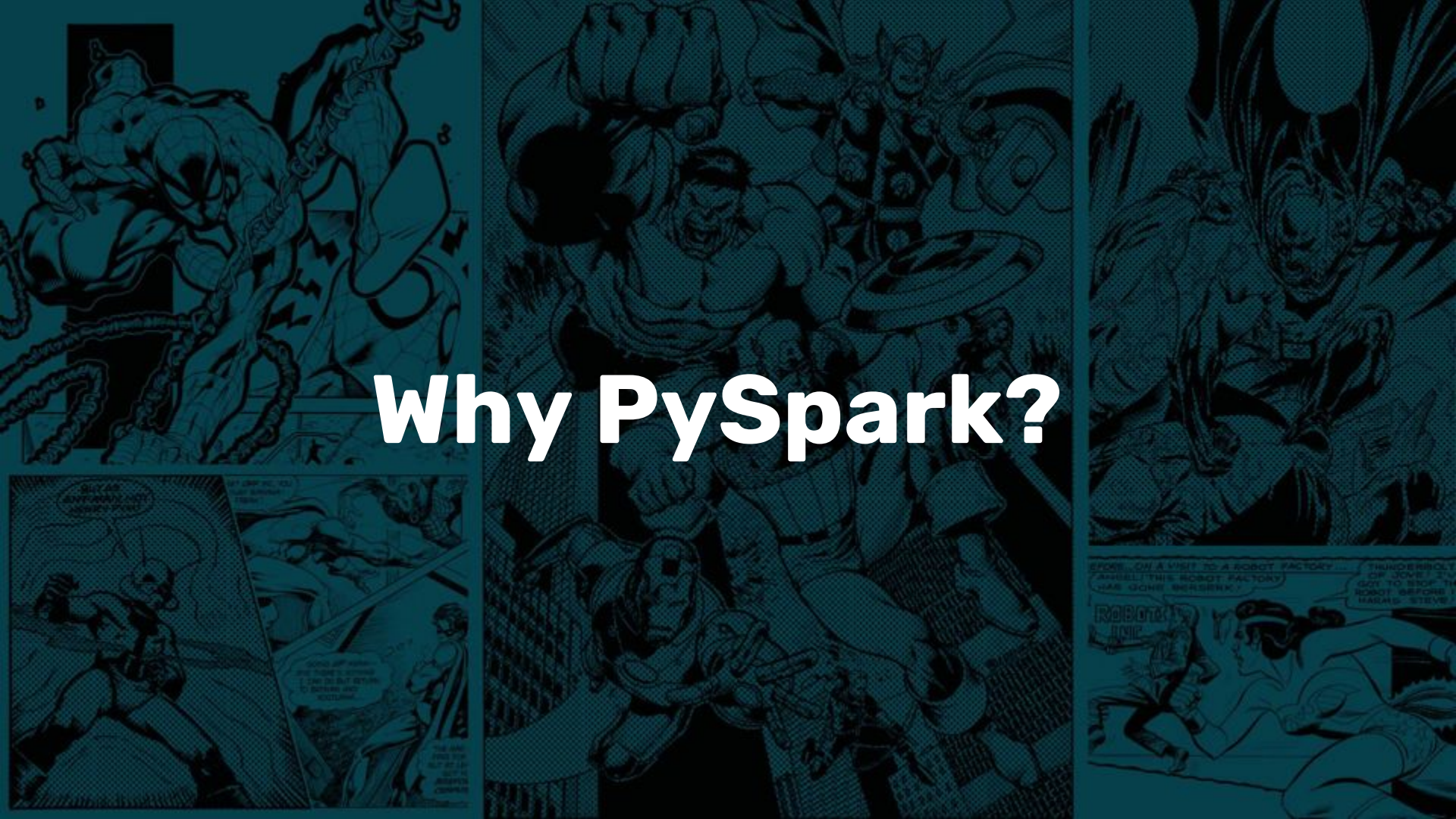
How PySpark can make you a better analyst?

