Overview

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

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

2 minutes ago (1s) 2

```
# File location and type
file_location = "/FileStore/tables/ratings.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 2 more fields]

Table + New result table: OFF

	userId	movieId	rating	timestamp
1	1	1	4	964982703
2	1	3	4	964981247
3	1	6	4	964982224
4	1	47	5	964983815
5	1	50	5	964982931
6	1	70	3	964982400
7	1	101	5	964980868

10,000 rows | Truncated data | 1.42 seconds runtime

2 minutes ago (<1s) 3

```
dbutils.fs.ls ("/FileStore/tables")
```

```
[FileInfo(path='dbfs:/FileStore/tables/genome_tags.csv', name='genome_tags.csv', size=18103, modificationTime=1713629413000),
 FileInfo(path='dbfs:/FileStore/tables/links-1.csv', name='links-1.csv', size=197979, modificationTime=1713629849000),
 FileInfo(path='dbfs:/FileStore/tables/links.csv', name='links.csv', size=1368578, modificationTime=1713629324000),
 FileInfo(path='dbfs:/FileStore/tables/movies.csv', name='movies.csv', size=494431, modificationTime=1713629849000),
 FileInfo(path='dbfs:/FileStore/tables/ratings.csv', name='ratings.csv', size=2483723, modificationTime=1713629860000),
 FileInfo(path='dbfs:/FileStore/tables/tags-1.csv', name='tags-1.csv', size=118660, modificationTime=1713629855000),
 FileInfo(path='dbfs:/FileStore/tables/tags.csv', name='tags.csv', size=118660, modificationTime=1713629849000),
 FileInfo(path='dbfs:/FileStore/tables/u.data', name='u.data', size=1979173, modificationTime=1713627135000)]
```

2 minutes ago (<1s) 4

```
# Create a view or table

temp_table_name = "ratings_csv"
```

```
df.createOrReplaceTempView(temp_table_name)
```

2 minutes ago (<1s) 5

```
%sql

/* Query the created temp table in a SQL cell */

select * from `ratings_csv`
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 2 more fields]

Table + New result table: OFF

	userId	movieId	rating	timestamp
1	1	1	4	964982703
2	1	3	4	964981247
3	1	6	4	964982224
4	1	47	5	964983815
5	1	50	5	964982931
6	1	70	3	964982400
7	1	101	5	964980868

10,000 rows | Truncated data | 0.40 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[209]` . Learn more

2 minutes ago (<1s) 6

```
#Read links.csv file

df = spark.read.format("csv").option("header",True).load("/FileStore/tables/links.csv")
```

(1) Spark Jobs

```
df: pyspark.sql.dataframe.DataFrame
  movieId: string
  imdbId: string
  tmdbId: string
```

2 minutes ago (<1s) 7

```
df.show()
```

(1) Spark Jobs

```
+-------+-------+------+
|movieId| imdbId|tmdbId|
+-------+-------+------+
|      1|0114709|   862|
|      2|0113497|  8844|
|      3|0113228| 15602|
|      4|0114885| 31357|
|      5|0113041| 11862|
|      6|0113277|   949|
|      7|0114319| 11860|
|      8|0112302| 45325|
|      9|0114576|  9091|
|     10|0113189|   710|
|     11|0112346|  9087|
|     12|0112896| 12110|
|     13|0112453| 21032|
|     14|0113987| 10858|
|     15|0112760|  1408|
|     16|0112641|   524|
|     17|0114388|  4584|
|     18|0113101|     5|
```

✓ 2 minutes ago (<1s) 8

```
import datetime
import pyspark.sql.functions as f
import pyspark.sql.types
import pandas as pd

from pyspark.sql.functions import year, month, dayofmonth
from pyspark.sql.functions import unix_timestamp, from_unixtime
from pyspark.sql import window
from pyspark.sql.functions import rank,min
```

✓ 2 minutes ago (<1s) 9

```
#File location and type

file_location = "/FileStore/tables/movies.csv"

file_type = "csv"

#csv options

infer_schema = "true"
first_row_is_header = "true"
delimiter = ','


df_movies = spark.read.format(file_type)\
        .option("inferSchema",infer_schema)\
        .option("header",first_row_is_header)\
        .option("sep",delimiter)\
        .load(file_location)

display(df_movies)
```

(3) Spark Jobs

df_movies: pyspark.sql.dataframe.DataFrame = [movieId: integer, title: string ... 1 more field]

Table | New result table: OFF

	movieId	title
1	1	Toy Story (1995)
2	2	Jumanji (1995)
3	3	Grumpier Old Men (1995)
4	4	Waiting to Exhale (1995)
5	5	Father of the Bride Part II (1995)
6	6	Heat (1995)
7	7	Sabrina (1995)

9,742 rows | 0.49 seconds runtime

✓ 2 minutes ago (<1s) 10

```
#create a View or Table

temp_table_name = "movies_csv"
df_movies.createOrReplaceTempView(temp_table_name)
```

