

Analysing a terabyte of game data

by Rimma Shafikova,

a data scientist at  **VGW**TM
VIRTUAL GAMING WORLDS

YOW! DATA 2021

Not impressive


Analysing a terabyte of data

YOW! DATA 2021

Analysing a terabyte of data

Analysing a ~~terabyte~~ of data *petabyte*

YOW! DATA 2021



Unstructured data

Analysing a terabyte of game data

unstructured

The background of the slide is a dense, textured pattern of numerous book spines, creating a sense of vastness and information. The spines are in various colors, including shades of yellow, brown, and white, and are oriented in different directions, giving the background a dynamic, layered appearance.

Unstructured data

- is not stored in a database
- an SQL query is not going to cut it
- is the largest share of all data in the world



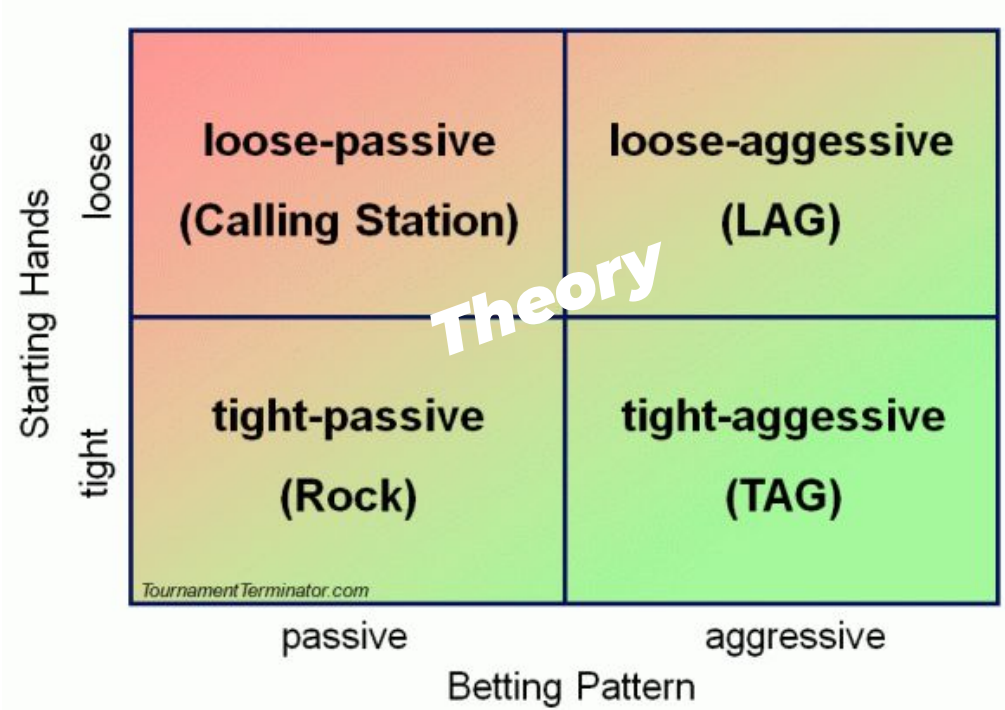
Préférence, a painting by Viktor Vasnetsov