Dorota Mierzwa
Fandom

WiMLDS meetup, 28.02.2019

Agenda

1. Why PySpark?
2. Data analysis workflow
3. Lessons and examples


Why PySpark?

Rick and Morty Wiki

RICKIPEDIA

RICK AND MORTY

1,476 PAGES

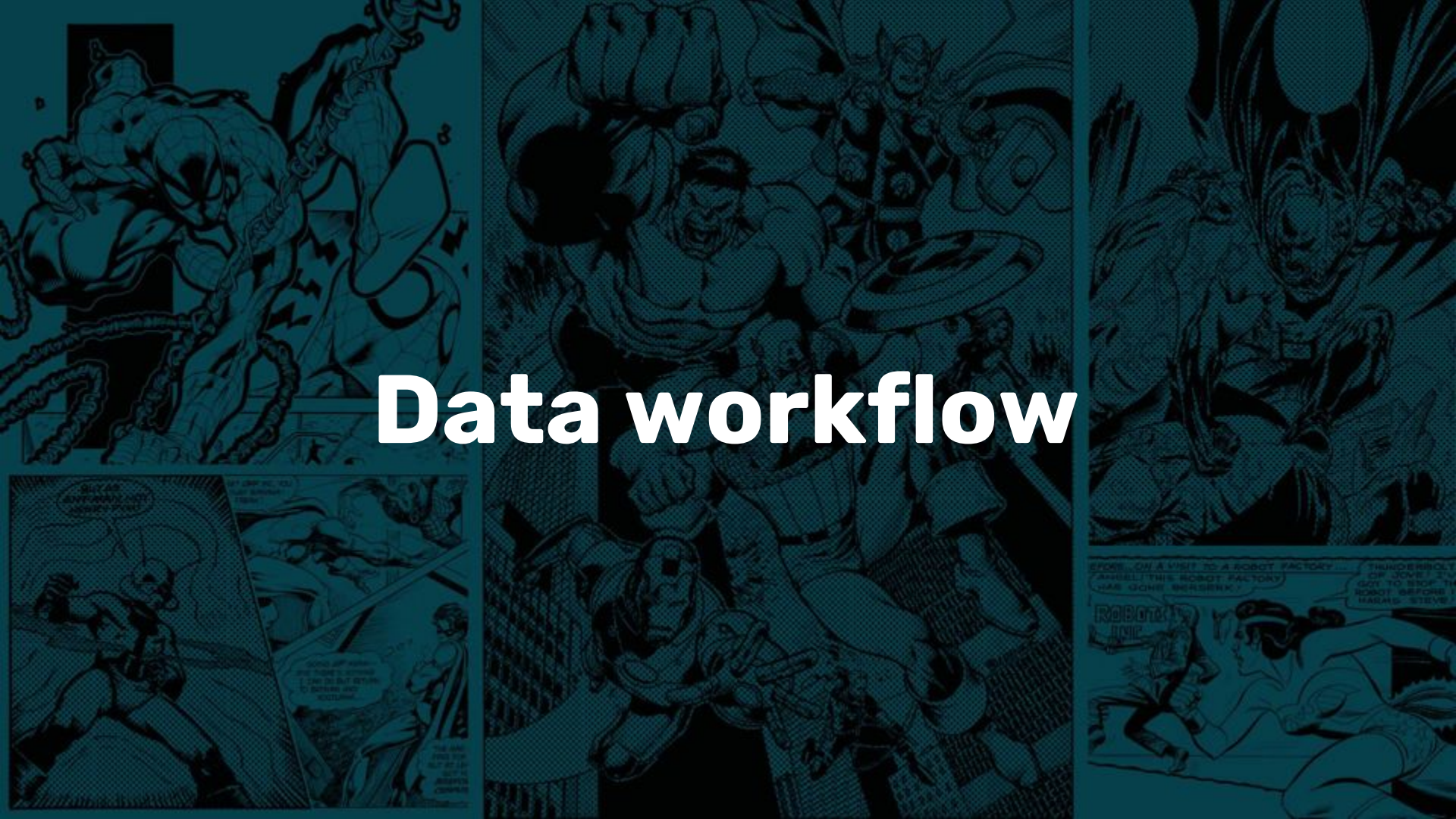
ADD NEW PAGE

other dimensions, building various robots and devices, and causing general mayhem in different parts of the universe

