



# Interactive Data Exploration With PyFlink and Zeppelin Notebooks

---

Marta Paes (@morsapaes)  
Developer Advocate

# About Ververica



Original Creators of  
Apache Flink®



Enterprise Stream Processing  
With Ververica Platform



Part of  
Alibaba Group



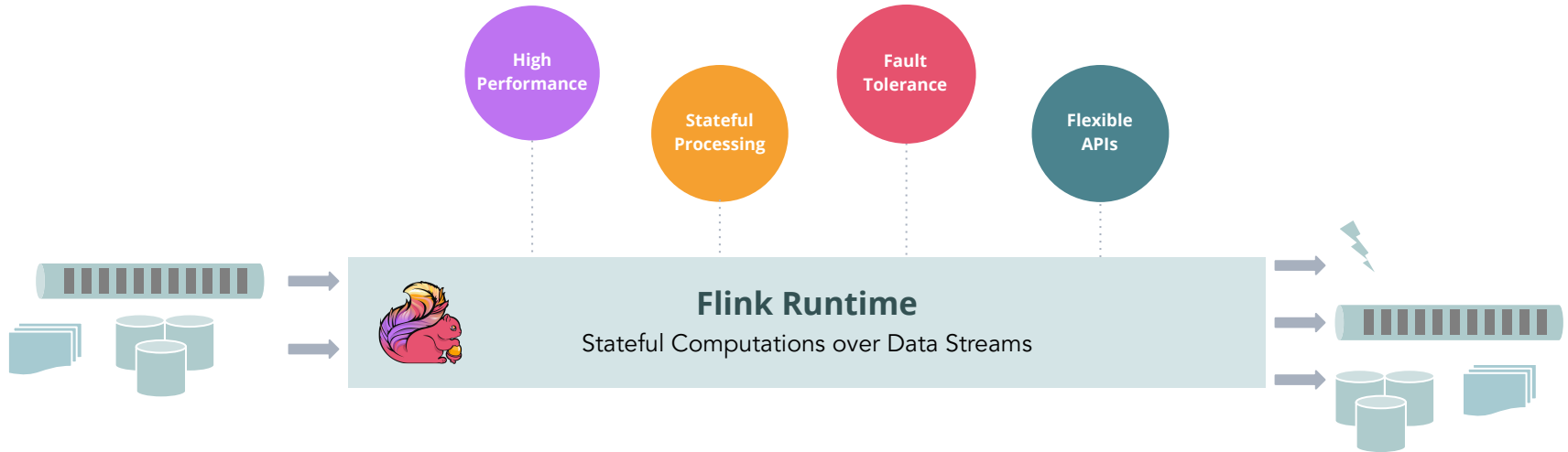
# Apache Flink

Flink is an **open source** framework and **distributed** engine for **stateful stream processing**.



# Apache Flink

Flink is an **open source** framework and **distributed** engine for **stateful stream processing**.



# Use Cases

This gives you a robust foundation for a wide range of use cases:



# Use Cases

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.



# Stateful Stream Processing

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.

- Explicit control over these primitives
- Complex computations and customization
- Maximize **performance** and **reliability**

## Example Use Cases

**NETFLIX**

[Large-scale Data Pipelines](#)

**ING** 

[ML-Based Fraud Detection](#)

**aws** 

[Service Monitoring & Anomaly Detection](#)



# Use Cases

More high-level or domain-specific use cases that can be modeled with SQL or Python and dynamic tables.





# Streaming Analytics & ML

More high-level or domain-specific use cases that can be modeled with SQL or Python and dynamic tables.

- Focus on logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**

## Example Use Cases



Uber

criteo.