✓ 2 minutes ago (<1s) 11

```
%sql

select * from movies_csv;
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [movieId: integer, title: string ... 1 more field]

Table + New result table: OFF

	movieId	title
1	1	Toy Story (1995)
2	2	Jumanji (1995)
3	3	Grumpier Old Men (1995)
4	4	Waiting to Exhale (1995)
5	5	Father of the Bride Part II (1995)
6	6	Heat (1995)

9,742 rows | 0.19 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[215]` . Learn more

+ +

▶ ✓ 2 minutes ago (1s) 12

```
#Read links file


df_links = spark.read.format(file_type)\
            .option("inferSchema",infer_schema)\
            .option("header",first_row_is_header)\
            .option("sep",delimiter)\
            .load("/FileStore/tables/links.csv")
display(df_links)
```

(3) Spark Jobs

df_links: pyspark.sql.dataframe.DataFrame = [movieId: integer, imdbId: integer ... 1 more field]

Table + New result table: OFF

	movieId	imdbId	tmdbId
1	1	114709	862
2	2	113497	8844
3	3	113228	15602
4	4	114885	31357
5	5	113041	11862
6	6	113277	949
7	7	114319	11860

10,000 rows | Truncated data | 0.59 seconds runtime

+ +

▶ ✓ 2 minutes ago (<1s) 13

```
#Read tags file

df_tags = spark.read.format(file_type)\
            .option("inferSchema",infer_schema)\
            .option("header",first_row_is_header)\
            .option("sep",delimiter)\
            .load("/FileStore/tables/tags.csv")
display(df_tags)
```

(3) Spark Jobs

df_tags: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 2 more fields]

Table + New result table: OFF

	userId	movieId	tag	timesta
1	2	60756	funny	1445714
2	2	60756	Highly quotable	1445714
3	2	60756	will ferrell	1445714
4	2	89774	Boxing story	1445715
5	2	89774	MMA	1445715
6	2	89774	Tom Hardy	1445715

7	2	106782	drugs	1445715

3,683 rows | 0.49 seconds runtime

2 minutes ago (1s) 14

```
#Read ratings file

df_ratings = spark.read.format(file_type)\
        .option("inferSchema",infer_schema)\
        .option("header",first_row_is_header)\
        .option("sep",delimiter)\
        .load("/FileStore/tables/ratings.csv")

display(df_ratings)
```

(3) Spark Jobs

df_ratings: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 2 more fields]

Table + New result table: OFF

	userId	movieId	rating	timestamp
1	1	1	4	964982703
2	1	3	4	964981247
3	1	6	4	964982224
4	1	47	5	964983815
5	1	50	5	964982931
6	1	70	3	964982400
7	1	101	5	964980868

10,000 rows | Truncated data | 0.79 seconds runtime

2 minutes ago (<1s) 15

```
#count of records

df_movies.count()
display(df_movies)
```

(3) Spark Jobs

Table + New result table: OFF

	movieId	title
1	1	Toy Story (1995)
2	2	Jumanji (1995)
3	3	Grumpier Old Men (1995)
4	4	Waiting to Exhale (1995)
5	5	Father of the Bride Part II (1995)
6	6	Heat (1995)

9,742 rows | 0.39 seconds runtime

2 minutes ago (<1s) 16

```
df_links.count()
display(df_links)
```

(3) Spark Jobs

Table + New result table: OFF

	movieId	imdbId	tmdbId
1	1	114709	862
2	2	113497	8844
3	3	113228	15602

4	4	114885	31357
5	5	113041	11862
6	6	113277	949
7	7	114319	11860

10,000 rows | Truncated data | 0.28 seconds runtime

2 minutes ago (<1s) 17

```
df_tags.count()
display(df_tags)
```

(3) Spark Jobs

Table + New result table: OFF

	userId	movieId	tag	timesta
1	2	60756	funny	1445714
2	2	60756	Highly quotable	1445714
3	2	60756	will ferrell	1445714
4	2	89774	Boxing story	1445715
5	2	89774	MMA	1445715
6	2	89774	Tom Hardy	1445715
7	2	106782	drugs	1445715

3,683 rows | 0.22 seconds runtime

2 minutes ago (1s) 18

```
#Movies with ratings: joining moveis DF with ratings DF

df_movies_with_ratings =  df_movies.join(df_ratings,"movieid","left")
display(df_movies_with_ratings)
```

(2) Spark Jobs

df_movies_with_ratings: pyspark.sql.dataframe.DataFrame = [movieId: integer, title: string ... 4 more fields]

Table + New result table: OFF

	movieId	title	genres
1	1	Toy Story (1995)	Adventure|Animation|Child
2	1	Toy Story (1995)	Adventure|Animation|Child
3	1	Toy Story (1995)	Adventure|Animation|Child
4	1	Toy Story (1995)	Adventure|Animation|Child
5	1	Toy Story (1995)	Adventure|Animation|Child
6	1	Toy Story (1995)	Adventure|Animation|Child
7			

10,000 rows | Truncated data | 0.79 seconds runtime

2 minutes ago (1s) 19