In the episode "The Wedding Squanchers", Rick willingly turned himself over to the Galactic Federation and was sent to one of their maximum security prisons. He currently resides there, with little chance of bail, parole or escape.

Appearance

Rick is a tall, lanky old man. He has long legs and arms and is very skinny. He has a dimly tanned ashy complexion and grey-blue hair with a bald spot on the back of his head. He has spiky hair on his head and a unibrow. His face is wrinkly as he has bags under his eyelids and a pressure fold above his unibrow that follows its position and laugh lines on both sides of his mouth. He wears a white lab coat with a light blue green shirt underneath. He also wears brown pants, a dark brown belt with a yellow buckle, and black shoes. He is occasionally seen with some green spill on his mouth, Showing up mainly when he is drunk or after he throws up. Rick speaks in a rambling, stammering manner that is often interrupted by belching and gagging, usually from being drunk.



Data workflow

JupyterHub

PySpark notebooks

- Operations on multiple nodes
- Querying heavy data
- Exploration
- Wrangle / join / aggregate
- Save result to few kB csv

Python notebooks

- Operations in local memory
- Use Pandas
- Query smaller amounts of data
- Visualize / summarize results

PySpark lesson 1

Answering the right questions

What your data can tell you?

It tells you a lot, but do you need it all?

In Pandas...

```
In [6]: df.info()
```

```
In [9]: df.head()
```

Out[9]:

	Unnamed: 0	day	ad_id	order	code_served	measurable_imps	viewable_imps	video_start	video_complete	video_error
0	0	2019-02-01	706185648	-3	2	0	0	0.0	0.0	0.0
1	1	2019-02-01	1235827404	-3	5	0	0	0.0	0.0	0.0
2	2	2019-02-01	1059282792	-7	13	0	0	0.0	0.0	0.0
3	3	2019-02-01	174660132	-5	15	0	0	0.0	0.0	0.0
4	4	2019-02-01	175067412	-6	27	0	0	0.0	0.0	0.0

```
video_error          2348 non-null float64
dtypes: float64(3), int64(6), object(1)
memory usage: 3.4+ MB
```

12	12	2019-02-01	878139660	-9	2	0	0	NaN	NaN	NaN

..and in PySpark

```
In [29]: df.count()
```

```
1214445
```

```
In [30]: df.printSchema()
```

```
root
 |-- day: string (nullable = true)
 |-- pv_unique_id: string (nullable = true)
 |-- line_item_id: long (nullable = true)
 |-- event_name: string (nullable = true)
 |-- video_id: string (nullable = true)
```

```
In [52]: df.take(1)
```

```
[Row(day='2019-02-01', pv_unique_id='eb73e51f-c06a-4', line_item_id=0, event_name='error', video_id='NaN')]
```

Nulls

In [55]:
```
# number of null values of a column
df.where(col('video_id').isNull()).count()
```

```
190800
```

In [56]:
```
# calculate % of not nulls
df.select([(count(when(col(c).isNull(), c)) / df.count() * 100).alias(c + ' null %')
                        for c in df.columns]).show()
```

```
+---------+------------------+------------------+----------------+----------------+
|day null %|pv_unique_id null %|line_item_id null %|event_name null %| video_id null %|
+---------+------------------+------------------+----------------+----------------+
|       0.0|               0.0|               0.0|             0.0|15.7108802786458|
+---------+------------------+------------------+----------------+----------------+
```