Préférence, a painting by Viktor Vasnetsov

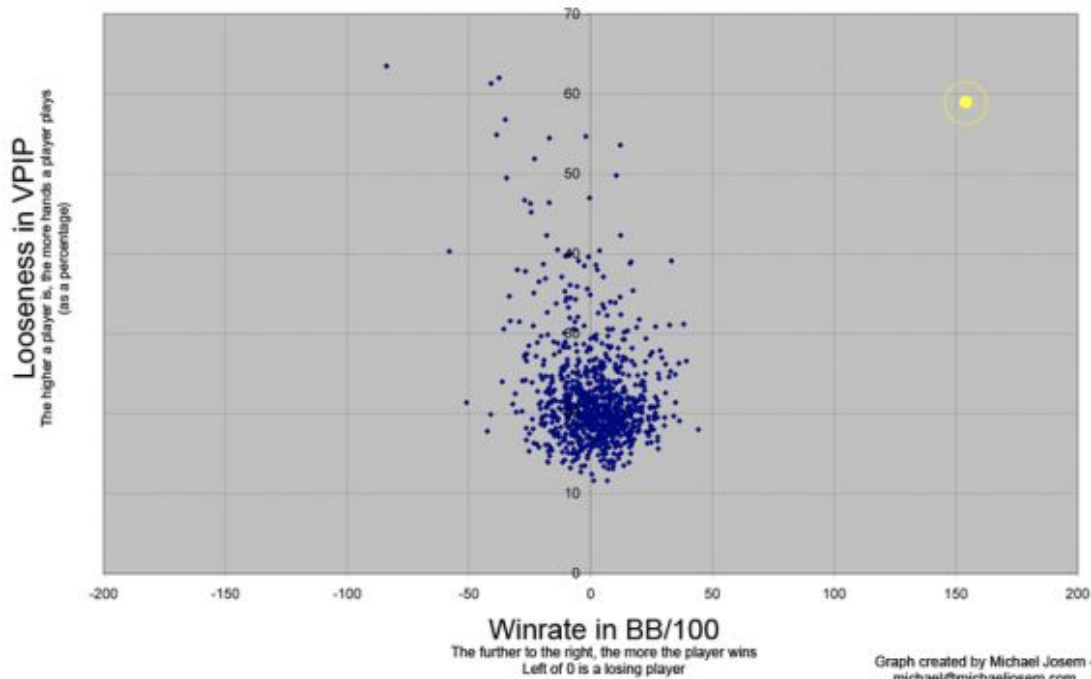


Example of poker stats usage: profiling



USE CASE

One graph that brought down an entire poker network



USE CASE

```
Poker      Game #35824123646: Hold'em No Limit ($0.05/$0.10 USD) - 2009/11/25 6:21:20 ET
Table 'May IV' 6-max Seat #4 is the button
Seat 1: Gles65 ($9.60 in chips)
Seat 2: shaunsaville ($2.65 in chips)
Seat 3: Pho3nix.two ($6.60 in chips)
Seat 4: k1tt9nM1nd ($10.10 in chips)
Seat 6: Hero ($9.90 in chips)
Hero: posts small blind $0.05
Gles65: posts big blind $0.10
shaunsaville: posts big blind $0.10
*** HOLE CARDS ***
Dealt to Hero [9s 6h]
shaunsaville: checks
Pho3nix.two: calls $0.10
k1tt9nM1nd: folds
Hero: folds
Gles65: checks
*** FLOP *** [Jh 2c Qc]
Gles65: checks
shaunsaville: checks
Pho3nix.two: bets $0.30
Gles65: folds
shaunsaville: folds
Uncalled bet ($0.30) returned to Pho3nix.two
Pho3nix.two collected $0.35 from pot
Pho3nix.two: doesn't show hand
*** SUMMARY ***
Total pot $0.35 | Rake $0
Board [Jh 2c Qc]
Seat 1: Gles65 (big blind) folded on the Flop
Seat 2: shaunsaville folded on the Flop
Seat 3: Pho3nix.two collected ($0.35)
Seat 4: k1tt9nM1nd (button) folded before Flop (didn't bet)
Seat 6: Hero (small blind) folded before Flop
```

$$\text{Playing Stat} = \frac{\text{Certain action (e.g. raising pre-flop)}}{\text{Chances to perform such action}}$$

$$\text{Playing Stat} = \frac{\text{Certain action (e.g. raising pre-flop)}}{\text{Chances to perform such action}}$$

Logic A

Logic B

*Daily,
monthly?*

Playing Stat

=

Certain action (e.g. raising pre-flop)

Chances to perform such action

Logic A

Logic B

*Daily,
monthly?*

Logic A

$$\text{Playing Stat} = \frac{\text{Certain action (e.g. raising pre-flop)}}{\text{Chances to perform such action}}$$