```
df_movies_with_ratings.count()
```

(3) Spark Jobs

```
100854
```

2 minutes ago (1s) 20

```
#Check if there are duplicates (Movies with more than 1 rating) ---> more the count more the rating

df_movies_no_dups = df_movies_with_ratings.groupby("movieid").count()
display(df_movies_no_dups)
```

(3) Spark Jobs

df_movies_no_dups: pyspark.sql.dataframe.DataFrame = [movieid: integer, count: long]

Table | + | New result table: OFF

	movieid	count
1	148	1
2	471	40
3	496	1
4	833	6
5	1088	42
6	1238	9
7	1342	11

9,742 rows | 0.89 seconds runtime

+ +

▶ ✓ 2 minutes ago (<1s) 21

```
#Join our ratings DF with tags DF (Join with users dataset)

df_ratings_tags = df_ratings.join(df_tags,"movieid","inner")
display(df_ratings_tags)
```

(2) Spark Jobs

df_ratings_tags: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 5 more fields]

Table | + | New result table: OFF

	movieId	userId	rating	timestamp	userId	tag
1	1	1	4	964982703	567	fun
2	1	1	4	964982703	474	pixar
3	1	1	4	964982703	336	pixar
4	3	1	4	964981247	289	old
5	3	1	4	964981247	289	moldy
6	47	1	5	964983815	474	serial killer
7	47	1	5	964983815	424	twist ending

10,000 rows | Truncated data | 0.48 seconds runtime

+ +

▶ ✓ 2 minutes ago (<1s) 22

```
df_ratings.describe()
```

```
DataFrame[summary: string, userId: string, movieId: string, rating: string, timestamp: string]
```

+ +

▶ ✓ 2 minutes ago (<1s) 23

```
#Adding new column from timestamp col using from_unixtime: Date conversion

df_ratings = df_ratings.withColumn("tsDate", f.from_unixtime("timestamp"))
```

```
df_ratings: pyspark.sql.dataframe.DataFrame
  userId: integer
  movieId: integer
  rating: double
  timestamp: integer
  tsDate: string
```

+ +

▶ ✓ 2 minutes ago (<1s) 24

```
df_ratings.describe()
```

```
DataFrame[summary: string, userId: string, movieId: string, rating: string, timestamp: string, tsDate: string]
```

+ +

▶ ✓ 2 minutes ago (<1s) 25

```
display(df_ratings)
```

(1) Spark Jobs

Table | + | New result table: OFF

	userId	movieId	rating	timestamp	tsDate
1	1	1	4	964982703	2000-07-30 18:45:03
2	1	3	4	964981247	2000-07-30 18:20:47
3	1	6	4	964982224	2000-07-30 18:37:04
4	1	47	5	964983815	2000-07-30 19:03:35
5	1	50	5	964982931	2000-07-30 18:48:51
6	1	70	3	964982400	2000-07-30 18:40:00
7	1	101	5	964980868	2000-07-30 18:14:28

10,000 rows | Truncated data | 0.30 seconds runtime

+ +

▶ ✓ 2 minutes ago (<1s) 26

```
#String to Date conversion in the required format
df_ratings = df_ratings.select('userId','movieId','rating', f.to_date(unix_timestamp('tsDate','yyyy-MM-dd HH:MM:SS')
alias("ratingDate"))
```

df_ratings: pyspark.sql.dataframe.DataFrame
- userId: integer
- movieId: integer
- rating: double
- ratingDate: date

+ +

▶ ✓ 2 minutes ago (<1s) 27

```
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
display(df_ratings)
```

(1) Spark Jobs

Table | + | New result table: OFF

	userId	movieId	rating	ratingDate
1	1	1	4	null
2	1	3	4	null
3	1	6	4	null
4	1	47	5	2000-03-30
5	1	50	5	null
6	1	70	3	null
7	1	101	5	null

10,000 rows | Truncated data | 0.40 seconds runtime

+ +

▶ ✓ 2 minutes ago (1s) 28

```
#year wise rating count
df_ratings_year = df_ratings.groupby("ratingDate").count()
display(df_ratings_year)
```

(2) Spark Jobs

df_ratings_year: pyspark.sql.dataframe.DataFrame = [ratingDate: date, count: long]

Table | + | New result table: OFF

	ratingDate	count
1	1996-12-22	7
2	2000-12-26	1
3	2000-03-17	16
4	2012-04-17	5
5	2000-07-03	10
6	2007-04-20	5

7	2012-10-06	2

4,207 rows | 1.39 seconds runtime

2 minutes ago (1s) 29