Percentile

In [69]:
```
# distinct video by pageview
video_by_pageview = df.select('pv_unique_id', 'video_id')\
    .groupBy('pv_unique_id')\
    .agg(countDistinct('video_id'))\
    .cache()
```

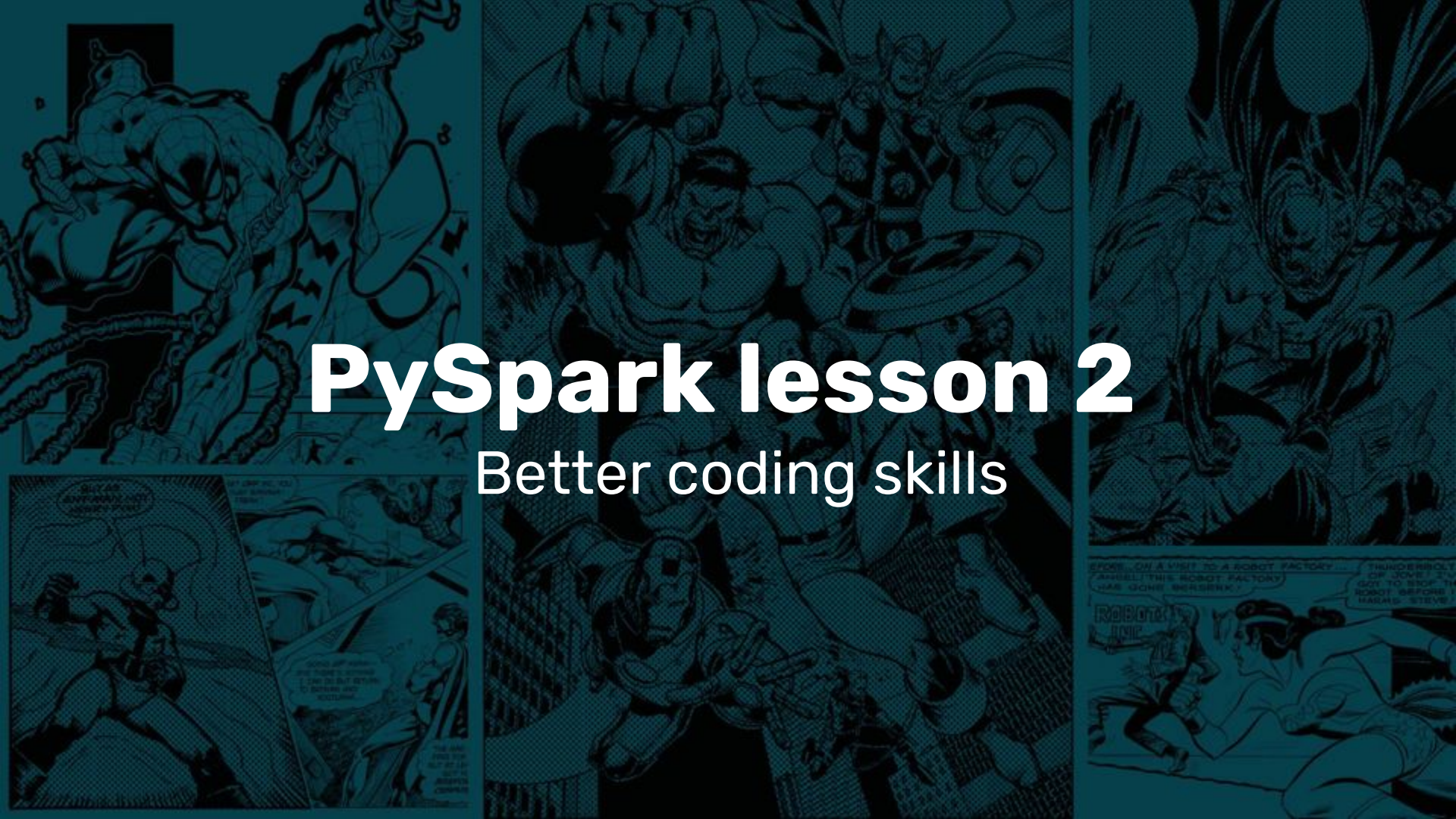
In [70]:
```
video_by_pageview.describe().show()
```

```
+-------+--------------+------------------------+
|summary|   pv_unique_id|count(DISTINCT video_id)|
+-------+--------------+------------------------+
|  count|        122685|                  122685|
|   mean|          null|      1.2723234299221584|
| stddev|          null|       1.951014985795724|
|    min|00006ee1-08b9-4|                       0|
|    max|fffff0d4-c17d-4|                      34|
+-------+--------------+------------------------+
```