[Unified Online/Offline Model Training](#)

[E2E Streaming Analytics Pipelines](#)

[ML Feature Generation](#)



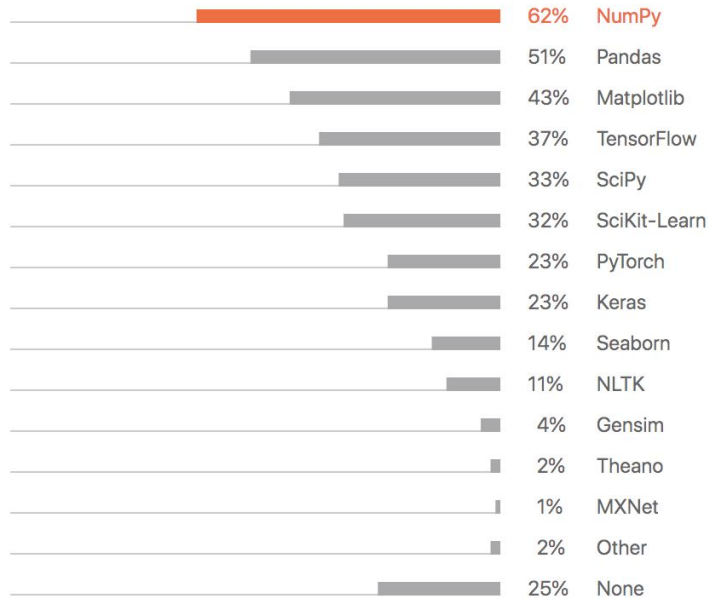
# More Flink Users





# Python is...pretty stacked?

## What data science frameworks do you use in addition to Python?

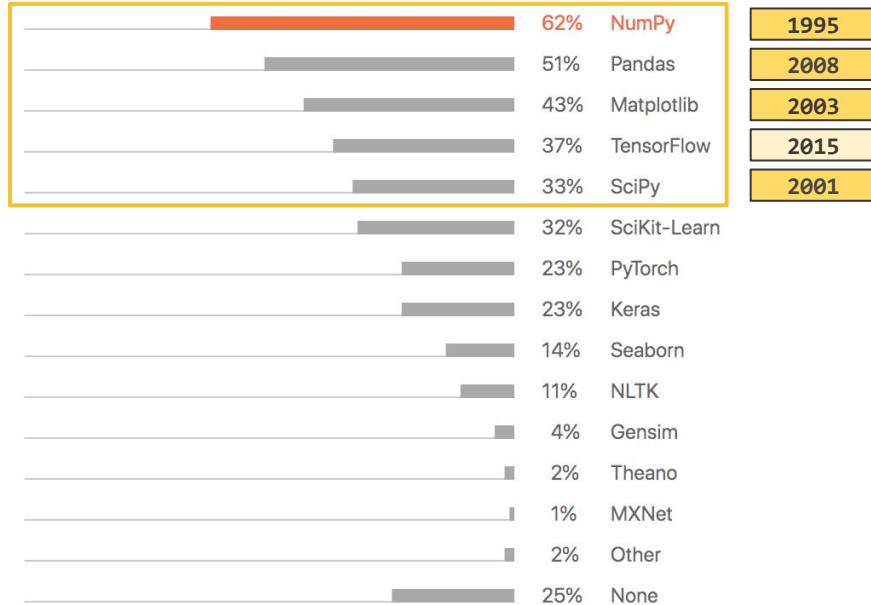


**Mature** analytics stack, with libraries that are **fast** and **intuitive**.



# ...and also timeless!

## What data science frameworks do you use in addition to Python?

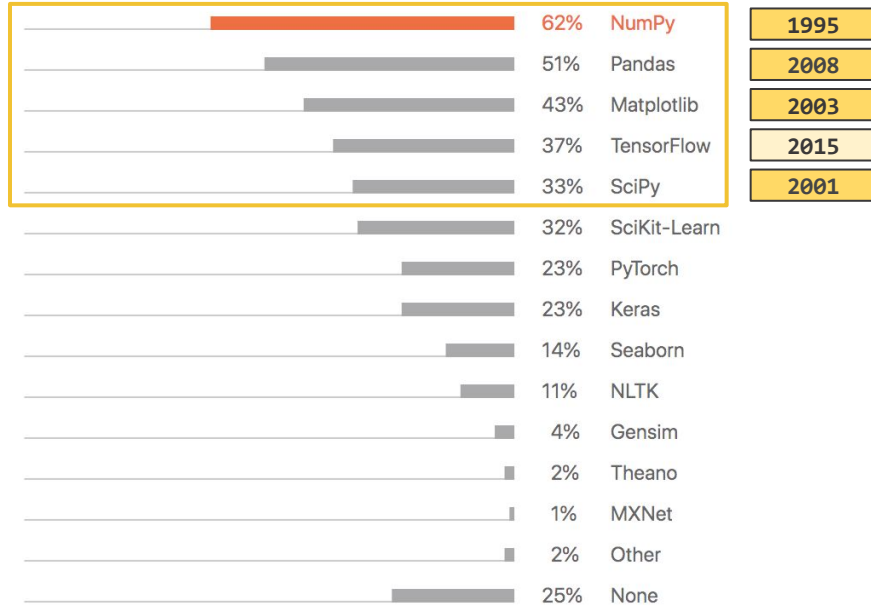


**Mature** analytics stack, with libraries that are **fast** and **intuitive**.



# ...and also timeless!

## What data science frameworks do you use in addition to Python?



**Mature** analytics stack, with libraries that are **fast** and **intuitive**.



Older libraries are mostly **restricted** to a data size that **fits in memory** (RAM), and designed to run on a **single core** (CPU).



This is a problem.



MORE DATA

FROM MORE  
PLACES



MOVING  
FASTER

IN MORE  
FORMATS

MORE  
REQUIREMENTS





But you still want to use these powerful libraries, right?



