Anomaly Detection in Trade Data Using SQL

https://github.com/brrttwrks/dataharvest/

Eric Barrett
Data Desk Manager

OCCRP
ORGANIZED CRIME AND CORRUPTION REPORTING PROJECT

DISCLAIMER

This is **not** an introduction to SQL.

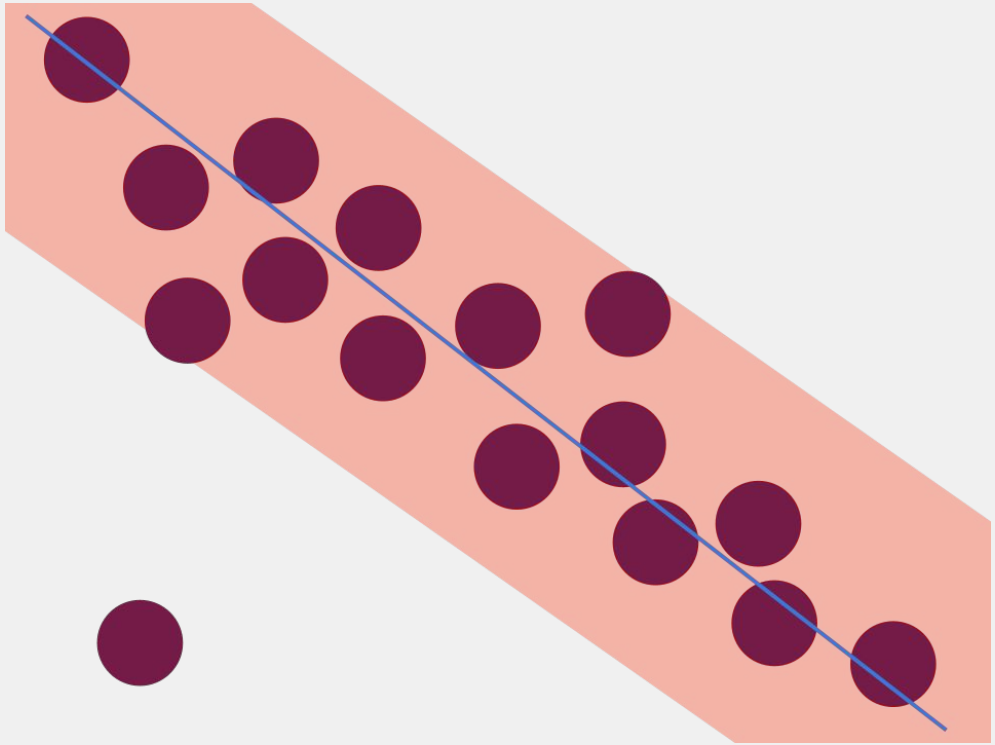
All are welcome to join us, but we are not teaching basic SQL statements and queries.

We will, however, build each SQL statement **step-by-step**.

Some topics we will touch on are:

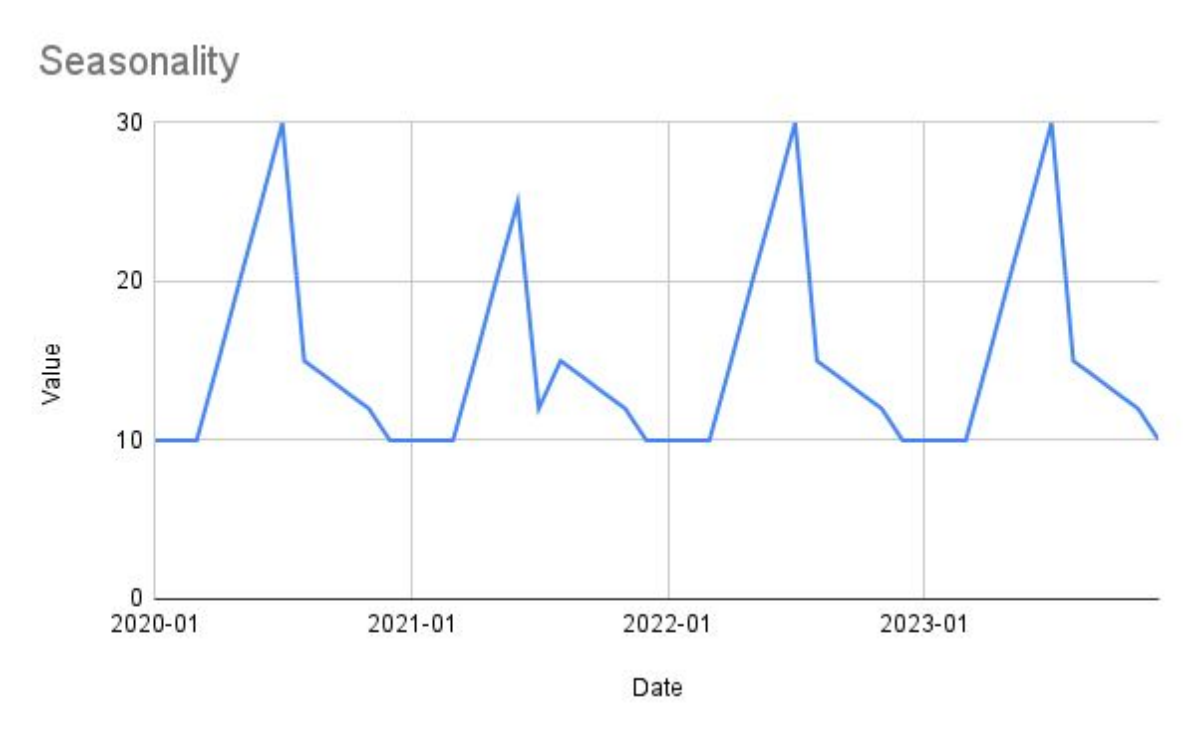
- time series data
- what is normal? basic statistical measures such as standard deviation and z-scores
- the difference and use of GROUP BY and window functions for analysis in SQL

What is normal anyway?



What is normal anyway?

What is normal anyway?



What is normal anyway?

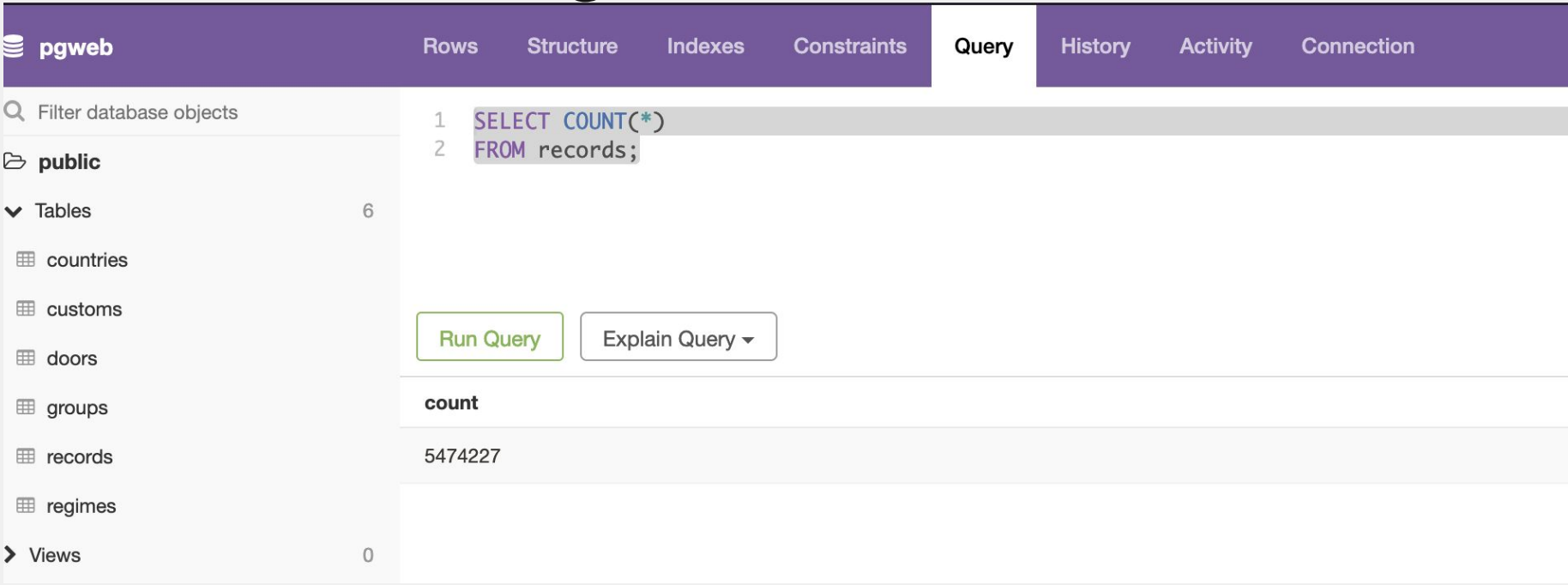
date	pb_ppb
2000	5
2001	5
2002	5
2003	5
2004	5
2005	4
2006	4
2007	4
2008	4
2009	4
2010	4
2011	10
2012	11
2013	5
2014	4
2015	4
2016	4
2017	4
2018	3
2019	3
2020	3

