida

VECTORASSEMBLER

Python ▶ ▼ - ✕

```
1 # Create a vector representation for features
2 assembler = VectorAssembler(inputCols=['shots', 'hits', 'assists',
3   |   'penaltyMinutes', 'timeOnIce', 'takeaways'], outputCol="features")
4 train_df = assembler.transform(df)
```

▶  train_df: pyspark.sql.dataframe.DataFrame = [game_id: integer, player_id: integer ... 21 more fields]

Command took 0.34 seconds -- by luciano@uniovi.es at 27/09/2023, 11:06:25 on visualizacion

- En sklearn, la mayoría de los modelos pueden tomar DataFrames sin procesar como entrada para el entrenamiento. En un entorno distribuido es un poco más complicado: se utilizan Assemblers para preparar nuestros datos de entrenamiento.
- VectorAssembler (librería Spark ML) es un módulo que permite convertir características numéricas en un único vector que es utilizado por los modelos de machine learning.
- A modo de resumen, lo que hace es tomar una lista de columnas (características) y combinarla en una única columna vectorial (vector de características). A continuación, se utiliza como entrada en los modelos de aprendizaje automático en Spark ML.

MODELO DE MACHINE LEARNING



The image displays two screenshots of a Jupyter Notebook interface. The top screenshot shows the first three lines of code: a comment, the creation of a LinearRegression object with 'features' as the feature column and 'goals' as the label column, and the fitting of the model to the training data. The bottom screenshot shows the next four lines of code: generating a training summary, and printing the coefficients, RMSE, and R2 values. The output of these commands is shown below the code, displaying the numerical results for each metric.

```
1 # Fit a linear regression model
2 lr = LinearRegression(featuresCol = 'features', labelCol='goals')
3 lr_model = lr.fit(train_df)
```

► (2) Spark Jobs

Command took 14.92 seconds -- by luciano@uniovi.es at 27/09/2023, 11:09:11 on visualizacion

Cmd 8

```
1 trainingSummary = lr_model.summary
2 print("Coefficients: " + str(lr_model.coefficients))
3 print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
4 print("R2: %f" % trainingSummary.r2)
```

Coefficients: [0.09353643668143533,-0.006632536309343135,0.005487355356078061,-0.000177792345791571,-4.317801186031907e-05,0.017566692293079472]
RMSE: 0.378003
R2: 0.126676

Command took 0.20 seconds -- by luciano@uniovi.es at 27/09/2023, 11:11:58 on visualizacion

- Se le indica al algoritmo de ML el nombre del vector que contiene los datos de entrada ('features') y se le proporciona el nombre del vector que contendrá los datos de salida ('goals')

USO DE PIPELINES (ETL)

- Un pipeline es una secuencia de transformaciones realizadas sobre unos datos
- En este ejemplo, el pipeline contiene la creación del VectorAssembler y el aprendizaje del modelo

```
1  # Regresion mediante arboles de decision
2  from pyspark.ml import Pipeline
3  from pyspark.ml.regression import RandomForestRegressor
4  from pyspark.ml.evaluation import RegressionEvaluator
5  (trainingData, testData) = df.randomSplit([0.7, 0.3])
6
7  # Crea un dataframe con las columnas que necesitamos
8  assembler = VectorAssembler(inputCols=['shots', 'hits', 'assists',
9  |   'penaltyMinutes', 'timeOnIce', 'takeaways'], outputCol="features")
10
11 # Modelo de regresion
12 rf = RandomForestRegressor(featuresCol='features', labelCol='goals')
13
14 # Pipeline: assembler -> random forest
15 pipeline = Pipeline(stages=[assembler, rf])
16
17 # Lanzamos el pipeline
18 model = pipeline.fit(trainingData)
19
20 # Evaluacion en el conjunto de test
21 predictions = model.transform(testData)
22
23 # Medicion del ajuste
24 evaluator = RegressionEvaluator(
25 |   labelCol="goals", predictionCol="prediction", metricName="rmse")
26 rmse = evaluator.evaluate(predictions)
27
28 evaluator2 = RegressionEvaluator(
29 |   labelCol="goals", predictionCol="prediction", metricName="r2")
30 r2 = evaluator2.evaluate(predictions)
31 print("R2 on test data = %g" % r2)
32
```

▶ (10) Spark Jobs

- ▶ trainingData: pyspark.sql.dataframe.DataFrame = [game_id: integer, player_id: integer ... 20 more fields]
- ▶ testData: pyspark.sql.dataframe.DataFrame = [game_id: integer, player_id: integer ... 20 more fields]
- ▶ predictions: pyspark.sql.dataframe.DataFrame = [game_id: integer, player_id: integer ... 22 more fields]