# Why PyFlink?

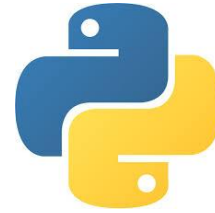


# Why PyFlink?

Expose the functionality of Flink to Python users



# Why PyFlink?



**Distribute** and **scale** the functionality of Python through Flink




# Flink at Alibaba scale

Real-time Data Applications incl. sub-second updates to the GMV dashboard




## Infrastructure



>5K nodes

>500K CPU cores

## Data Size




985PB

## State Size (Biggest)



100TB

## Throughput (Peak)



2.5B events/sec

## Latency



Sub-sec



# PyFlink in a Nutshell\*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)



# PyFlink in a Nutshell\*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)

## Execution



## UDF Support



# PyFlink in a Nutshell\*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)

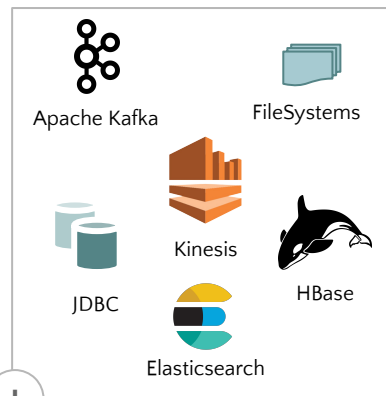
## Execution



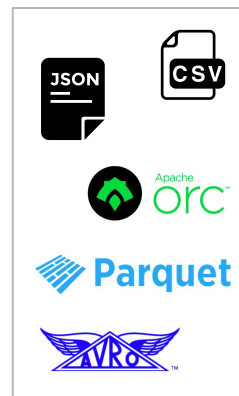
## UDF Support



## Native Connectors



## Formats



## ML Library (WIP)



## Notebooks





# PyFlink in a Nutshell\*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)

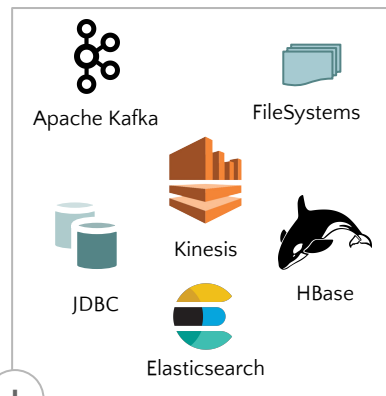
## Execution



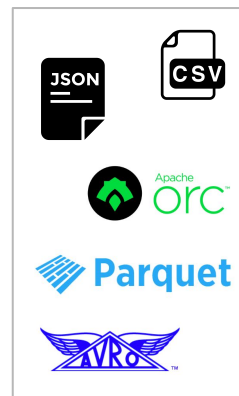
## UDF Support



## Native Connectors



## Formats

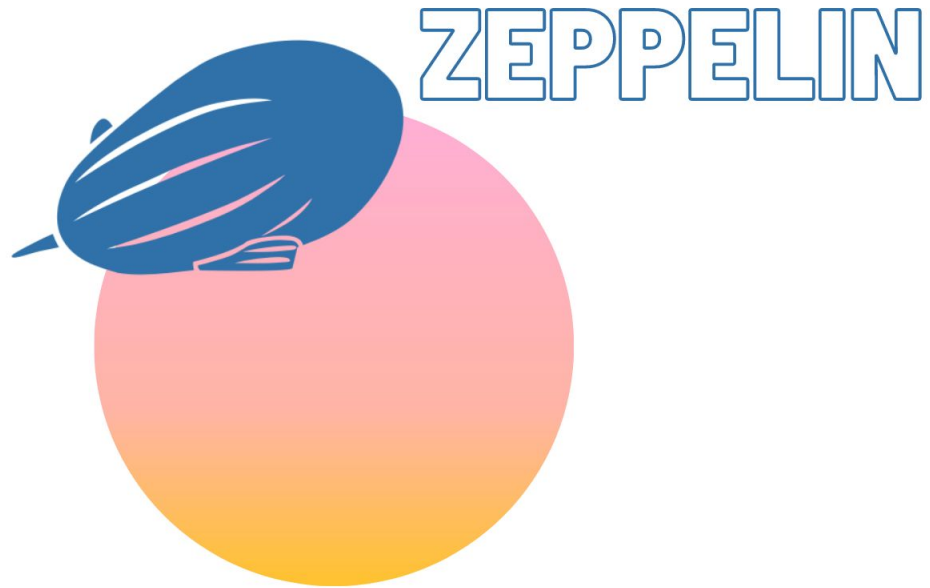


## ML Library (WIP)



## Notebooks





# Apache Zeppelin

Web-based notebook that provides an interactive and collaborative computing environment.