date	pb_ppb	zscore
2000	5	0.1813985435
2001	5	0.1813985435
2002	5	0.1813985435
2003	5	0.1813985435
2004	5	0.1813985435
2005	4	-0.5804753391
2006	4	-0.5804753391
2007	4	-0.5804753391
2008	4	-0.5804753391
2009	4	-0.5804753391
2010	4	-0.5804753391
2011	10	3.990767956
2012	11	4.752641839
2013	5	0.1813985435
2014	4	-0.5804753391
2015	4	-0.5804753391
2016	4	-0.5804753391
2017	4	-0.5804753391
2018	3	-1.342349222
2019	3	-1.342349222
2020	3	-1.342349222

Why SQL?

- not subject to the low size limits that spreadsheets are (larger-then-memory data is fine)
- fast
- freely available (Sqlite, PostgreSQL, …)
- SQL is tried and true standard for querying tabular data for decades
- easy to write queries - declarative statements
- easy to document queries

The Data - Georgian Trade Data



The Data - Georgian Trade Data

```
1 SELECT *
2 FROM records
3 LIMIT 50;
```

Run Query | Explain Query ▾ | 50 rows in 0 ms | JSON | CSV

year	month	hsn	qty	weight	value_usd	regime_id	group_id	door_id	country_id	customs_id
2022	1	01022110000	33.000	18000.000	67828.70403	1	1	1	233	69101
2022	1	01022110000	32.000	18500.000	88212.29572	1	1	1	348	69101
2022	1	01022110000	32.000	18500.000	66274.16523	1	1	1	348	69604
2022	1	01022999000	4.000	1825.000	5800.00000	1	1	1	804	69501
2022	1	01039219000	6728.000	797590.000	1203615.84800	1	1	1	643	69101

The Data - Georgian Trade Data

```
1 SELECT DISTINCT year
2 FROM records;
```

Run Query | Explain Query

year
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024

```
1 SELECT year,
2        COUNT(year)
3 FROM records
4 GROUP BY year;
```

Run Query | Explain Query

year	count
2008	376502
2009	408582
2010	444232
2011	241073
2012	231212
2013	234645
2014	293742
2015	256660
2016	265951
2017	355424
2018	374673
2019	380817
2020	324296
2021	366577
2022	415112
2023	474235
2024	30494

The Data - Georgian Trade Data

```
SELECT year,
       COUNT(year)
FROM records
GROUP BY year
ORDER BY count DESC;
```

Run Query | Explain Query

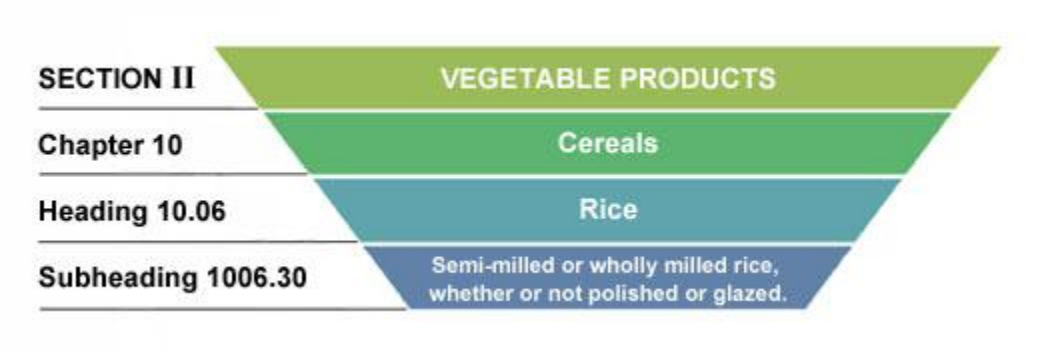
year	count
2023	474235
2010	444232
2022	415112
2009	408582
2019	380817
2008	376502
2018	374673
2021	366577
2017	355424
2020	324296
2014	293742
2016	265951
2015	256660
2011	241073
2013	234645
2012	231212
2024	30494

What are HS Codes?

