

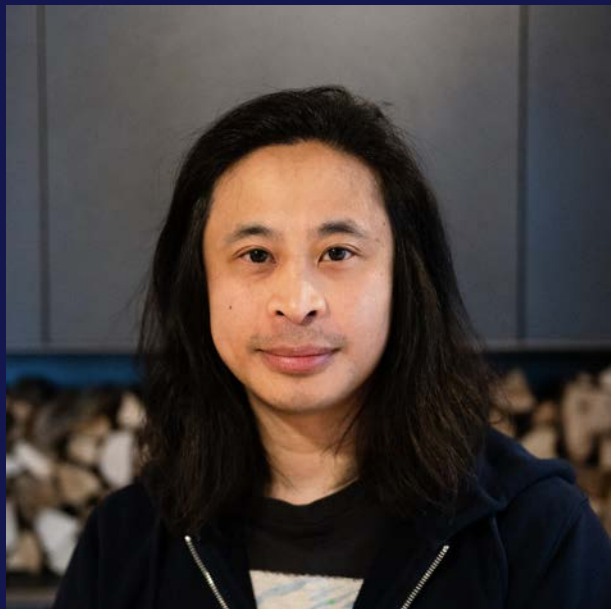


CONF42

# Sketching Algorithms

Making Sense of Big Data in a Single Stroke

# Hello 🖐️



## Tun Shwe

VP of Data / DevRel



🇬🇧 Quix (London, UK)



🚀 High growth startups

- Data engineering (Python, Spark, Kafka)
- Analytics engineering (OLAP, dbt, \*, MDS)
- Data strategy (streaming-centric)



📖 Areas of interest

- Less is more
- Real-time data and AI ecosystem

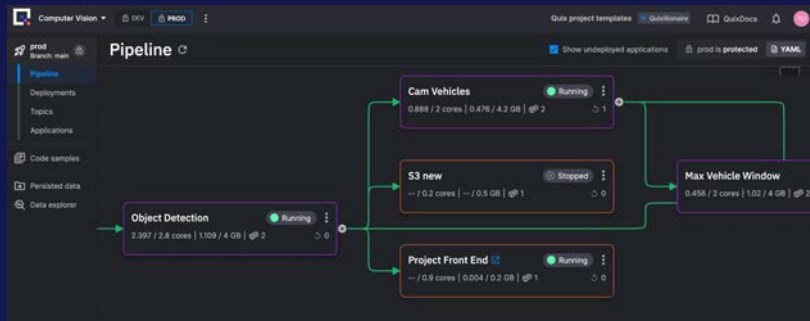
# Quix

## Python-first stream processing



## Quix Streams

Open source library for building stream processing applications using pure Python. Streaming DataFrames inspired by Pandas to easily move from batch to streaming.



## Quix Cloud

Platform to quickly build, deploy and monitor streaming ETL pipelines without having to manage infrastructure. Integrate data quickly with open source connectors.

# Quix Streams

Open Source library with Streaming DataFrames and stateful window operations

```
from quixstreams import Application, State

app = Application(broker_address="localhost:9092")

input_topic = app.topic("my_input_topic")
output_topic = app.topic("my_output_topic")

# Create a Streaming DataFrame
sdf = app.dataframe(topic=input_topic)

sdf["field_C"] = sdf.apply(lambda value: value["field_A"] + value["field_B"])

sdf = sdf.to_topic(output_topic)

if __name__ == "__main__":
    app.run(sdf)
```

# Quix Cloud

Streaming data pipelines with fully managed Kafka, Kubernetes and Git



# What is a sketch?





# What is a sketch?

Sketches are

**small**

**stateful**

**streaming** data programs

that


deal with a **huge volume of data**

and provide **approximate answers**

orders of magnitude **faster**

than

exact techniques



# $\approx$ Approximate answers

Favouring approximate solutions over exact solutions enables real time analysis over huge volumes of data (using fewer resources)



 You

Sketching algorithms are great, am I right?