...



## Advantages

- Support for *a lot* of interpreters
- Polyglot notes
- Built-in interactive visualizations
- Multi-tenancy
- Pluggable notebook storage (e.g. git)



# ApacheCon@Home 2020

## Exploring the Movie Lovers on MUBI dataset (Kaggle)

FINISHED

We will be looking at three different files:

1. `mubi_movie_data.csv` - Data from all movies registered on Mubi.
2. `mubi_ratings_data.csv` - Data from ratings on Mubi for users who did not set their profile in private mode (~15 million rows).
3. `mubi_ratings_user_data.csv` - Aggregated data from users for a specific day.

### Flink

We will use a mix of the `%flink.pyflink` and `%flink.bsql` interpreters. For `%flink.pyflink`, Zeppelin creates the following environment variables for us:

- `s_env`: StreamExecutionEnvironment
- `b_env`: ExecutionEnvironment
- `st_env`: StreamTableEnvironment
- `bt_env`: BatchTableEnvironment

For simplicity, we're batch-reading files stored locally (also to keep my machine from exploding during the presentation!). The cool thing about PyFlink is that you can e.g. replace the existing source tables with Kafka-backed ones that continuously consume data and not have to change any of the code (other than `bt_env` to `st_env`).

Took 0 sec. Last updated by anonymous at September 30 2020, 6:05:28 PM.