*It matters how it changes
depending on the position
(UTG, SB, BB)*

Logic B

*How does it change when
playing against certain
player?*

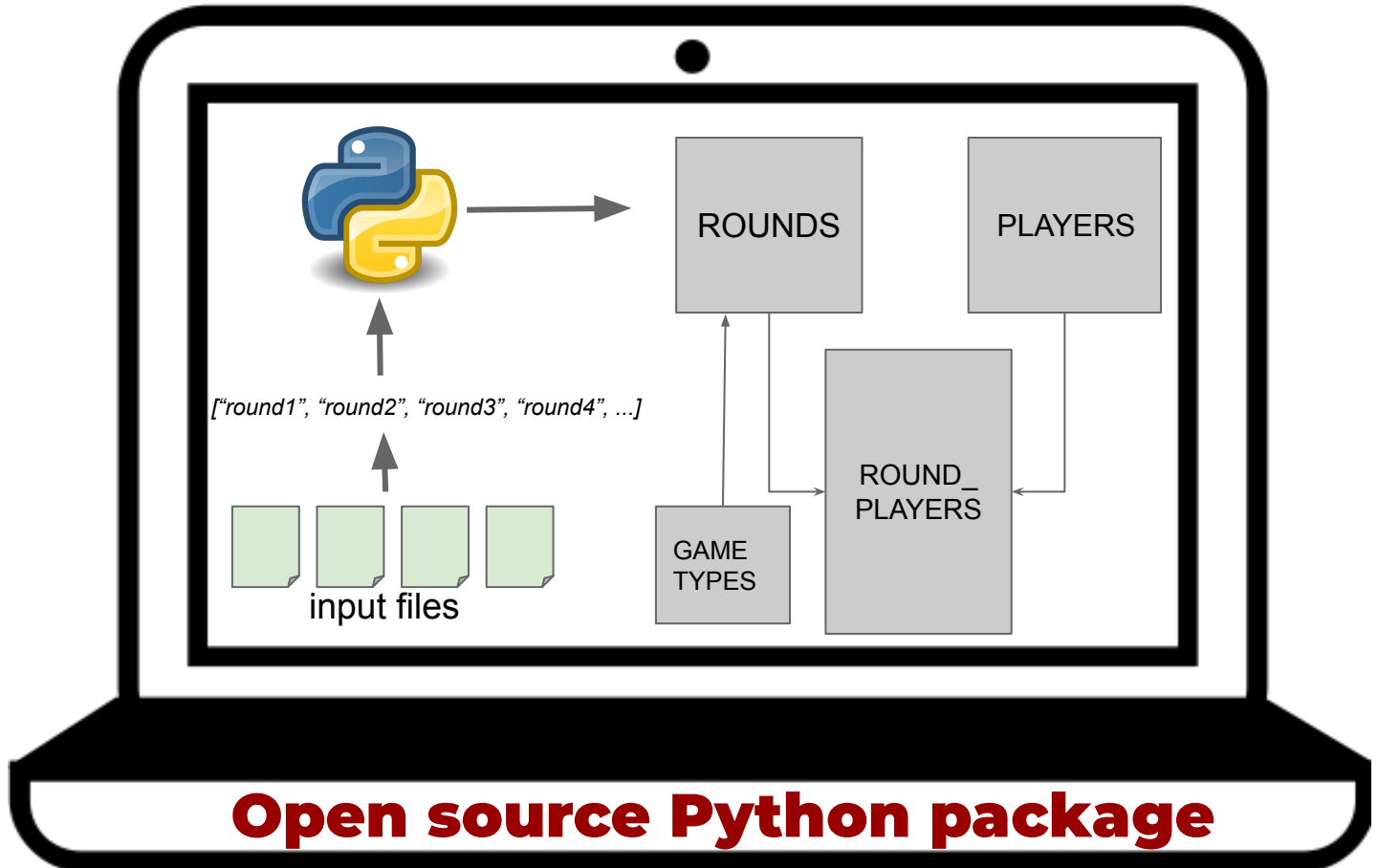
*Daily,
monthly?*

Logic A

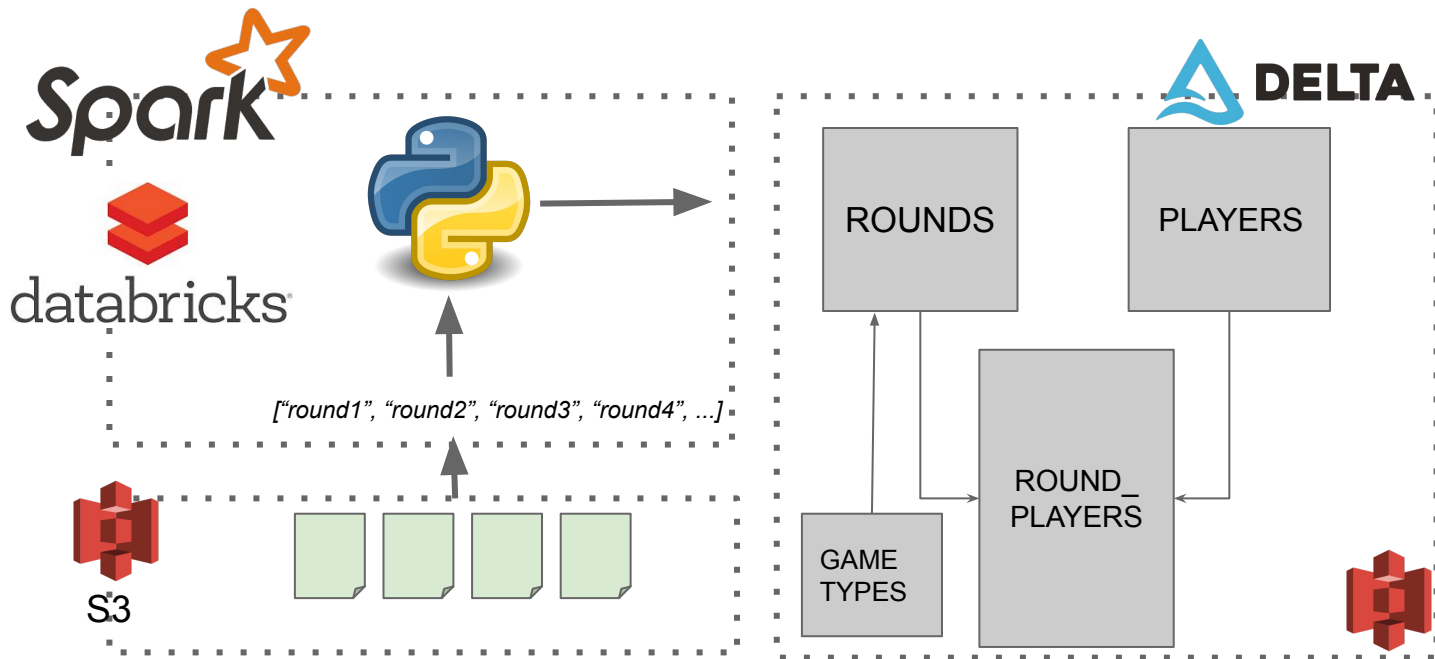
$$\text{Playing Stat} = \frac{\text{Certain