 ChatGPT

Absolutely! Sketching algorithms are indeed great tools in the field of computer science and data analysis. They offer efficient and often approximate solutions to complex problems, particularly in scenarios where exact solutions are computationally expensive or impractical. Sketching algorithms are commonly used in various areas such as:

1. **Streaming Algorithms:** Sketching algorithms are widely used in processing continuous data streams where storing the entire dataset is not feasible due to memory constraints. Examples include Count-Min Sketch for frequency estimation and the Flajolet-Martin algorithm for estimating the cardinality of a set.
2. **Approximate Query Processing:** In databases, sketching algorithms can be used to approximate query results quickly, especially for queries over large datasets. This allows for faster response times without sacrificing too much accuracy.
3. **Big Data Analytics:** With the rise of big data, traditional algorithms often struggle to process and analyze massive datasets efficiently. Sketching algorithms provide scalable solutions for tasks like summarization, sampling, and pattern discovery in large datasets.
4. **Machine Learning:** Sketching techniques can be integrated into machine learning algorithms to handle high-dimensional data or reduce the computational complexity of certain operations like matrix factorization and dimensionality reduction.
5. **Network Monitoring:** Sketching algorithms are used in network monitoring applications to estimate traffic characteristics, detect anomalies, and summarize network behavior efficiently.

Overall, sketching algorithms play a crucial role in enabling scalable and efficient data processing and analysis across various domains, making them an essential tool in the modern computational toolkit.

## ■ Sketch characteristics

### ■ Small

They have to be small. Usually only a few kilobytes in size, meaning reduced memory requirements and will grow sublinearly in space

### ● Stateful

Maintain a data structure that keeps the state of observations and does not retain the exact representation of data points

### ■ Streaming

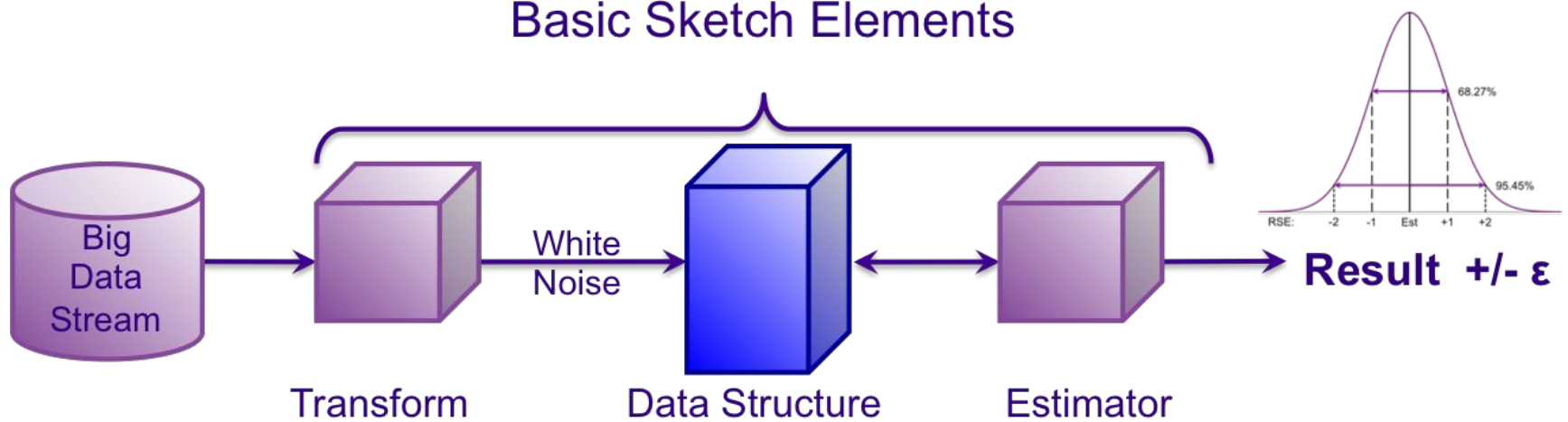
Deal with data in a single pass. The sketch needs to look at each item in the stream only once

### ■ Fast

Orders of magnitude faster than exact techniques. Results are mergeable without loss in accuracy