## Option 1. Create table using %flink.pyflink

FLINK JOB FINISHED

```
%flink.pyflink

c_exp_drop_movies = """
DROP TABLE IF EXISTS mubi_movies
"""

c_exp_movies = """
CREATE TABLE mubi_movies (
  movie_id INT,
  movie_title STRING,
  movie_release_year STRING, --Needs cleanup, so bringing in as STRING
  movie_url STRING,
  movie_title_language STRING,
  movie_popularity INT,
  movie_image_url STRING,
  director_id STRING,      --Needs cleanup, so bringing in as STRING
  director_name STRING,
  director_url STRING
)
WITH (
  'connector' = 'filesystem',
  'path' = '/Users/martapaesmoreira/Desktop/ACAH2020_data/mubi_movie_data.csv',
  'format' = 'csv'
)
"""

bt_env.execute_sql(c_exp_drop_movies)
bt_env.execute_sql(c_exp_movies)

mubi_movies = bt_env \
    .from_path("mubi_movies") \
    .select("movie_id.count as movie_cnt")

z.show(mubi_movies)
```

settings

movie_cnt
226575

## Option 2. Create table using %flink.bsql

FLINK JOB FINISHED

```
%flink.bsql

DROP TABLE IF EXISTS mubi_movies;

CREATE TABLE mubi_movies (
  movie_id INT,
  movie_title STRING,
  movie_release_year STRING, --Needs cleanup, so bringing in as STRING
  movie_url STRING,
  movie_title_language STRING,
  movie_popularity INT,
  movie_image_url STRING,
  director_id STRING,      --Needs cleanup, so bringing in as STRING
  director_name STRING,
  director_url STRING
)
WITH (
  'connector' = 'filesystem',
  'path' = '/Users/martapaesmoreira/Desktop/ACAH2020_data/mubi_movie_data.csv',
  'format' = 'csv'
);

--Get a record count.
SELECT COUNT(*) movie_cnt FROM mubi_movies;
```

Table has been dropped.  
Table has been created.

settings

movie_cnt
226575



## Query the Movie Table

FINISHED

We will use PyFlink to query the `mubi_movies` table and get the average movie popularity per movie release year. What do we see?

1. There are two clear outlier years when it comes to popularity (1878 and 1902). If you check further, there is only one movie release in each of these years on Mubi ("Sallie Gardner at a Gallop" and "A Trip to the Moon") — they're just really popular!
2. The 1920s were a busy period for silent movie releases, which seem pretty popular with Mubi users.
3. The 1920s-1960s are also considered the golden era of Hollywood, so that can also explain the increased popularity of movies released in this period.

If you visit the [Flink Web UI](#) once you click "Run", you will see the job that generated from this code!

Took 0 sec. Last updated by anonymous at September 18 2020, 8:44:26 AM.

## AVG Movie Popularity / Release Year

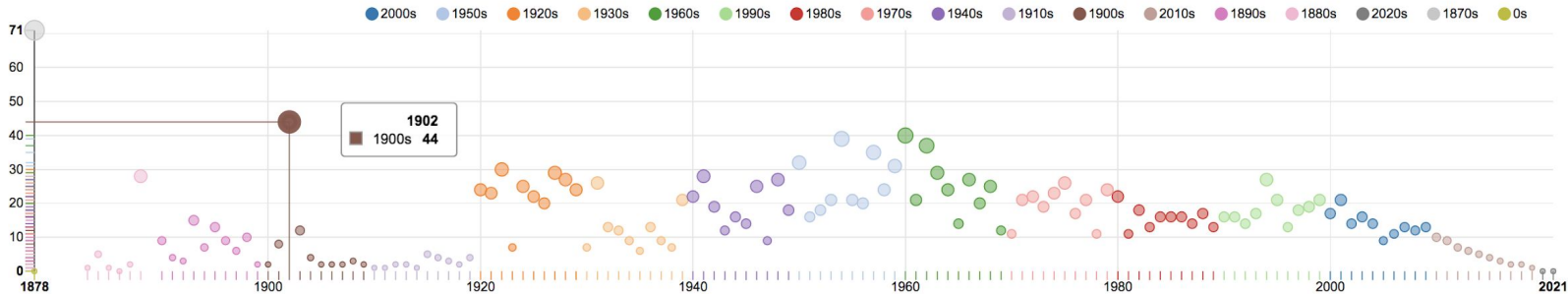
FLINK JOB FINISHED

```
%flink.pyflink
```

```
movies_avg_popularity = bt_env \  
    .from_path("mubi_movies") \  
    .group_by("movie_release_year") \  
    .select("movie_release_year.substring(1,4) AS movie_year, movie_release_year.substring(1,3)+'\0s\' AS movie_decade, movie_popularity.avg AS popularity_avg")
```

```
z.show(movies_avg_popularity)
```