```
#finding Average ratings

df_avg_ratings = df_ratings.groupby("movieId").mean("rating")
display(df_avg_ratings)
```

(2) Spark Jobs

df_avg_ratings: pyspark.sql.dataframe.DataFrame = [movieId: integer, avg(rating): double]

Table + New result table: OFF

	movieId	avg(rating)
1	1580	3.487878787878788
2	2366	3.64
3	3175	3.58
4	1088	3.369047619047619
5	32460	4.25
6	44022	3.217391304347826
7	96488	4.25

9,724 rows | 0.69 seconds runtime

2 minutes ago (1s) 30

```
#Join avg ratings DF with movies Df on movie_ID

df = df_avg_ratings.join(df_movies,"movieId","inner")
df = df.withColumnRenamed("avg(rating)","average_rating")
display(df)
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [movieId: integer, average_rating: double ... 2 more fields]

Table + New result table: OFF

	movieId	average_rating	title
1	1580	3.487878787878788	Men in Black (a.k.a. MIB) (1997)
2	2366	3.64	King Kong (1933)
3	3175	3.58	Galaxy Quest (1999)
4	1088	3.369047619047619	Dirty Dancing (1987)
5	32460	4.25	Knockin' on Heaven's Door (1997)
6	44022	3.217391304347826	Ice Age 2: The Meltdown (2006)

9,724 rows | 0.79 seconds runtime

2 minutes ago (<1s) 31

```
#What do we have now in the DF?

df.printSchema()
```

```
root
 |-- movieId: integer (nullable = true)
 |-- average_rating: double (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)
```

2 minutes ago (1s) 32

```
#Grouping by rating count! -> Which movie have been rated how many times?
```

```
df_total_ratings = df_ratings.groupby("movieId").count()
display(df_total_ratings)
```

(2) Spark Jobs

df_total_ratings: pyspark.sql.dataframe.DataFrame = [movieId: integer, count: long]

Table | New result table: OFF

	movieId	count
1	1580	165
2	2366	25
3	3175	75
4	1088	42
5	32460	4
6	44022	23
7	96488	4

9,724 rows | 0.59 seconds runtime

2 minutes ago (<1s) 33

```
#Filtering movies which are having less than 5 ratings in total!

df_total_ratings = df_total_ratings.where("count > 10")
df_ratings_filtered = df_ratings.join(df_total_ratings,"movieId",'inner')
```

df_total_ratings: pyspark.sql.dataframe.DataFrame
  movieId: integer
  count: long

df_ratings_filtered: pyspark.sql.dataframe.DataFrame
  movieId: integer
  userId: integer
  rating: double
  ratingDate: date
  count: long

2 minutes ago (1s) 34

```
display(df_ratings_filtered)
```

(3) Spark Jobs

Table | New result table: OFF

	movieId	userId	rating	ratingDate	count
1	1	1	4	null	215
2	3	1	4	null	52
3	6	1	4	null	102
4	47	1	5	2000-03-30	203
5	50	1	5	null	204
6	70	1	3	null	55
7	101	1	5	null	23

10,000 rows | Truncated data | 1.00 second runtime

2 minutes ago (<1s) 35

```
#how many records do we have after filtering out the no. of ratings > 10

df_total_ratings.count()
```

(3) Spark Jobs

2121

2 minutes ago (2s) 36

```
#max rating per user

df_rating_per_user = df_ratings_filtered.select('userId','movieId','rating').groupby('userId','movieId').max('rating
display(df_rating_per_user)
```

(4) Spark Jobs

df_rating_per_user: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 1 more field]

Table + New result table: OFF

	userId	movieId	max(rating)
1	610	43928	2
2	610	76251	3.5
3	1	1208	4
4	1	1348	4
5	4	457	5
6	4	599	2
7	6	327	1

10,000 rows | Truncated data | 1.63 seconds runtime

2 minutes ago (<1s) 37

```
#joining df_rating_per user with out movies DF

df_rating_per_user_movie = df_rating_per_user.join(df_movies,'movieId','inner')
```

df_rating_per_user_movie: pyspark.sql.dataframe.DataFrame
- movieId: integer
- userId: integer
- max(rating): double
- title: string
- genres: string

2 minutes ago (1s) 38

```
#renaming max column into some good name

df_rating_per_user_movie = df_rating_per_user_movie.withColumnRenamed("max(rating)","Max_rating")
display(df_rating_per_user_movie)
```

(4) Spark Jobs

df_rating_per_user_movie: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 3 more fields]

Table + New result table: OFF

	movieId	userId	Max_rating	title
1	1580	608	3.5	Men in Black (a.k.a. MIB) (1997)
2	1580	607	3	Men in Black (a.k.a. MIB) (1997)
3	1580	603	4	Men in Black (a.k.a. MIB) (1997)
4	1580	599	3	Men in Black (a.k.a. MIB) (1997)
5	1580	597	3	Men in Black (a.k.a. MIB) (1997)
6	1580	590	3.5	Men in Black (a.k.a. MIB) (1997)
7				