# ■ Sketch components

## Basic Sketch Elements



**Mathematically provable error bounds**

Why exact == slow



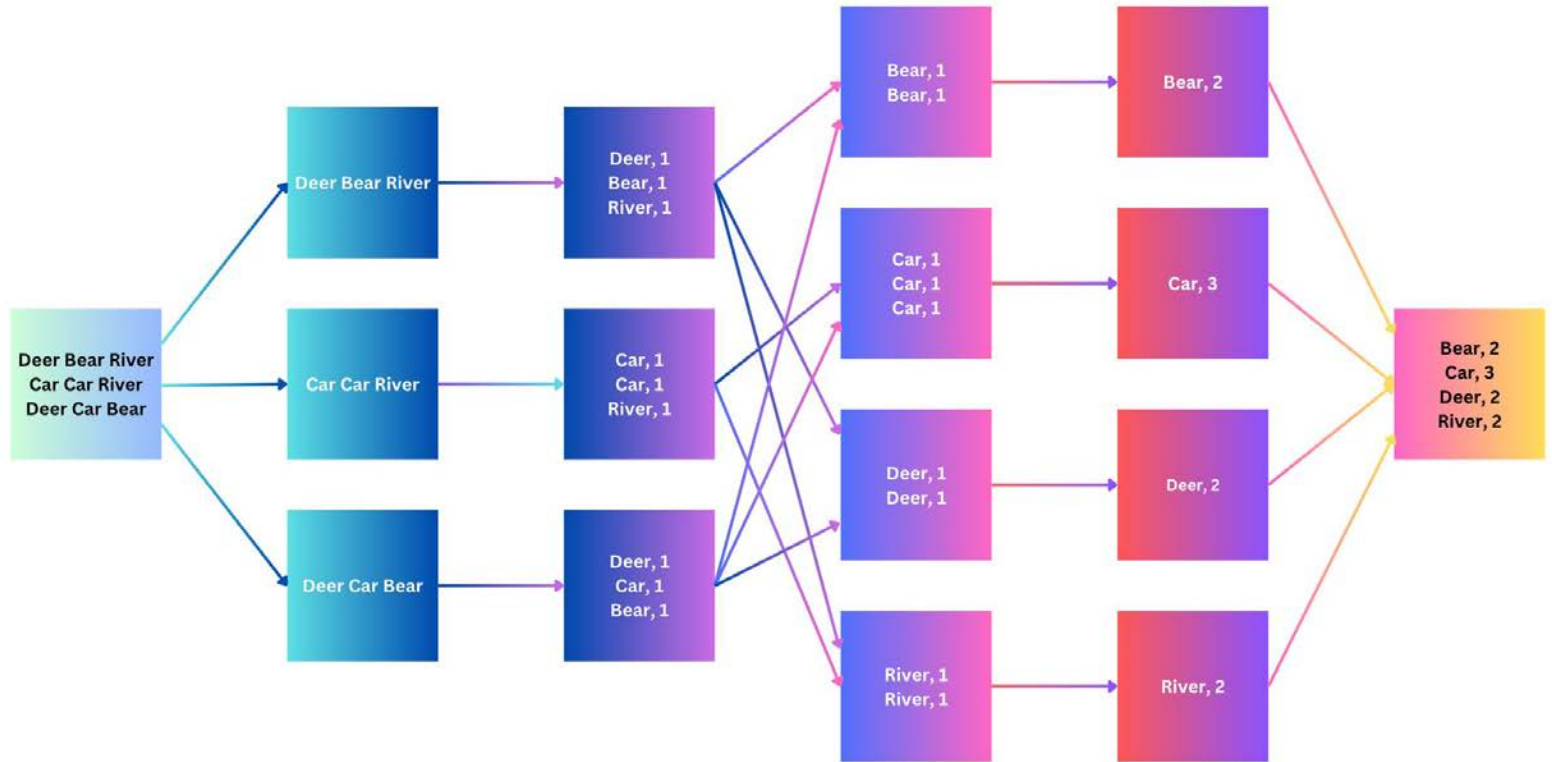


# Distributed processing

Hadoop, Spark, Snowflake and similar  
distributed processing engines/databases use  
parallelisation