The **Harmonized Commodity Description and Coding System**, also known as the **Harmonized System** (**HS**) of tariff nomenclature is an internationally standardized system of names and numbers to classify traded products.

https://en.wikipedia.org/wiki/Harmonized_System

What are HS Codes?



Increasing Specificity

- 02
- 0210
- 021006
- 02100630
- ...

https://en.wikipedia.org/wiki/Harmonized_System

What are HS Codes?

```
1  SELECT hsn,
2         substr(hsn, 1, 2) AS level
3  FROM records
4  WHERE hsn LIKE '02%'
5  LIMIT 10;
```

Run Query | Explain Query

hsn	level
02012090000	02
02023090000	02
02023090000	02
02023090000	02
02023090000	02
02031211000	02
02032110000	02
02032110000	02
02032211000	02
02032211000	02

What are HS Codes?

```
SELECT *,
       substring(hsn, 1, 4) AS hsn_group
FROM records
WHERE hsn LIKE '02%'
LIMIT 10;
```

Run Query | Explain Query ▾ | 10 rows in 1 ms | JSON | CSV

year	month	hsn	qty	weight	value_usd	regime_id	group_id	door_id	country_id	customs_id	hsn_group
2022	2	02023090000	189014.290	189014.290	514859.63000	1	2	1	76	69501	0202
2022	2	02023090000	75168.000	75168.000	417304.56940	1	2	1	440	69601	0202
2022	2	02023090000	20073.500	20073.500	91259.55866	1	2	1	616	69501	0202
2022	2	02031211000	24480.320	24480.320	35496.46000	1	2	1	124	69501	0203
2022	2	02031955000	104688.000	104688.000	224807.40000	1	2	1	76	69501	0203
2022	2	02032110000	129756.680	129756.680	254144.81000	1	2	1	76	69501	0203
2022	2	02032110000	24000.000	24000.000	39818.85000	1	2	1	616	69501	0203
2022	2	02032211000	25839.210	25839.210	63306.06000	1	2	1	76	69501	0203
2022	2	02032211000	258508.150	258508.150	387286.28000	1	2	1	124	69501	0203
2022	2	02032211000	49592.880	49592.880	76221.57496	1	2	1	208	69501	0203

1. Building Queries By Iterating

```
1  SELECT *,
2         substring(hsn, 1, 2) AS hsn_group
3  FROM records
4  WHERE regime_id = 1
5  LIMIT 10;
6
```

getting level one value from hsn and filtering by imports

Run Query | Explain Query ▾ | 10 rows in 5 ms | JSON | CSV | X

year	month	hsn	qty	weight	value_usd	regime_id	group_id	door_id	country_id	customs_id	hsn_group
2022	2	01022110000	64.000	35000.000	133598.24550	1	1	1	233	69101	01
2022	2	01031000000	552.000	64462.000	260383.16170	1	1	1	643	69101	01
2022	2	01039219000	5214.000	623180.000	890086.38880	1	1	1	643	69101	01
2022	2	01051119000	378000.000	16261.000	125200.00000	1	1	1	792	69601	01
2022	2	01051199000	40050.000	1602.000	50979.30042	1	1	1	528	69001	01
2022	2	01061100900	3.000	40.000	3726.04467	1	1	1	710	69001	01
2022	2	01061410900	5.000	31.900	87.05260	1	1	1	643	69101	01
2022	2	01061900900	3.000	49.000	789.54969	1	1	1	643	69101	01
2022	2	01061900900	1.000	30.000	675.00000	1	1	1	840	69001	01
2022	2	01063200900	1157.000	152.000	5561.50000	1	1	1	203	69001	01