R2 on test data = 0.123073

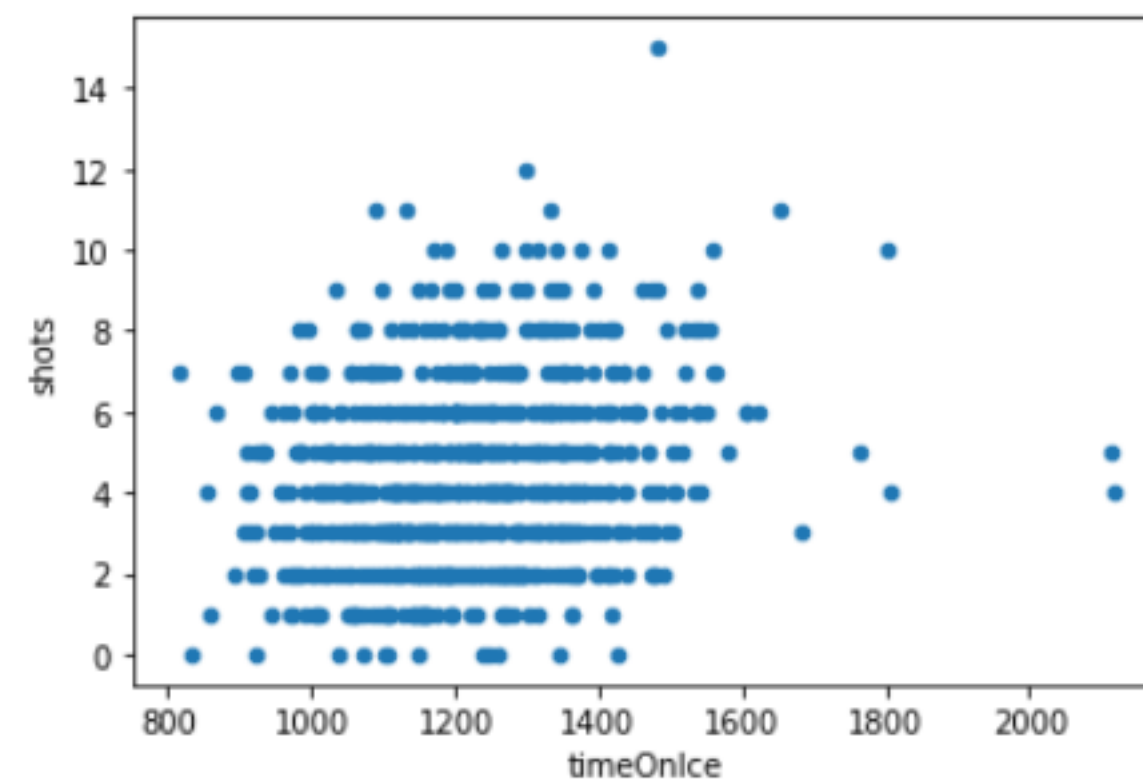
Command took 40.06 seconds -- by luciano@uniovi.es at 27/09/2023, 11:13:53 on visualizacion

EXPORTACIÓN A PANDAS

- sample_pd es un DataFrame pandas, las operaciones que se realicen sobre él no se paralelizan

```
1 sample_pd.plot(x="timeOnIce",y="shots",kind="scatter")
```

Out[22]: <AxesSubplot:xlabel='timeOnIce', ylabel='shots'>



Command took 0.41 seconds -- by luciano@uniovi.es at 27/09/2023, 11:22:43 on visualization

Python

```
1 # DBTITLE 1,Pandas UDF
2 df.createOrReplaceTempView("stats")
3
4 # COMMAND -----
5
6 # Sample data for a player
7 sample_pd = spark.sql("""
8     select * from stats
9     where player_id = 8471214
10 """).toPandas()
11
12 display(sample_pd)
```