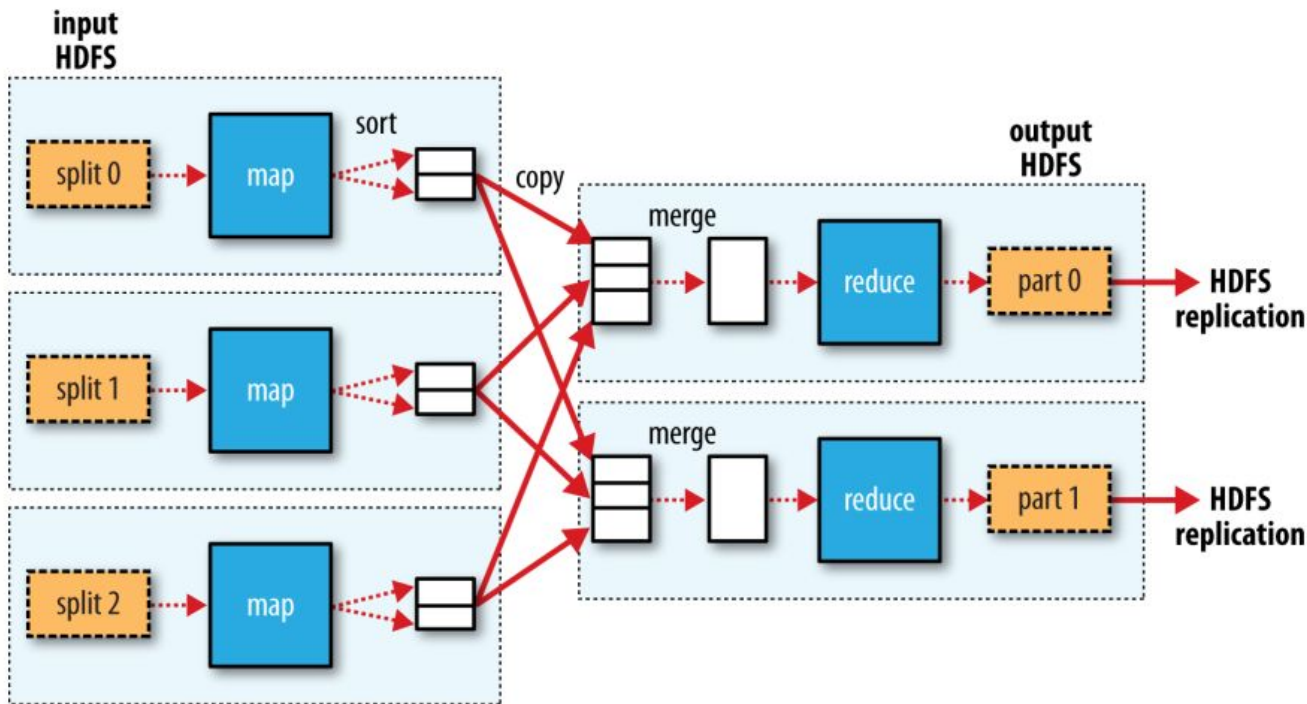
# Unique word count

Split → Map → Shuffle → Reduce



# Massively Parallel Processing (MPP)

MapReduce: Hadoop, Spark, Snowflake, etc







# Shuffling is slow

Network IO makes everything slow

# Latency numbers every programmer should know

## Latency Comparison Numbers (~2012)

|                                    |             |    |         |    |                                |
|------------------------------------|-------------|----|---------|----|--------------------------------|
| L1 cache reference                 | 0.5         | ns |         |    |                                |
| Branch mispredict                  | 5           | ns |         |    |                                |
| L2 cache reference                 | 7           | ns |         |    | 14x L1 cache                   |
| Mutex lock/unlock                  | 25          | ns |         |    |                                |
| Main memory reference              | 100         | ns |         |    | 20x L2 cache, 200x L1 cache    |
| Compress 1K bytes with Zip         | 3,000       | ns | 3       | us |                                |
| Send 1K bytes over 1 Gbps network  | 10,000      | ns | 10      | us |                                |
| Read 4K randomly from SSD*         | 150,000     | ns | 150     | us | ~1GB/sec SSD                   |
| Read 1 MB sequentially from memory | 250,000     | ns | 250     | us |                                |
| Round trip within same datacenter  | 500,000     | ns | 500     | us |                                |
| Read 1 MB sequentially from SSD*   | 1,000,000   | ns | 1,000   | us | 1 ms ~1GB/sec SSD, 4X memory   |
| Disk seek                          | 10,000,000  | ns | 10,000  | us | 10 ms 20x datacenter roundtrip |
| Read 1 MB sequentially from disk   | 20,000,000  | ns | 20,000  | us | 20 ms 80x memory, 20X SSD      |
| Send packet CA->Netherlands->CA    | 150,000,000 | ns | 150,000 | us | 150 ms                         |

## Notes

1 ns =  $10^{-9}$  seconds

1 us =  $10^{-6}$  seconds = 1,000 ns

1 ms =  $10^{-3}$  seconds = 1,000 us = 1,000,000 ns

## Credit

By Jeff Dean: <http://research.google.com/people/jeff/>

Originally by Peter Norvig: <http://norvig.com/21-days.html#answers>

# Why sketches == fast



# ■ 4 main parts of sketching

## ■ Stream processor

Process data as it streams in with random algorithmic selection

## ● Data structure

Start off with an empty summary and update it with each data point over time. The size grows sublinearly