In [72]:
```
video_by_pageview.approxQuantile("count(DISTINCT video_id)", [0.5], 0.25)
```

```
[24.0]
```

Percentile **Relative error**



PySpark lesson 2

Better coding skills

	day	pv_video	skin	event_name	closed
0	2019-01-10	86656a5b-dbd7-4	A	impression	1.0
1	2019-01-10	9ca57b3d-e6b4-4	B	impression	1.0
2	2019-01-10	07569b59-33ed-4	A	viewable_impression	0.0
3	2019-01-10	118f54c5-5e6e-4	A	impression	0.0
4	2019-01-10	aac0cd47-ea0f-4	A	impression	0.0

Creating dummy columns from categorical variables

	day	pv_video	skin	closed	impression	viewable_impression
0	2019-01-10	86656a5b-dbd7-4	A	1.0	1	0
1	2019-01-10	9ca57b3d-e6b4-4	B	1.0	1	0
2	2019-01-10	07569b59-33ed-4	A	0.0	0	1
3	2019-01-10	118f54c5-5e6e-4	A	0.0	1	0
4	2019-01-10	aac0cd47-ea0f-4	A	0.0	1	0

Pandas:

```
In [42]: df2 = pd.get_dummies(df2, columns=['event_name'])
```

PySpark:

```
# create a list of distinct values
events = df.select('event_name').distinct().rdd.flatMap(lambda x:x).collect()
```

```
events
```

```
['first_quartile', 'midpoint', 'completed', 'viewable_impression', 'init', 'third_quartile', 'started']
```

```
dummies = [when(col('event_name') == e, 1).otherwise(0).alias(str(e)) for e in events]
```

PySpark:

```
df.select(df.columns[::3] + dummies).show(5)
```

```
+----------+----------+--------------+--------+---------+--------------------+----+--------------+-------+
|       day|event_name|first_quartile|midpoint|completed|viewable_impression|init|third_quartile|started|
+----------+----------+--------------+--------+---------+--------------------+----+--------------+-------+
|2019-02-01|      init|             0|       0|        0|                   0|   1|             0|      0|
|2019-02-01|      init|             0|       0|        0|                   0|   1|             0|      0|
|2019-02-01|      init|             0|       0|        0|                   0|   1|             0|      0|
|2019-02-01|      init|             0|       0|        0|                   0|   1|             0|      0|
|2019-02-01|   started|             0|       0|        0|                   0|   0|             0|      1|
+----------+----------+--------------+--------+---------+--------------------+----+--------------+-------+
only showing top 5 rows
```

PySpark:

```
In [38]: dummies[0]

         Column<b'CASE WHEN (event_name = midpoint) THEN 1 ELSE 0 END AS `midpoint`'>
```