▶ (1) Spark Jobs

Table ▾ +

| | game_id | player_id | team_id | timeOnIce | assists | goals | shots | hits | pos |
|---|------------|-----------|---------|-----------|---------|-------|-------|------|-----|
| 1 | 2015020519 | 8471214 | 15 | 1157 | 0 | 0 | 2 | 1 | 0 |
| 2 | 2017020479 | 8471214 | 15 | 1245 | 1 | 1 | 4 | 0 | 0 |
| 3 | 2017020695 | 8471214 | 15 | 1308 | 0 | 0 | 2 | 4 | 0 |
| 4 | 2015020558 | 8471214 | 15 | 1205 | 0 | 1 | 6 | 4 | 0 |
| 5 | 2017020654 | 8471214 | 15 | 1165 | 1 | 0 | 2 | 4 | 0 |
| 6 | 2016020719 | 8471214 | 15 | 1261 | 0 | 0 | 7 | 5 | 0 |
| 7 | 2016020380 | 8471214 | 15 | 1317 | 0 | 0 | 1 | 1 | 0 |

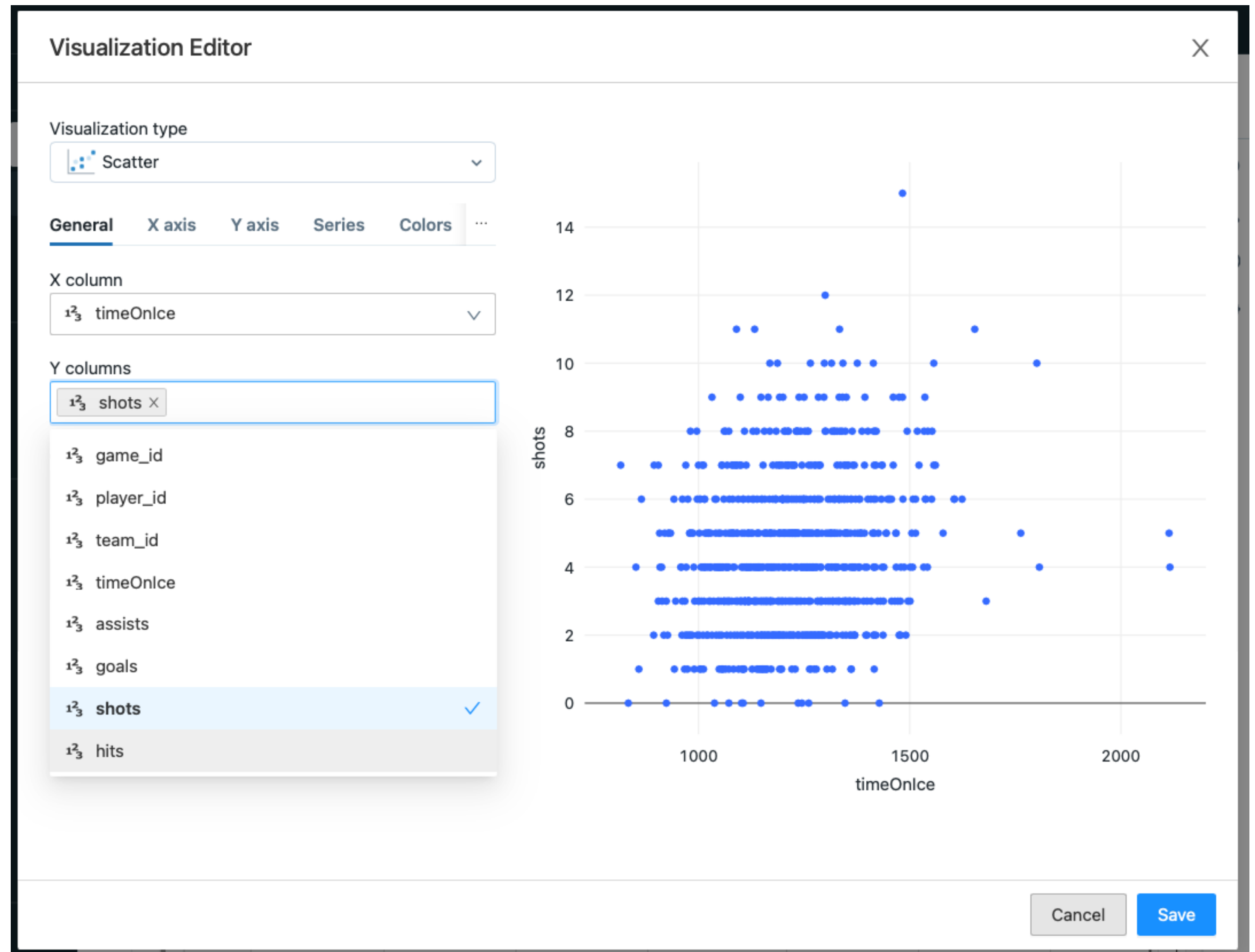
788 rows | 5.14 seconds runtime

Refreshed now

Command took 5.14 seconds -- by luciano@uniovi.es at 27/09/2023, 11:18:26 on visualization