10,000 rows | Truncated data | 1.20 seconds runtime

2 minutes ago (<1s) 39

```
temp_table_movie_with_ratings = "tb_rating_per_user_movie"
df_rating_per_user_movie.createOrReplaceTempView(temp_table_movie_with_ratings)
```

2 minutes ago (1s) 40

```
df_10 = spark.sql("""select * from tb_rating_per_user_movie where title Like 'Fugitive%'""")
display(df_10)
```

(4) Spark Jobs

df_10: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 3 more fields]

Table + New result table: OFF

	movieId	userId	Max_rating	title	genres
1	457	610	4.5	Fugitive, The (1993)	Thriller
2	457	609	4	Fugitive, The (1993)	Thriller
3	457	608	3	Fugitive, The (1993)	Thriller
4	457	607	5	Fugitive, The (1993)	Thriller
5	457	603	3	Fugitive, The (1993)	Thriller
6	457	602	5	Fugitive, The (1993)	Thriller
7	457	600	3	Fugitive, The (1993)	Thriller

190 rows | 0.90 seconds runtime

2 minutes ago (1s) 41

```
df_query = spark.sql("""select * from tb_rating_per_user_movie""")
display(df_query)
```

(4) Spark Jobs

df_query: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 3 more fields]

Table + New result table: OFF

	movieId	userId	Max_rating	title
1	1580	608	3.5	Men in Black (a.k.a. MIB) (1997)
2	1580	607	3	Men in Black (a.k.a. MIB) (1997)
3	1580	603	4	Men in Black (a.k.a. MIB) (1997)
4	1580	599	3	Men in Black (a.k.a. MIB) (1997)
5	1580	597	3	Men in Black (a.k.a. MIB) (1997)
6	1580	590	3.5	Men in Black (a.k.a. MIB) (1997)
7				

10,000 rows | Truncated data | 0.69 seconds runtime

2 minutes ago (1s) 42

```
df_rating = df_rating_per_user_movie.groupby('movieId','userId','title','genres').max('Max_rating')
display(df_rating)
```

(4) Spark Jobs

df_rating: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 3 more fields]

Table + New result table: OFF

	movieId	userId	title
1	1580	608	Men in Black (a.k.a. MIB) (1997)
2	1580	607	Men in Black (a.k.a. MIB) (1997)
3	1580	603	Men in Black (a.k.a. MIB) (1997)
4	1580	599	Men in Black (a.k.a. MIB) (1997)
5	1580	597	Men in Black (a.k.a. MIB) (1997)
6	1580	590	Men in Black (a.k.a. MIB) (1997)
7			

10,000 rows | Truncated data | 1.19 seconds runtime

2 minutes ago (1s) 43

```
#users with movies > 4 ratings

df_rating = df_rating.withColumnRenamed('max(Max_rating)', 'max_rating')
```

```
df_rating = df_rating.where('max_rating >= 4')
display(df_rating)
```

(4) Spark Jobs

df_rating: pyspark.sql.dataframe.DataFrame = [movieId: integer, userId: integer ... 3 more fields]

Table + New result table: OFF

	movieId	userId	title
1	1580	603	Men in Black (a.k.a. MIB) (1997)
2	1580	587	Men in Black (a.k.a. MIB) (1997)
3	1580	580	Men in Black (a.k.a. MIB) (1997)
4	1580	579	Men in Black (a.k.a. MIB) (1997)
5	1580	570	Men in Black (a.k.a. MIB) (1997)
6	1580	560	Men in Black (a.k.a. MIB) (1997)
7			

10,000 rows | Truncated data | 1.39 seconds runtime

+ +

▶ ✓ 2 minutes ago (1s) 44

```
#Best Movie per genre
df_movies_genre = df_rating.groupby('genres','title').count()
display(df_movies_genre)
```

(5) Spark Jobs

df_movies_genre: pyspark.sql.dataframe.DataFrame = [genres: string, title: string ... 1 more field]

Table + New result table: OFF

	genres	title
1	Comedy|Drama	American Splendor (2003)
2	Crime|Drama|Mystery|Thriller	General's Daughter, The (1999)
3	Comedy	In & Out (1997)
4	Comedy|Drama|Romance	Pretty in Pink (1986)
5	Drama|Romance	Circle of Friends (1995)
6	Children|Comedy	George of the Jungle (1997)
7		

2,080 rows | 1.29 seconds runtime

+ +

▶ ✓ 2 minutes ago (1s) 45

```
#identity genre of user

df_rating_genre = df_rating.select('userId','title','genres').groupby('userId','genres').count()
display(df_rating_genre)
```

(5) Spark Jobs

df_rating_genre: pyspark.sql.dataframe.DataFrame = [userId: integer, genres: string ... 1 more field]

Table + New result table: OFF

	userId	genres	count
1	292	Action|Comedy|Sci-Fi	2
2	177	Crime|Drama	7
3	12	Comedy|Romance	6
4	514	Western	1
5	489	Fantasy|Horror|Romance|Thriller	1
6	483	Action|Adventure|Animation|Children|Comedy|Fantasy	1
7	43	Adventure|Animation|Children|Comedy|Musical	1