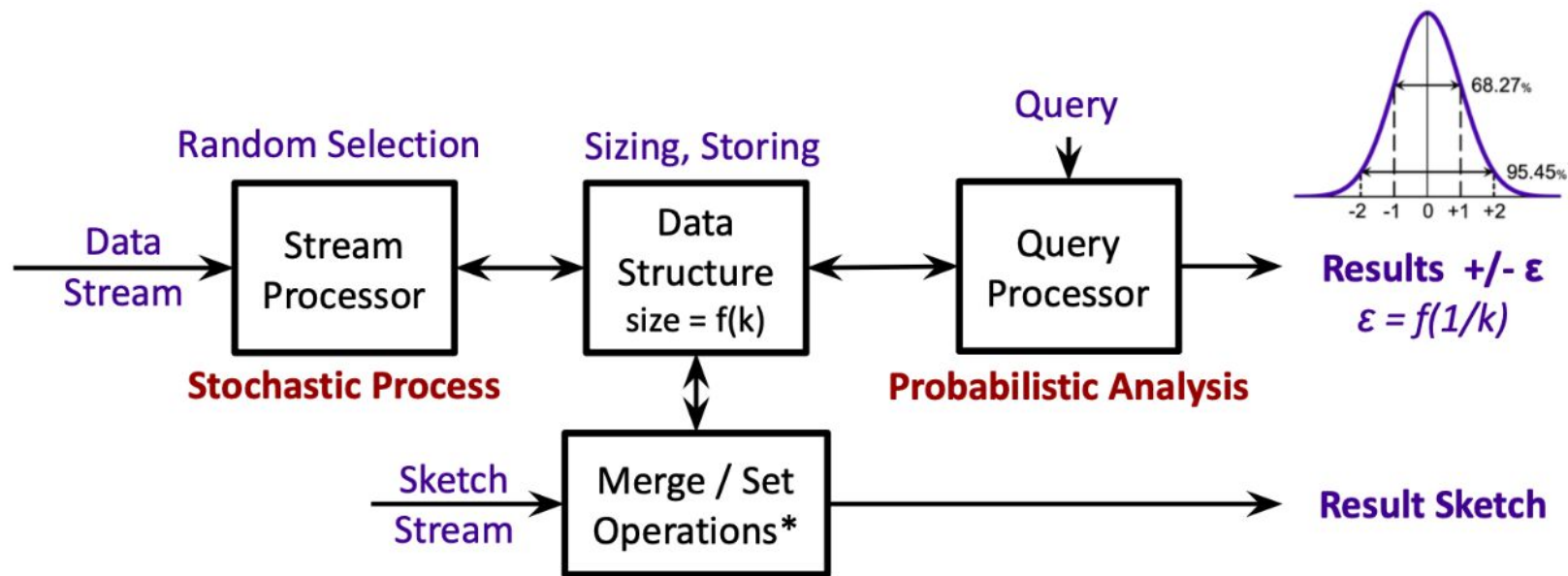
## ■ Query processor

Compute the desired result and model error properties using probability/statistics