action (e.g. raising pre-flop)}}{\text{Chances to perform such action}}$$

*It matters how it changes
depending on the position
(UTG, SB, BB)*

Logic B



Open source Python package





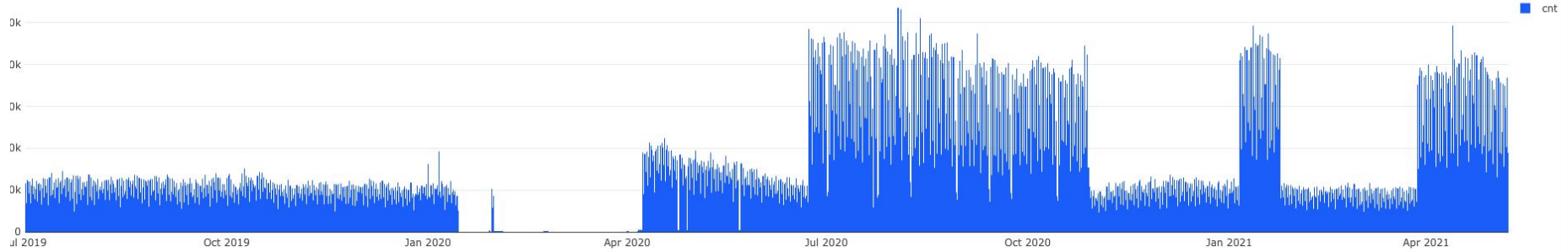
The fresh cavalier, fragment from a painting by Pavel Fedotov