10,000 rows | Truncated data | 1.43 seconds runtime

+ +

2 minutes ago (2s) 46

```
#DF recent movie

df_recent_movie = df_ratings.groupby('userId','movieId').agg(f.max(df_ratings['ratingDate']))
display(df_recent_movie)
```

(2) Spark Jobs

df_recent_movie: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 1 more field]

Table + New result table: OFF

	userId	movieId	max(ratingDate)
1	1	1208	null
2	1	1348	null
3	4	457	null
4	4	599	null
5	6	274	null
6	6	327	1996-07-17
7	6	520	1996-04-17

10,000 rows | Truncated data | 1.59 seconds runtime

2 minutes ago (2s) 47

```
#Visualization 1

display(df_movies_genre)
#movies_df['genres_arr'] = movies_df['genres'].str.split('|')
#movies_df.head()
#counter_lambda = lambda x: len(x)
#movies_df['genre_count'] = movies_df.genres_arr.apply(counter_lambda)
#movies_df.head()

#movies_df.set_index('movieId')
#movies_df.head()
```

(5) Spark Jobs

Table + New result table: OFF

	genres	title
1	Comedy|Drama	American Splendor (2003)
2	Crime|Drama|Mystery|Thriller	General's Daughter, The (1999)
3	Comedy	In & Out (1997)
4	Comedy|Drama|Romance	Pretty in Pink (1986)
5	Drama|Romance	Circle of Friends (1995)
6	Children|Comedy	George of the Jungle (1997)
7		

2,080 rows | 1.59 seconds runtime

2 minutes ago (1s) 48

```
df_movies_genre_viz = df_rating.toPandas()
#df_movies_genre['genre_arr'] = df_movies_genre['genres'].str.split('|')
#df_movies_genre.head()
```

(4) Spark Jobs

2 minutes ago (<1s) 49

```
df_movies_genre_viz['genre_arr'] = df_movies_genre_viz['genres'].str.split('|')
```

2 minutes ago (<1s) 50

```
df_movies_genre_viz.head()
```

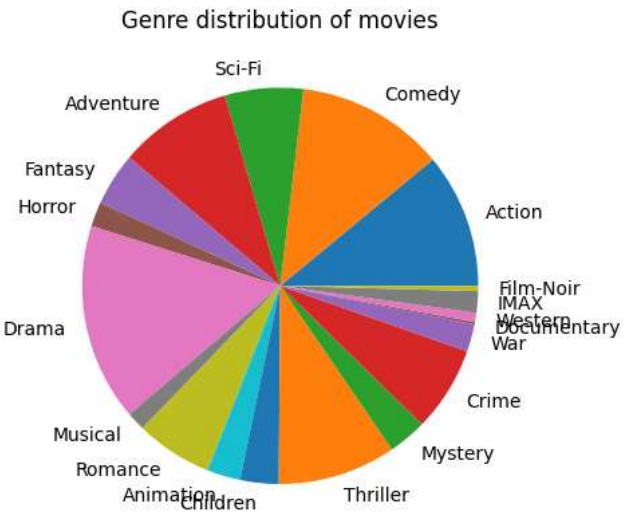
2 minutes ago (<1s) 51

```
counter_lambda = lambda x: len(x) #counter gets incremented each time
df_movies_genre_viz['genre_count'] = df_movies_genre_viz.genre_arr.apply(counter_lambda)
df_movies_genre_viz.head()

df_movies_genre_viz.set_index('movieId')
df_movies_genre_viz.head()
```