2. Building Queries By Iterating

```
SELECT *
FROM (
    SELECT *,
           substring(hsn, 1, 2) AS hsn_group
    FROM records
    WHERE regime_id = 1
) as sq1
LIMIT 10;
```

querying on the previous query result table - same results here

Run Query | Explain Query ▾ | 10 rows in 1 ms | JSON | CSV | XML

year	month	hsn	qty	weight	value_usd	regime_id	group_id	door_id	country_id	customs_id	hsn_group
2022	2	01022110000	64.000	35000.000	133598.24550	1	1	1	233	69101	01
2022	2	01031000000	552.000	64462.000	260383.16170	1	1	1	643	69101	01
2022	2	01039219000	5214.000	623180.000	890086.38880	1	1	1	643	69101	01
2022	2	01051119000	378000.000	16261.000	125200.00000	1	1	1	792	69601	01
2022	2	01051199000	40050.000	1602.000	50979.30042	1	1	1	528	69001	01
2022	2	01061100900	3.000	40.000	3726.04467	1	1	1	710	69001	01
2022	2	01061410900	5.000	31.900	87.05260	1	1	1	643	69101	01
2022	2	01061900900	3.000	49.000	789.54969	1	1	1	643	69101	01
2022	2	01061900900	1.000	30.000	675.00000	1	1	1	840	69001	01
2022	2	01063200900	1157.000	152.000	5561.50000	1	1	1	203	69001	01

3. Building Queries By Iterating

```sql
SELECT year || '-' || month AS month,
       hsn_group,
       SUM(value_usd)
FROM (
    SELECT *,
           substring(hsn, 1, 2) AS hsn_group
    FROM records
    WHERE regime_id = 1
) as sq1
GROUP BY year, month, hsn_group
LIMIT 10;
```

Run Query | Explain Query

month	hsn_group	sum
2008-1	01	198273.25876
2008-1	02	11326586.89526
2008-1	03	3306484.16808
2008-1	04	4105872.72858
2008-1	05	50142.83642
2008-1	06	567688.15230
2008-1	07	2730050.84628
2008-1	08	1888115.16744
2008-1	09	1817828.32580
2008-1	10	11607863.91400

getting a year-month value for analyzing by month and grouping the value_usd for each hsn_group and month

4. Building Queries By Iterating

```sql
SELECT year || '-' || month AS month,
       hsn_group,
       SUM(value_usd)
FROM (
    SELECT *,
           substring(hsn, 1, 2) AS hsn_group
    FROM records
    WHERE regime_id = 1
) as sq1
GROUP BY year, month, hsn_group
HAVING hsn_group = '93'
LIMIT 10;
```

month	hsn_group	sum
2008-10	93	609666.29264
2008-11	93	473894.92238
2008-12	93	1549126.57334
2009-1	93	444597.29018
2009-2	93	561898.88494
2009-3	93	479193.60382
2009-4	93	680362.00166
2009-5	93	204152.92298
2009-6	93	857232.64602
2009-7	93	81775.37584

filtering the previous grouped results by hsn_group ‘93’ or military equipment

5. Building Queries By Iterating

```sql
SELECT year || '-' || month AS month,
       hsn_group,
       SUM(value_usd)
FROM (
    SELECT *,
           substring(hsn, 1, 2) AS hsn_group
    FROM records
    WHERE regime_id = 1
) as sq1
GROUP BY year, month, hsn_group
HAVING hsn_group = '93'
ORDER BY month;
```

OOPS! The dates aren't ordering because of the single digit months need a zero padding

month	hsn_group	sum
2008-10	93	609666.29264
2008-11	93	473894.92238
2008-12	93	1549126.57334
2009-1	93	444597.29018
2009-2	93	561898.88494
2009-3	93	479193.60382
2009-4	93	680362.00166
2009-5	93	204152.92298
2009-6	93	857232.64602
2009-7	93	81775.37584