settings



## Active Users over time

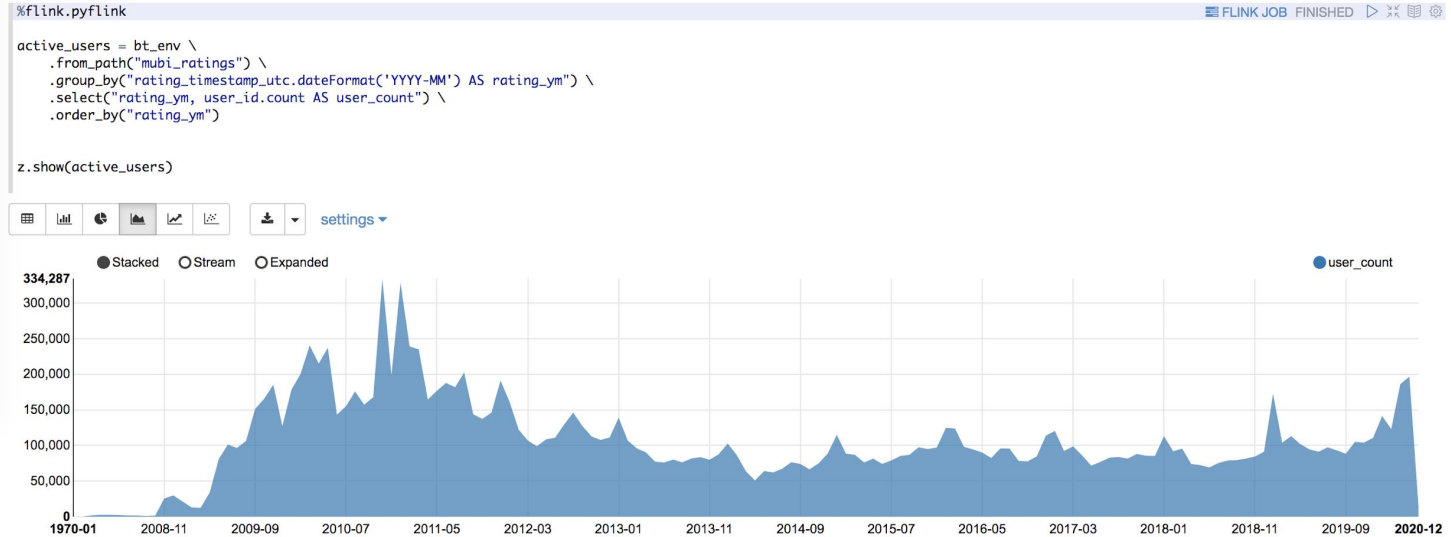
FINISHED   

Measure: rating activity.

The number of active users in the Mubi platform has seen a steady increase over the years, with a decline followed by steadybut lower activity in comparison, from 2012. It's funny to see that:

1. There is a peak active users on **January 1st** every year, which is likely when people are making their "BEST OF YEAR X" lists.
2. There was a huge boost in active users when the **Corona pandemic** hit (2020-03).