2 minutes ago (<1s) 52

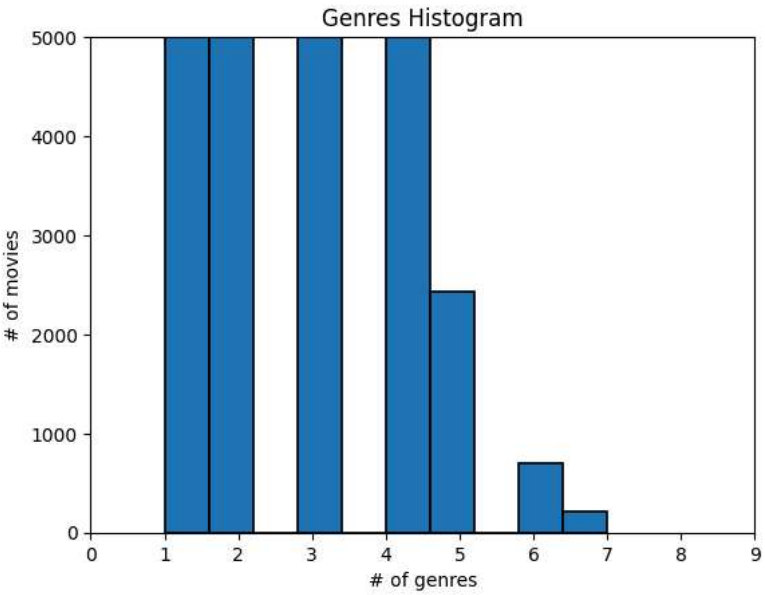
```
from collections import Counter

import pandas as pd
from pandas import DataFrame as df
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

flattened_genres = [item for sublist in df_movies_genre_viz.genre_arr for item in sublist]

genre_dict = dict(Counter(flattened_genres))

print(genre_dict)

# now lets plot this genre distribution as a pie chart
plt.pie(genre_dict.values(), labels=genre_dict.keys())
plt.title('Genre distribution of movies')
plt.show()

#plt.savefig('./movie-genres-pie.png')
```

```
{'Action': 12808, 'Comedy': 14137, 'Sci-Fi': 7449, 'Adventure': 10798, 'Fantasy': 5033, 'Horror': 2339, 'Drama': 187
68, 'Musical': 1777, 'Romance': 7314, 'Animation': 3156, 'Children': 3664, 'Thriller': 11245, 'Mystery': 3665, 'Crim
e': 8105, 'War': 2612, 'Documentary': 222, 'Western': 866, 'IMAX': 2050, 'Film-Noir': 491}
```



2 minutes ago (<1s) 53

```
import matplotlib.pyplot as plt
from collections import Counter

# Number of genres against number of movies
plt.hist(df_movies_genre_viz.genre_count, edgecolor='black', linewidth=1.2)
plt.title("Genres Histogram")
plt.xlabel("# of genres")
plt.ylabel("# of movies")
plt.axis([0, 9, 0, 5000])
plt.show()

# List of genres
flattened_genres = [item for sublist in df_movies_genre_viz.genres for item in sublist]
genre_dict = dict(Counter(flattened_genres))
print(genre_dict)
```



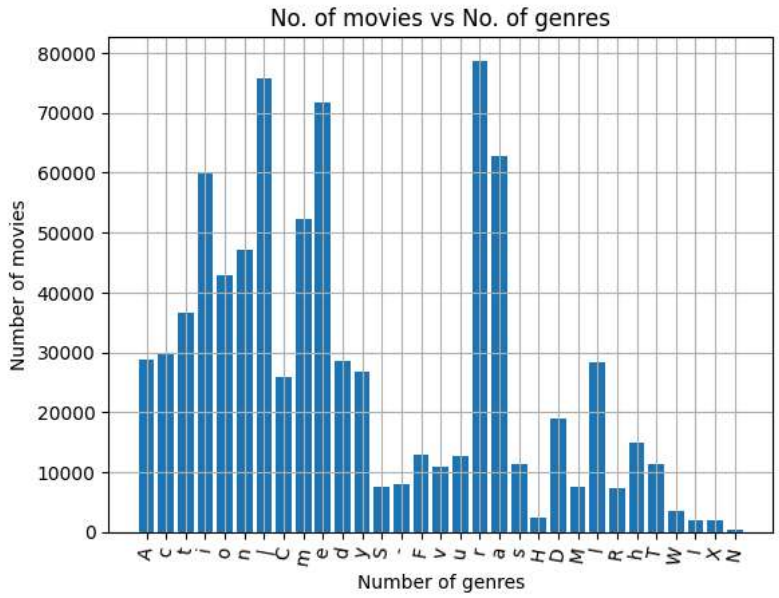

```
{'A': 28812, 'c': 29570, 't': 36548, 'i': 59791, 'o': 42806, 'n': 47017, ' ': 75634, 'C': 25906, 'm': 52193, 'e': 71680, 'd': 28599, 'y': 26722, 'S': 7449, '-': 7940, 'F': 12973, 'v': 10798, 'u': 12797, 'r': 78698, 'a': 62683, 's': 11341, 'H': 2339, 'D': 18990, 'M': 7492, 'l': 28422, 'R': 7314, 'h': 14909, 'T': 11245, 'W': 3478, 'I': 2050, 'X': 2050, 'N': 491}
```

2 minutes ago (<1s) 54

```
# For better readability
x = list(range(len(genre_dict)))
```

```
plt.title('No. of movies vs No. of genres')
plt.xticks(x, genre_dict.keys(), rotation=80)
plt.bar(x, genre_dict.values())
plt.xlabel("Number of genres")
plt.ylabel("Number of movies")
plt.grid()
plt.plot()
```

[]