How long will it take to backfill?

★ Poker Stats Monitor Upload [gi](#) [GI](#) [fpdb](#)

Refresh ⌵ ⌵ ⌵

fpdb_hands_optimized_counts





Alyonushka, a painting by Viktor Vasnetsov

How long will it take to backfill?

Backlog of N hours

processing at S hours per hour

$$\frac{N}{s - 1}$$

How long will it take to backfill?

$$\frac{N}{s} + \frac{N/s}{s}$$

How long will it take to backfill?

$$\frac{N}{s} + \frac{N/s}{s} + \frac{N}{s^3} + \dots$$

How long will it take to backfill?

$$\frac{N}{s} + \frac{N/s}{s} + \frac{N}{s^3} + \dots = N \sum_{i=1}^{\infty} \frac{1}{s^i} = \frac{N}{s-1}$$

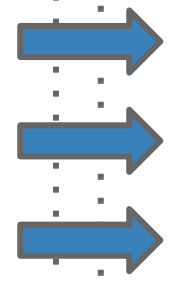
answer: DAYS

HOW CAN WE MAKE IT FASTER?



Spark

databricks

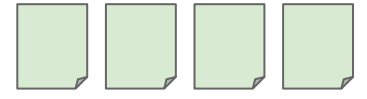


DELTA LAKE

output tables



Input files



Delta table: parquet files with glory

```
spark.sql(f"""CREATE TABLE {table_name}
(
  Id BIGINT,
  gametypeId INTEGER,
  sessionId INTEGER,
  fileId INTEGER,
  startTime TIMESTAMP,
)
USING DELTA LOCATION '{s3_location}'
""")
```



DELTA LAKE



Objects (863)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 Inventory](#) to get a list of all objects in your bucket [more](#)

Show versions

<input type="checkbox"/>	Name		Type
<input type="checkbox"/>	._delta_log/		Folder
<input type="checkbox"/>	part-00000-0224cf4e-4b71-42	i1c916d276-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-027b9780-12ab-4	ld16852f4c-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-0363cd5d-e0e5-4f	laa9e4a5bb-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-03d1cba7-3c3b-4f	722c64b60f-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-040ef92a-b276-43	id79d1b052-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-0436f024-9d63-4e	392771035-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-04b03d8c-6e8c-47	4aab27447-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-055a7f3f-e53d-47	f06b57d21-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-07e7e976-1cd1-42	iaf9516e70-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-08bc6b80-698b-4	9d5fdbb8c-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-08ee31c7-e420-4e	ifdc6d9e9c-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-092cba02-ab2b-4e	31f9ae063-c000.snappy.parquet	parquet
<input type="checkbox"/>	part-00000-0987bf05-b70b-4e	i511f0f01-c000.snappy.parquet	parquet

Delta table: partition!

```
spark.sql(f"""CREATE TABLE {table_name}
  (
    Id BIGINT,
    gametypeId INTEGER,
    sessionId INTEGER,
    fileId INTEGER,
    startTime TIMESTAMP,
  )
  USING DELTA LOCATION '{s3_location}'
  PARTITIONED BY startTime
""")
```

Delta table: partition.