Took 0 sec. Last updated by anonymous at September 18 2020, 8:45:36 AM.



## Top 10 Rated Movies in Year X

FLINK JOB FINISHED

```
%flink.pyflink

l_top_10_year = bt_env \
    .from_path("mubi_movies") \
    .select("movie_id, movie_title, movie_release_year")

r_top_10_year = bt_env \
    .from_path("mubi_ratings") \
    .filter("rating_timestamp_utc.dateFormat('YYYY')='"+z.textBox("year", "2020")+"'") \
    .group_by("movie_id") \
    .select("movie_id AS m_id, rating_score.avg AS r_avg, critic.count AS c_critic") \
    .where("r_avg.isNotNull")

top_10_year = l_top_10_year \
    .join(r_top_10_year) \
    .where("movie_id=m_id") \
    .select("movie_title, movie_release_year.substring(1,4) AS movie_year, r_avg AS avg_rating_score, c_critic AS n_user_critics") \
    .order_by("avg_rating_score.desc, n_user_critics.desc") \
    .fetch(10)

z.show(top_10_year)
```

year

Grid, Bar, Pie, Line, Area, Scatter, Download, settings

movie_title	movie_year	avg_rating_score	n_user_critics
On Cinema at the Cinema	2012	5.0	6
The Day Today	1994	5.0	6
The Peter Serafinowicz Show	2007	5.0	4
Dans l'obscurite	2007	5.0	4
Joe Hisaishi Budokan Studio Ghibli 25 Years	2008	5.0	4





## Using Pandas (and other Python libraries)

FINISHED ▶ 🔍 📄 ⚙️

### Conversion .toPandas()

One way to use PyFlink with Pandas is to first use it to reduce the amount of data we want to act upon (which might be a considerably small subset of the original dataset), taking advantage of how performant PyFlink is even on the largest of largest datasets; and then convert the resulting table into a Pandas [DataFrame](#).

Took 0 sec. Last updated by anonymous at September 18 2020, 9:29:13 AM.

### Plotting a Histogram

FLINK JOB FINISHED ▶ 🔍 📄 ⚙️

```
%Flink.pyFlink

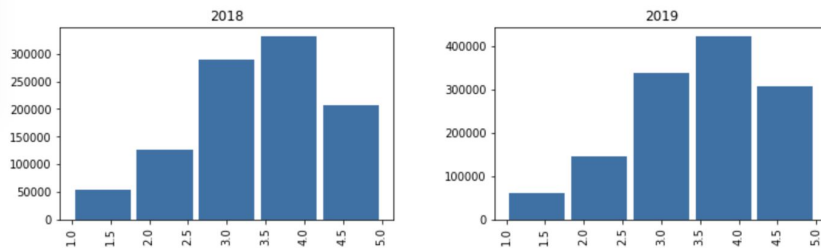
ratings_1820 = bt_env \
    .from_path("mubi_ratings") \
    .filter("rating_timestamp_utc.dateFormat('YYYY').in('2018','2019','2020')") \
    .select("rating_timestamp_utc.dateFormat('YYYY') AS rating_year, rating_score")

pdf = ratings_1820.to_pandas()

ax = pdf.hist(column='rating_score', by='rating_year', bins=5, grid=False, figsize=(12,8), color='#3171A9', zorder=2, rwidth=0.9)

z.show(ax)
```

```
[<AxesSubplot:title={'center':'2018'}>
<AxesSubplot:title={'center':'2019'}>]
[<AxesSubplot:title={'center':'2020'}> <AxesSubplot:>]
```



# Want to learn more about Flink?

#flinkforward

Join the Apache Flink community virtually at  
Flink Forward Global 2020!

Register for free:

[flink-forward.org/global-2020](https://flink-forward.org/global-2020)

**FLINK  
FORWARD**   
October 19-22, 2020





# Thank you, ApacheCon!

---

Follow me on Twitter: @morsapaes

Learn more about Flink: <https://flink.apache.org/>