## ■ Merge/set operations

Operations on multiple sketches without loss in accuracy

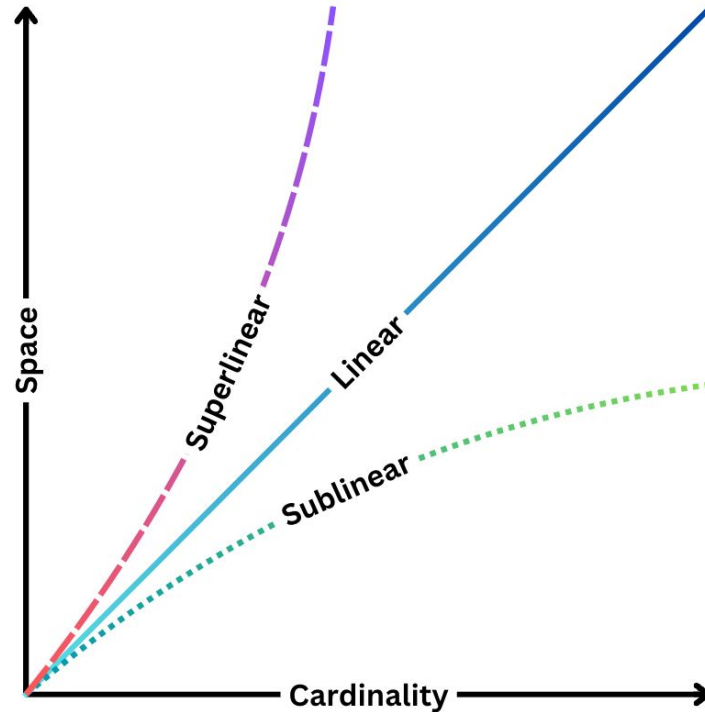
# Sketch design



A Single Sketch Contains Many Algorithms

# ■ Sublinear data structure growth

As the input stream increases in size, sketch size remains smaller than linear





# Mergability

Being able to merge multiple sketches without loss in accuracy solves non-additive challenges

## ■ Non-additive challenges are everywhere

| Store name          | Sales             | Unique products count |
|---------------------|-------------------|-----------------------|
| Quix London Store   | 10,560,299        | 1,567                 |
| Quix Paris Store    | 8,297,433         | 1,315                 |
| Quix New York Store | 7,312,654         | 1,108                 |
| Quix Tokyo Store    | 7,117,852         | 986                   |
| <b>Total</b>        | <b>33,288,238</b> | <b>4,976</b>          |



## ■ Unique counts are non-additive

| Store name          | Sales             | Unique products count  |
|---------------------|-------------------|--|
| Quix London Store   | 10,560,299        | 1,567  |
| Quix Paris Store    | 8,297,433         | 1,315  |
| Quix New York Store | 7,312,654         | 1,108  |
| Quix Tokyo Store    | 7,117,852         | 986  |
| <b>Total</b>        | <b>33,288,238</b> | <b>4,976</b>  |

# Non-additive challenges solved

| Date       | Store name        | Sales  | Unique products count | Sketch  |
|------------|-------------------|--------|-----------------------|---|
| 01/02/2024 | Quix London Store | 31,457 | 1,427                 |  |
| 02/02/2024 | Quix London Store | 37,592 | 1,431                 |  |
| 03/02/2024 | Quix London Store | 42,888 | 1,445                 |  |
| 04/02/2024 | Quix London Store | 29,749 | 1,445                 |  |
| ...        | ...               | ...    | ...                   | ...   |

| Date       | Store name       | Sales  | Unique products count | Sketch  |
|------------|------------------|--------|-----------------------|---|
| 01/02/2024 | Quix Paris Store | 20,273 | 1,210                 |  |
| 02/02/2024 | Quix Paris Store | 25,766 | 1,252                 |  |
| 03/02/2024 | Quix Paris Store | 26,819 | 1,261                 |  |
| 04/02/2024 | Quix Paris Store | 21,521 | 1,260                 |  |
| ...        | ...              | ...    | ...                   | ...   |

| Date     | Store name        | Sales   | Unique products count | Sketch  |
|----------|-------------------|---------|-----------------------|---|
| Feb 2024 | Quix London Store | 851,773 | 1,501                 |  |
| Feb 2024 | Quix Paris Store  | 653,021 | 1,275                 |  |

| Date     | Store name         | Sales     | Unique products count | Sketch  |
|----------|--------------------|-----------|-----------------------|---|
| Feb 2024 | Quix Europe Stores | 1,504,794 | 2,312                 |  |

# Types of Sketches



# ■ Types of sketches

## ■ Counting uniques

Estimating count distinct (Count Min), cardinality amongst many duplicates (Hyper-Log Log and Compressed Probabilistic Counting)

## ● Frequent items

Estimating heavy hitters, e.g. what are the top most viewed talks in this conference?

## ■ Quantiles/data distributions

Estimates distribution with well defined error bounds, e.g. how much time people spend on a web page

## ■ Sampling

Sampling keeps some resemblance or transformation of the item from the stream, e.g. reservoir sampling where you choose  $k$  samples from a list of  $n$  items