DELTA LAKE



```
spark.sql(f"""CREATE TABLE {table_name}
(
  Id BIGINT,
  gametypeId INTEGER,
  sessionId INTEGER,
  fileId INTEGER,
  startTime TIMESTAMP,
  startDate DATE
)
USING DELTA LOCATION '{s3_location}'
PARTITIONED BY startDate
""")
```



Find objects by prefix Show view

<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	_delta_log/	Folder
<input type="checkbox"/>	startDate=2018-01-29 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-01-30 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-01-31 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-01 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-02 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-03 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-04 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-05 00%3A00%3A00/	Folder
<input type="checkbox"/>	startDate=2018-02-06 00%3A00%3A00/	Folder

Delta table: partition and use in queries

```
1 %sql
2 with dups as (
3 SELECT h.id, count(1) as cnt, min(startDate) as mind, max(startDate) as maxd
4 FROM delta_... b_hands_optimized h
5 WHERE startDate = '2018-01-30 00:00:00'
6 GROUP BY 1
7 HAVING cnt > 1
8 )
9 INSERT INTO delta_... _hands_for_merge
10 SELECT distinct h.* from delta_... _hands_optimized h WHERE exists (SELECT id from dups)
```

▶ (6) Spark Jobs

Cancelled

Command took 1.67 hours -- by rimma.shafikova@vgw.co at 22/03/2021, 14:23:08 on ds-rimma-cluster

Cmd 4

```
1 %sql
2 with dups as (
3 SELECT h.id, count(1) as cnt, min(startDate) as mind, max(startDate) as maxd
4 FROM delta_... b_hands_optimized h
5 WHERE startDate = '2018-01-30 00:00:00'
6 GROUP BY 1
7 HAVING cnt > 1
8 )
9 INSERT INTO delta_... _hands_for_merge
10 SELECT distinct h.* from delta_... _hands_optimized h WHERE exists (SELECT id from dups and h.startDate = '2018-01-30 00:00:00')
```

▶ (13) Spark Jobs

OK

Command took 49.71 seconds -- by rimma.shafikova@vgw.co at 22/03/2021, 16:28:46 on ds-rimma-cluster

Cmd 5

How long will it take to backfill?

answer: fewer DAYS

Can we make it faster?

Configure New Cluster

8 Workers: 244.0 GB Memory, 32 Cores, 8 DBU

1 Driver: 30.5 GB Memory, 4 Cores, 1 DBU ⓘ

Cluster Mode ⓘ

Standard | ▼

UI | [JSON](#)

Pool ⓘ

None | ▼

Databricks Runtime Version ⓘ

Runtime: 7.3 LTS (Scala 2.12, Spark 3.0.1) | ▼

Autopilot Options

Enable autoscaling ⓘ

Enable autoscaling local storage ⓘ

Worker Type ⓘ

i3.xlarge

30.5 GB Memory, 4 Cores, 1 DBU | ▼

Workers

8

Driver Type

Same as worker

30.5 GB Memory, 4 Cores, 1 DBU | ▼



At the school door,
a painting by Nikolay Bogdanov-Belsky

```
>>> huge_df.join(large_df, "ID").show(5)
```

```
>>> df = hugedf.toPandas()
```

```
>>> from sklearn.ensemble import RandomForestClassifier  
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)  
>>> clf.fit(X, y)
```



**It is not
just a notebook
in the cloud!**

I just created a 20 node Spark cluster and my pandas code doesn't run any faster. What is going wrong?

If you are working with any single-node libraries, they will not inherently become distributed when you switch to using Databricks.

1. **Native Spark:** if you're using Spark data frames and libraries (e.g. MLlib), then your code will be parallelized and distributed natively by Spark.
2. **Koalas:** Alternatively, you can use [Koalas](#), which allows you to use the pandas DataFrame API to access data in Apache Spark DataFrames.
3. **Pandas UDFs:** A new feature in Spark that enables parallelized processing on Pandas data frames within a Spark environment.