PySpark, attempt no. 1:

```
# dummy columns
outstream = outstream.withColumn('ready', when(col('event_name')== 'ready', lit(1)).otherwise(lit(0)))\
    .withColumn('first_quartile', when(col('event_name')== 'first_quartile', lit(1)).otherwise(lit(0)))\
    .withColumn('midpoint', when(col('event_name')== 'midpoint', lit(1)).otherwise(lit(0)))\
    .withColumn('completed', when(col('event_name')== 'completed', lit(1)).otherwise(lit(0)))\
    .withColumn('play_triggered', when(col('event_name')== 'play_triggered', lit(1)).otherwise(lit(0)))\
    .withColumn('in_viewport_without_offer', when(col('event_name')== 'in_viewport_without_offer', lit(1)).otherwise(li
    .withColumn('impression', when(col('event_name')== 'impression', lit(1)).otherwise(lit(0)))\
    .withColumn('viewable_impression', when(col('event_name')== 'viewable_impression', lit(1)).otherwise(lit(0)))\
    .withColumn('unmute', when(col('event_name')== 'unmute', lit(1)).otherwise(lit(0)))\
    .withColumn('mute', when(col('event_name')== 'mute', lit(1)).otherwise(lit(0)))\
    .withColumn('init', when(col('event_name')== 'init', lit(1)).otherwise(lit(0)))\
    .withColumn('error', when(col('event_name')== 'error', lit(1)).otherwise(lit(0)))\
    .withColumn('in_viewport_with_fallback_bid', when(col('event_name')== 'in_viewport_with_fallback_bid', lit(1)).othe
    .withColumn('third_quartile', when(col('event_name')== 'third_quartile', lit(1)).otherwise(lit(0)))\
    .withColumn('paused', when(col('event_name')== 'paused', lit(1)).otherwise(lit(0)))\
    .withColumn('ad_can_play', when(col('event_name')== 'ad_can_play', lit(1)).otherwise(lit(0)))\
    .withColumn('clicked', when(col('event_name')== 'clicked', lit(1)).otherwise(lit(0)))\
    .withColumn('resumed', when(col('event_name')== 'resumed', lit(1)).otherwise(lit(0)))\
    .withColumn('in_viewport_with_direct', when(col('event_name')== 'in_viewport_with_direct', lit(1)).otherwise(lit(0
    .withColumn('started', when(col('event_name')== 'started', lit(1)).otherwise(lit(0)))\
.groupby('day', 'line_item_id', 'pv_unique_id').sum().cache()
```

PySpark lesson 3

Communicating results

Numerous columns in PySpark:

```
In [40]: df.show()
```

```
+----------+-----------------+----------------------+--------------+------------------------+------------------+-----
--------------+-------------------------+-----------------------------------+--------------------+--------------------
----------+---------------+-------------+-------------+---------------+-----------------------------------+---
--------------------+---------------+--------------------+-----------------+-----------------+--------------------
-------------+-----------------+
|       day|sum(line_item_id)|sum(sum(line_item_id))|sum(sum(ready))|sum(sum(first_quartile))|sum(sum(midpoint))|sum
(sum(completed))|sum(sum(play_triggered))|sum(sum(in_viewport_without_offer))|sum(sum(impression))|sum(sum(viewable_i
mpression))|sum(sum(unmute))|sum(sum(mute))|sum(sum(init))|sum(sum(error))|sum(sum(in_viewport_with_fallback_bid))|su
m(sum(third_quartile))|sum(sum(paused))|sum(sum(can_play))|sum(sum(clicked))|sum(sum(resumed))|sum(sum(in_viewport
_with_direct))|sum(sum(started))|
+----------+-----------------+----------------------+--------------+------------------------+------------------+-----
--------------+-------------------------+-----------------------------------+--------------------+--------------------
----------+---------------+-------------+-------------+---------------+-----------------------------------+---
--------------------+---------------+--------------------+-----------------+-----------------+--------------------
-------------+-----------------+
|2019-01-17| 552893084971621|     1749873995772277|        123590|                    5981|              4796|
          3638|                10791|                              61773|                9379|
      5608|             20|           14|       124941|            520|                                8326|
                4134|           4280|                4253|               46|            3112|
         2382|             9381|
|2019-01-15| 620904626739261|     1934063545835394|        138119|                    5751|              4379|
          3049|                11116|                              69916|               10022|
```

Who are the end users?