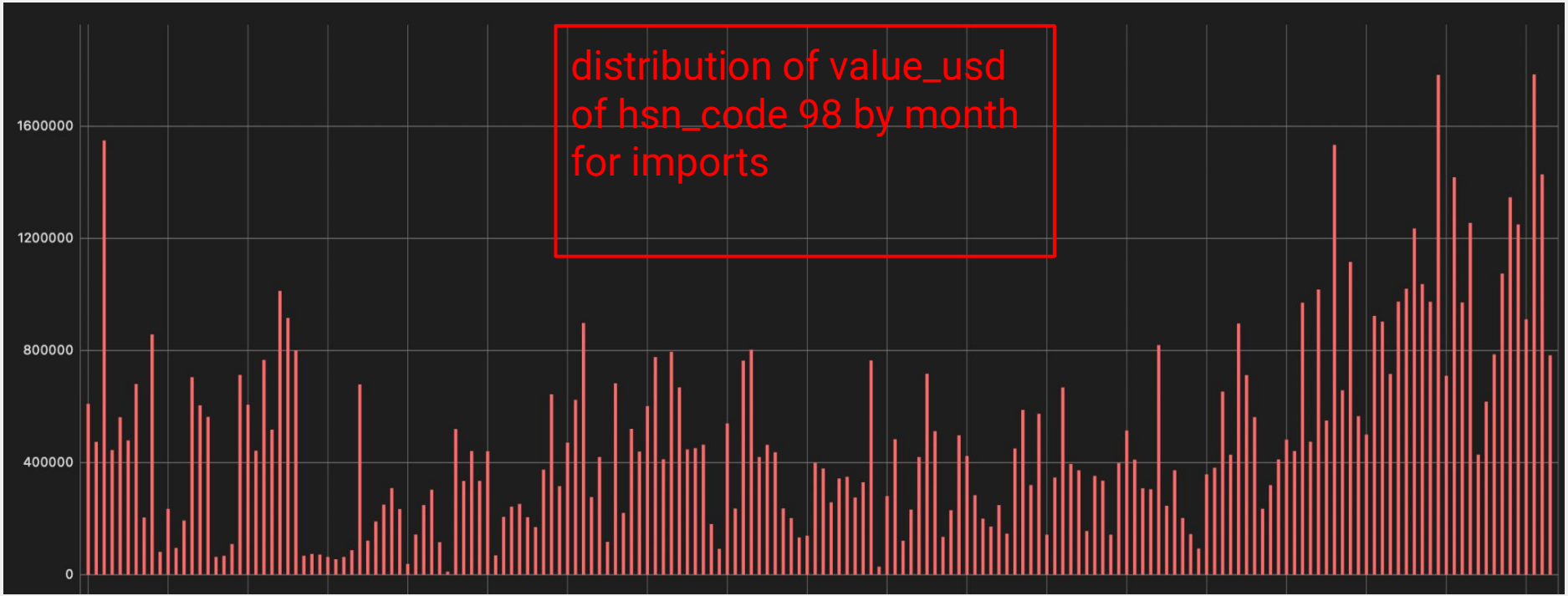
6. Building Queries By Iterating

```
SELECT year || '-' || lpad(month::text, 2, '0') AS month,
       hsn_group,
       SUM(value_usd)
FROM (
  SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
  FROM records
  WHERE regime_id = 2 -- exports only
) AS sq1
GROUP BY year, month, hsn_group
HAVING year >= 2012 -- filter by years after
   AND hsn_group = '93'
ORDER BY year, month
```

used the lpad function to pad the single digits with a ‘0’ on the left - now the order is working!

month	hsn_group	sum
2008-10	93	609666.29264
2008-11	93	473894.92238
2008-12	93	1549126.57334
2009-01	93	444597.29018
2009-02	93	561898.88494
2009-03	93	479193.60382