# Count Min sketch

Naive example

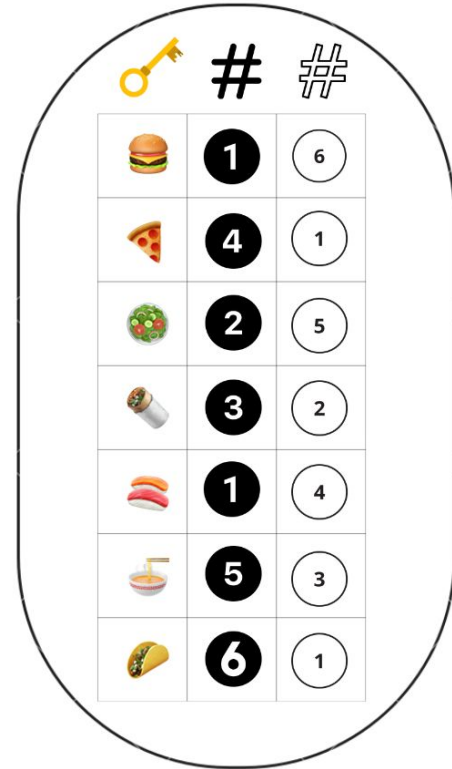


# Count Min sketch

With hashing



|   | # | # |
|---|---|---|
| 1 |   |   |
| 2 |   |   |
| 3 |   |   |
| 4 |   |   |
| 5 |   |   |
| 6 |   |   |



# Count Min sketch

With hashing



|   | # | # |
|---|---|---|
| 1 | 1 |   |
| 2 |   |   |
| 3 |   |   |
| 4 |   |   |
| 5 |   |   |
| 6 |   | 1 |

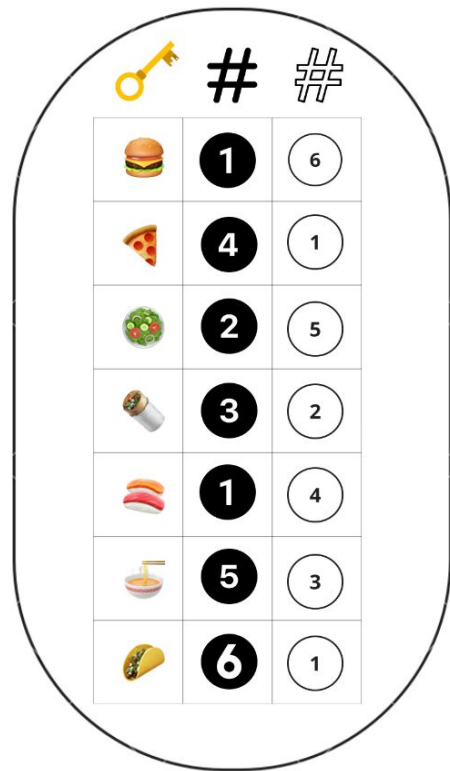
|  | # | # |
|--|---|---|
|  | 1 | 6 |
|  | 4 | 1 |
|  | 2 | 5 |
|  | 3 | 2 |
|  | 1 | 4 |
|  | 5 | 3 |
|  | 6 | 1 |

# Count Min sketch

With hashing



|   | # | # |
|---|---|---|
| 1 | 1 | 1 |
| 2 |   |   |
| 3 |   |   |
| 4 | 1 |   |
| 5 |   |   |
| 6 |   | 1 |





# Count Min sketch

With hashing



|   | # | # |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 |   |
| 3 |   |   |
| 4 | 2 |   |
| 5 |   | 1 |
| 6 |   | 1 |

| Key     | # | # |
|---------|---|---|
| Burger  | 1 | 6 |
| Pizza   | 4 | 1 |
| Salad   | 2 | 5 |
| Burrito | 3 | 2 |
| Sushi   | 1 | 4 |
| Ramen   | 5 | 3 |
| Taco    | 6 | 1 |

# Count Min sketch

With hashing



|          | # | # |
|----------|---|---|
| <b>1</b> | 3 | 4 |
| <b>2</b> | 1 | 1 |
| <b>3</b> | 1 | 1 |
| <b>4</b> | 3 | 2 |
| <b>5</b> | 1 | 1 |
| <b>6</b> | 1 | 1 |

|   | 🔑 | # | # |
|---|---|---|---|
| 🍔 | 1 | 6 |   |
| 🍕 | 4 | 1 |   |
| 🥗 | 2 | 5 |   |
| 🌯 | 3 | 2 |   |
| 🍣 | 1 | 4 |   |
| 🍜 | 5 | 3 |   |
| 🌮 | 6 | 1 |   |