- Fellow analyst
- My team members
- Other teams

What do they need?

My team:

- Code review
- Reuse the notebook for similar analysis

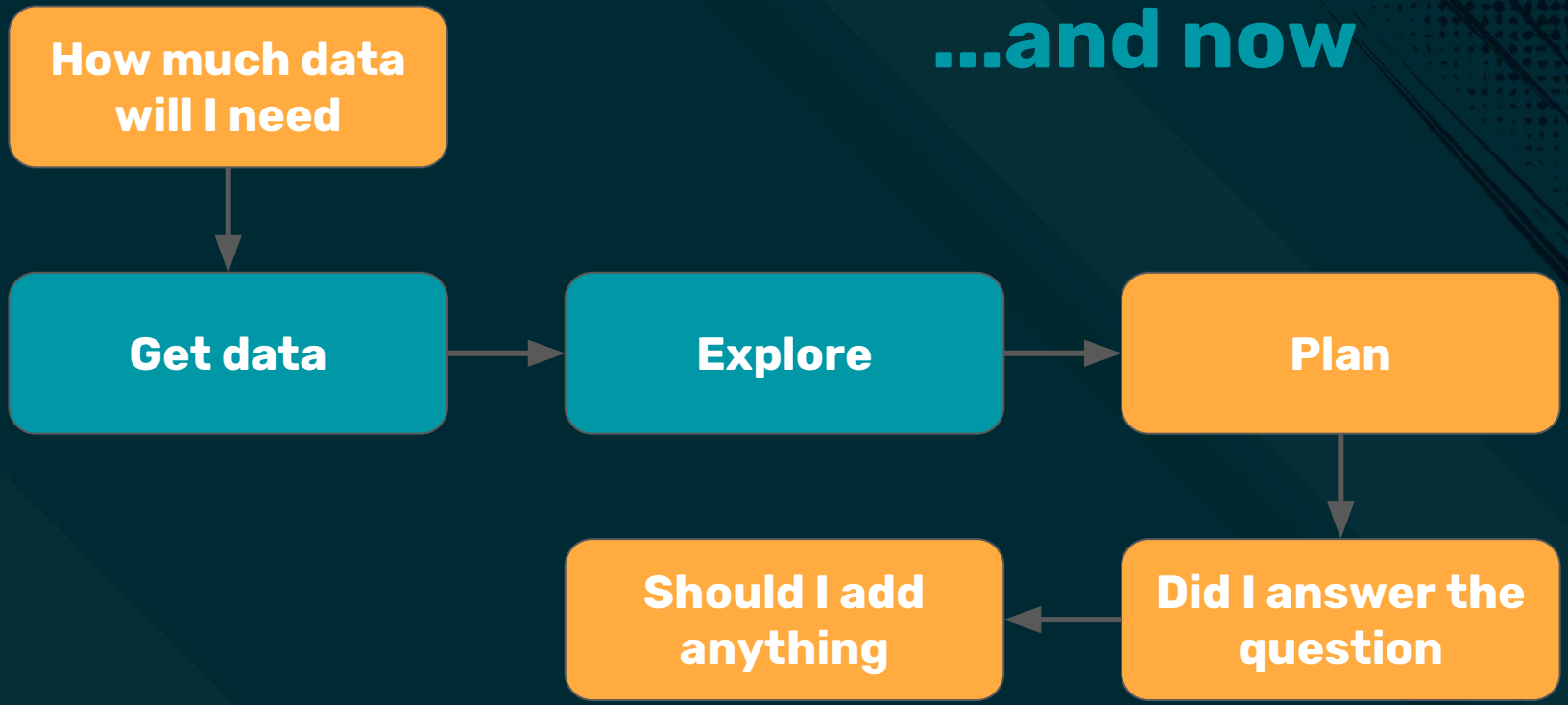
Other teams:

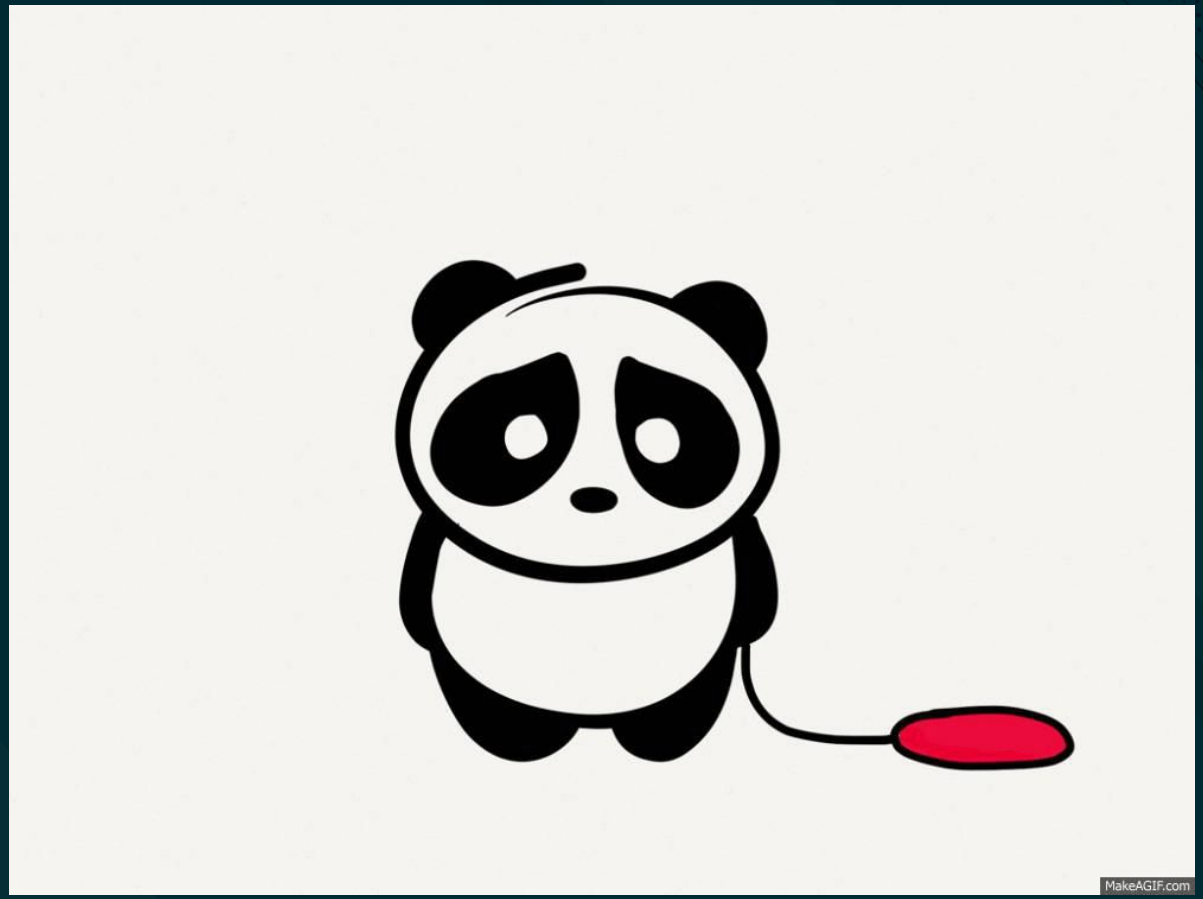
- Clear information what was analysed and what is the result

A better analyst

Previously...


Get data
Explore
What I need to do with the data
What else can I add
Did I answer the question


How much data will I need
...and now
Get data
Explore
Plan
Did I answer the question
Should I add anything


MakeAGIF.com

- Online resources rarely underline readability of notebooks
- Inefficient code is not “punished”
- Encourages overusing some functions

Thank You