+ +

▶ ✓ 2 minutes ago (1s) 55

```
df_Pandas_ratings = df_ratings.toPandas()
```

(1) Spark Jobs

+ +

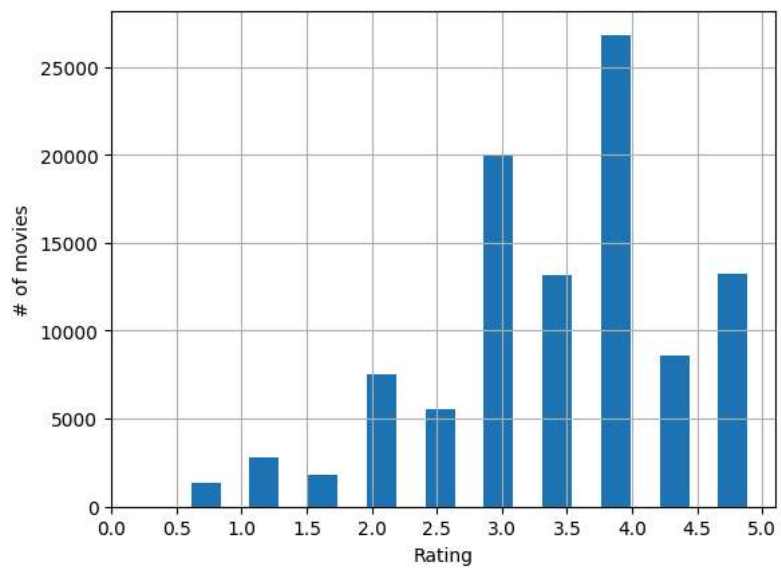
▶ ✓ 2 minutes ago (<1s) 56

```
df_Pandas_ratings.head()
```

+ +

▶ ✓ 2 minutes ago (<1s) 57

```
# Histogram of movie ratings
plt.hist(df_Pandas_ratings.rating,rwidth=0.5)
plt.xticks([0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0])
plt.xlabel('Rating')
plt.ylabel('# of movies')
plt.grid()
plt.show()
```



2 minutes ago (<1s) 58

```
# Top 25 movies

#most_rated=movielens.groupby('title').size().sort_values(ascending=False)[:25]

movie_lens = df_rating.toPandas
```

2 minutes ago (1s) 59

```
#most_rated = movie_lens.groupby('title').size().sort_values(ascending=False)[:25]
display(df_rating)
```

(4) Spark Jobs

Table + New result table: OFF

	movieId	userId	title
1	1580	603	Men in Black (a.k.a. MIB) (1997)
2	1580	587	Men in Black (a.k.a. MIB) (1997)
3	1580	580	Men in Black (a.k.a. MIB) (1997)
4	1580	579	Men in Black (a.k.a. MIB) (1997)
5	1580	570	Men in Black (a.k.a. MIB) (1997)
6	1580	560	Men in Black (a.k.a. MIB) (1997)
7			

10,000 rows | Truncated data | 1.10 seconds runtime

2 minutes ago (<1s) 60

```
df_genres_count = df_rating.groupby('genres').count()
```

df_genres_count: pyspark.sql.dataframe.DataFrame
genres: string
count: long

2 minutes ago (1s) 61

```
display(df_genres_count)
```

(5) Spark Jobs

Table + New result table: OFF

	genres	count

1	Action|Drama|Horror	14
2	Comedy|Horror|Thriller	47
3	Action|Adventure|Drama|Fantasy	141
4	Adventure|Sci-Fi|Thriller	15
5	Action|Adventure|Drama	259
6	Animation|Children|Drama|Musical|Romance	43
7	Adventure|Sci-Fi	31

493 rows | 1.30 seconds runtime

2 minutes ago (1s) 62

```
# Load processed movie and ratings data into Spark DataFrames
movies_df = spark.read.csv("dbfs:/FileStore/tables/movies.csv", header=True, inferSchema=True)
ratings_df = spark.read.csv("dbfs:/FileStore/tables/ratings.csv", header=True, inferSchema=True)
```

(4) Spark Jobs

movies_df: pyspark.sql.dataframe.DataFrame
  movieId: integer
  title: string
  genres: string

ratings_df: pyspark.sql.dataframe.DataFrame
  userId: integer
  movieId: integer
  rating: double
  timestamp: integer

2 minutes ago (14s) 63

```
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Split data into training and testing sets
(training, test) = ratings_df.randomSplit([0.8, 0.2])

# Train ALS model
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
          coldStartStrategy="drop")
model = als.fit(training)

# Evaluate the model
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

(9) Spark Jobs

(1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

training: pyspark.sql.dataframe.DataFrame
  userId: integer
  movieId: integer
  rating: double
  timestamp: integer

test: pyspark.sql.dataframe.DataFrame
  userId: integer
  movieId: integer
  rating: double
  timestamp: integer

predictions: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 3 more fields]

```
Root-mean-square error = 1.078625079803259
```

2 minutes ago (5s) 64 Python

```
# Generate top N recommendations for all users
userRecs = model.recommendForAllUsers(10)
```

```
# Show recommendations for a specific user
userId = 1
userRecs.filter(userRecs.userId == userId).show()
```

(2) Spark Jobs

userRecs: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|     1|[{7748, 6.6349745...|
+------+--------------------+
```