# Count Min sketch

With hashing



 **3** **1** = **1**

|   | #        | #        |
|---|----------|----------|
| 1 | <b>3</b> | <b>4</b> |
| 2 | <b>1</b> | <b>1</b> |
| 3 | <b>1</b> | <b>1</b> |
| 4 | <b>3</b> | <b>2</b> |
| 5 | <b>1</b> | <b>1</b> |
| 6 | <b>1</b> | <b>1</b> |

|   | 🔑   | #        | # |
|---|---|----------|---|
| 1 |  | <b>1</b> | 6 |
| 2 |  | <b>4</b> | 1 |
| 3 |  | <b>2</b> | 5 |
| 4 |  | <b>3</b> | 2 |
| 5 |  | <b>1</b> | 4 |
| 6 |  | <b>5</b> | 3 |
| 7 |  | <b>6</b> | 1 |

# Count Min sketch

With hashing



$$\begin{array}{l} \text{burger} \quad \textcircled{3} \quad \textcircled{1} = \textcircled{1} \\ \text{pizza} \quad \textcircled{3} \quad \textcircled{4} = \textcircled{3} \end{array}$$

|   | # | # |
|---|---|---|
| 1 | 3 | 4 |
| 2 | 1 | 1 |
| 3 | 1 | 1 |
| 4 | 3 | 2 |
| 5 | 1 | 1 |
| 6 | 1 | 1 |

|         | key | # | # |
|---------|-----|---|---|
| burger  | 1   | 6 |   |
| pizza   | 4   | 1 |   |
| salad   | 2   | 5 |   |
| taco    | 3   | 2 |   |
| sushi   | 1   | 4 |   |
| ramen   | 5   | 3 |   |
| burrito | 6   | 1 |   |

# Open Source Sketches



- **Apache DataSketches (Java, C++, Python)**

[github.com/apache/datasketches-python](https://github.com/apache/datasketches-python)



- Apache DataSketches (Java, C++, Python)

[github.com/apache/datasketches-python](https://github.com/apache/datasketches-python)



Apache  
DataSketches

```
File Edit View Run Kernel Tabs Settings Help
sketching-spotify.ipynb x +
Python 3 (ipykernel)
[ ]: pip install datasketches pandas
[25]: from datasketches import count_min_sketch
import pandas as pd
[26]: df = pd.read_csv("spotify.csv")
print(f"Size: {df.size}")
df[["artist", "title"]].head()
Size: 78456008
[26]:
  artist      title
0  Bad Bunny, Jhay Cortez  Dakiti
1      Ariana Grande  positions
2      24kGoldn      Mood
3  Internet Money  Lemonade
4      Justin Bieber  Lonely
[27]: confidence = 0.90
num_hashes = count_min_sketch.suggest_num_hashes(confidence)
relative_error = 0.01
num_buckets = count_min_sketch.suggest_num_buckets(relative_error)
cms = count_min_sketch(num_hashes, num_buckets)
```



```
File Edit View Run Kernel Tabs Settings Help
sketching-spotify.ipynb x +
Python 3 (ipykernel)

[27]: confidence = 0.90
num_hashes = count_min_sketch.suggest_num_hashes(confidence)

relative_error = 0.01
num_buckets = count_min_sketch.suggest_num_buckets(relative_error)

cms = count_min_sketch(num_hashes, num_buckets)

for artist in df["artist"]:
    cms.update(str(artist))

print(cms)

### Count Min sketch summary:
num hashes      : 3
num buckets     : 272
capacity bins   : 816
filled bins     : 816
pct filled      : 100%
### End sketch summary

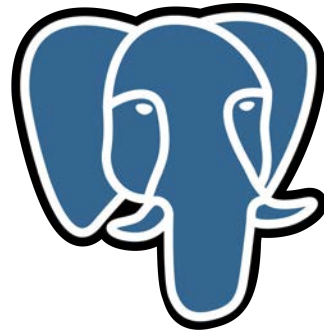
[34]: print(f"BTS           {cms.get_estimate('BTS')}")
print(f"Dua Lipa       {cms.get_estimate('Dua Lipa')}")
print(f"Taylor Swift   {cms.get_estimate('Taylor Swift')}")
print(f"Ariana Grande   {cms.get_estimate('Ariana Grande')}")

BTS           29667.0
Dua Lipa      28927.0
Taylor Swift  21832.0
Ariana Grande 17066.0

Simple 0 2 Python 3 (ipykernel) | Idle Mode: Command Ln 4, Col 60 sketching-spotify.ipynb 0
```

## ■ DataSketch extensions

Supported by databases, engines and has an active community



# Thank you



[github.com/quixio/quix-streams](https://github.com/quixio/quix-streams)  
[quix.io/slack-invite](https://quix.io/slack-invite)