VISUALIZACIONES EN DATABRICKS

- En la ventana de visualización de cualquier consulta a la base de datos se puede lanzar el editor de visualizaciones
- Las visualizaciones pueden añadirse a dashboards



```
1 sample_spark = spark.sql("""
2     select * from stats
3     where player_id = 8471214
4 """)
5
6 display(sample_spark)
7
```

▼ (3) Spark Jobs

- ▶ Job 39 [View](#) (Stages: 1/1)
- ▶ Job 40 [View](#) (Stages: 1/1)
- ▶ Job 41 [View](#) (Stages: 1/1)


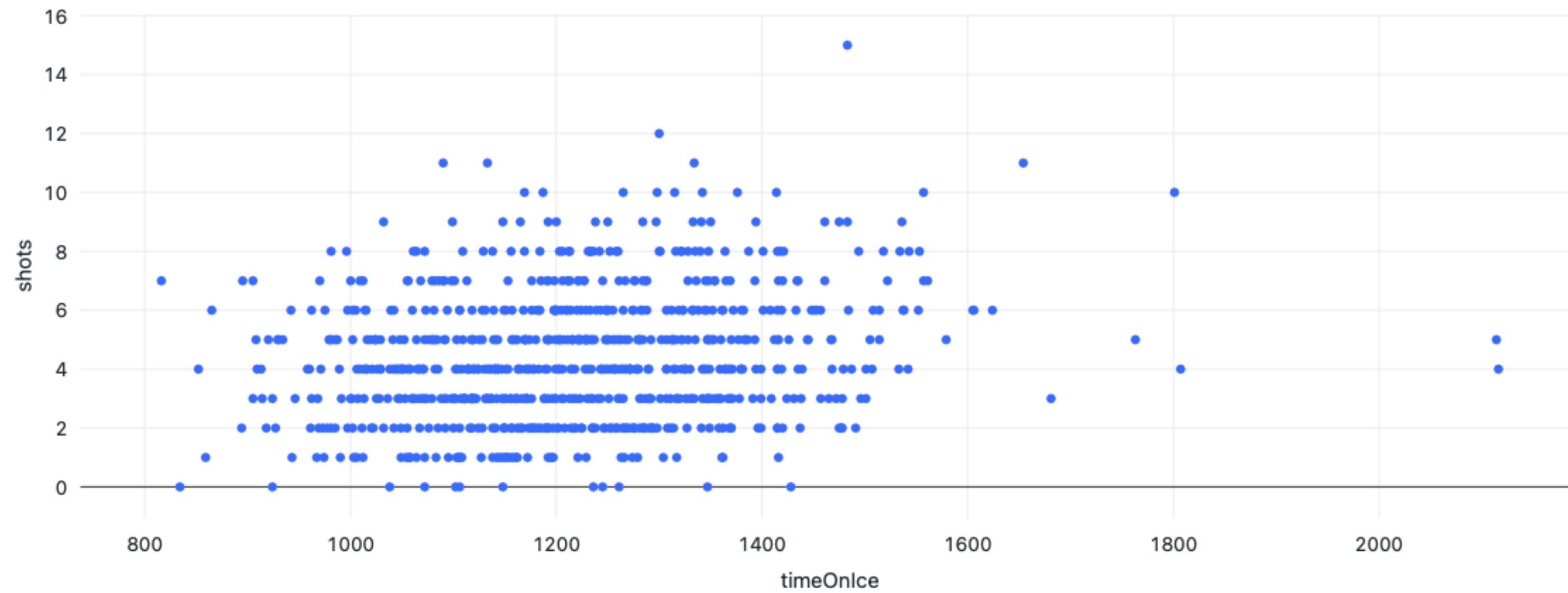
▶  sample_spark: pyspark.sql.dataframe.DataFrame = [game_id: integer, player_id: integer ... 20 more fields]

Table [Visualization 1](#) ▼ +



 Edit Visualization

788 rows

Refreshed 3 minutes ago

Command took 2.33 seconds -- by luciano@uniovi.es at 27/09/2023, 11:36:07 on visualizacion

Your dashboard view is untitled! Please enter a title here.

Untitled



View of notebook: [2023-09-27 - DBFS Example](#)

Present Dashboard

Layout option:

Stack

Float

Dashboard width:

1024px



Remove all graphs

Delete this dashboard