Native Spark

```
val oneMonthAgo = now.plusDays(-30);
val numberOfDays = Days.daysBetween(oneMonthAgo, now).getDays()
val dtfOut = DateTimeFormat.forPattern("yyyyMMdd")
for (f<- 0 to numberOfDays-1) {
    var n = dtfOut.print(oneMonthAgo.plusDays(f))
    var df = spark.read.parquet("s3a://location/day="+n)
    initialDF = initialDF.union(df)
}
```



koalas json



The actual koalas search
result first appearance



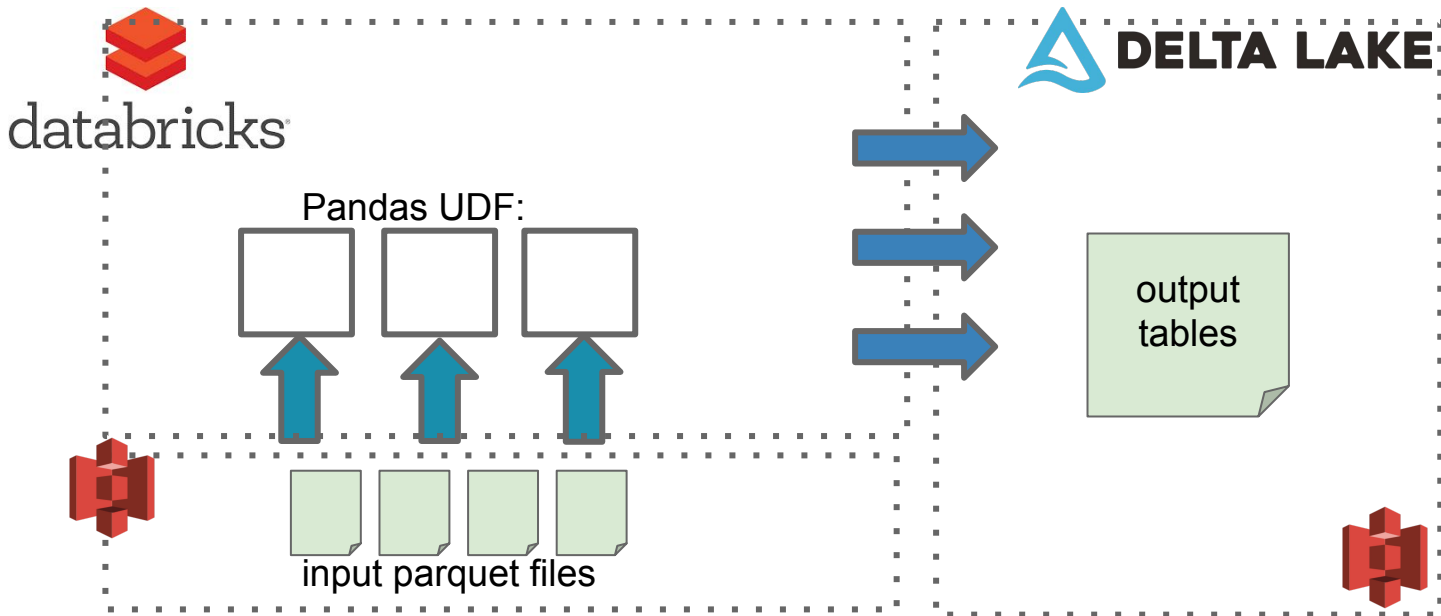


pandas json



The actual pandas search
result first appearance





Pandas UDF

```
@pandas_udf(schema, PandasUDFType.GROUPED_MAP)
def analyse_players(df):
    def CustomFunction():
        #do something
    def TakeOneReturnMany(row, row_accumulator):
        split_row = row[target_column]
        for s in split_row:
            new_row = {}
            new_row['ID'] = row['pokerId']
            new_row[new_columns_names[0]] = s[new_columns_names[0]]
            row_accumulator.append(new_row)
    new_rows = []
    df.apply(TakeOneReturnMany, axis=1, args = (new_rows))
    new_df = pd.DataFrame(new_rows)
    return new_df
```

Using Pandas UDF

```
result_df = df.groupby('hour').apply(analyse_players)
```




New planet by Konstantin Yuon

Conclusion

1. Partition your data and tell spark to use those partitions.
2. Rewrite in Native Spark if possible (pyspark, scala, etc)
3. If porting custom code: strip it down to the bare essentials.
4. Wrap those bare essentials in Pandas UDF