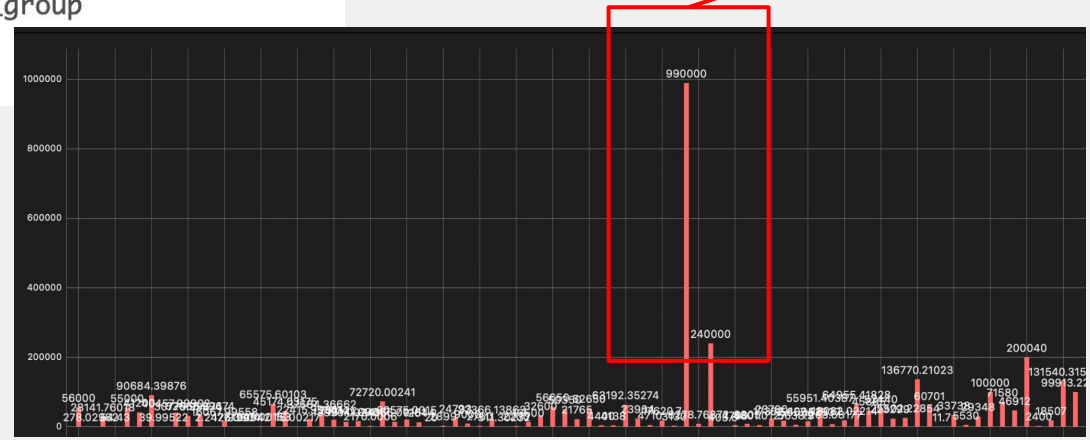
6b. Building Queries By Iterating



7. Building Queries By Iterating

```sql
SELECT year || '-' || LPAD(month::text, 2, '0')
 AS month,
        hsn_group,
        SUM(value_usd)
FROM (
    SELECT *,
           substring(hsn, 1, 2) AS hsn_group
    FROM records
    WHERE regime_id = 2
) as sq1
GROUP BY year, month, hsn_group
HAVING hsn_group = '93'
ORDER BY month;
```

changing the regime_id to 2 (exports) tells a different story. What's going on in 2018?



8. Building Queries By Iterating

```sql
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2021 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT *
FROM base_cte;
```

here we use a common table expression (CTE) to create a base query we will **reuse**. From this table, we will do our analysis.

month	hsn_group	sum
2021-04	93	52440.00000
2021-06	93	23299.00000
2021-07	93	25502.22854
2021-11	93	136770.21023
2022-01	93	60701.00000
2022-03	93	11.70000
2022-04	93	33738.00000
2022-05	93	5530.00000
2022-07	93	29348.00000

Using Z-scores to Measure Not Normal

Assuming the mean or average is “normal”, then the z-score measures how far away from normal a value is.

For example we can say if the value is 2.576 z-scores (a common threshold) or more from the average, it is not normal.

To calculate the z-score:

(value - average) / standard deviation

This translates to, by how many standard deviations does our value differ from the average?

9. Building Queries By Iterating

```
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2012 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT *,
       (sum - AVG(sum) OVER(PARTITION BY hsn_group)) / STDDEV_POP(sum) OVER(PARTITION BY hsn_group) AS hsn_zscore -- zscore
FROM base_cte
```

our base CTE

our query on the base CTE table

9b. Building Queries By Iterating

month	hsn_group	sum	hsn_zscore
2012-03	93	20114.00000	-0.23656313893795891944
2012-06	93	13990.00243	-0.28414470630746606341
2012-09	93	16141.99817	-0.26742436487831321355
2012-10	93	2170.00060	-0.37598246537794243003
2013-01	93	72720.00241	0.17216923221349961654
2013-02	93	14165.00151	-0.28278501757156381023
2013-10	93	20575.00150	-0.23298129984038170606
2014-01	93	12644.00000	-0.29460272895860263525
2014-02	93	20.00000	-0.39268730518626077025
2014-05	93	2899.00000	-0.37031836547147720160

What is a Window Function?

A window function enables us to run an aggregation - SUM, AVG, ETC - based on a subset of the table defined by a partition. The value is returned as a new column, but we still keep all the existing table columns.

This differs from a GROUP BY, which collapses the data in the rows to give us the aggregation - meaning we actually lose data in a GROUP BY.

We use window functions when we want to compare an existing value in a column with an aggregate.

What is a Window Function?

For example if we want to subtract the average of each hsn_group from the sum value and divide that by the standard deviation. The average of the hsn_group is an aggregation and the standard deviation is an aggregation.

Let’s first try a GROUP BY and see what we get.

Then let’s try the example with the window function.

What is a Window Function?

```sql
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2012 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT hsn_group,
       AVG(sum)
FROM base_cte
GROUP BY hsn_group;
```

hsn_group	avg
93	50560.918168064516

What is a Window Function?

```sql
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2012 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT *,
       (sum - AVG(sum) OVER(PARTITION BY hsn_group))/STDDEV_POP(sum) OVER(PARTITION BY hsn_group) AS hsn_zscore -- zscore
FROM base_cte;
```

What is a Window Function?

month	hsn_group	sum	hsn_zscore
2012-03	93	20114.00000	-0.23656313893795891944
2012-06	93	13990.00243	-0.28414470630746606341
2012-09	93	16141.99817	-0.26742436487831321355
2012-10	93	2170.00060	-0.37598246537794243003
2013-01	93	72720.00241	0.17216923221349961654
2013-02	93	14165.00151	-0.28278501757156381023
2013-10	93	20575.00150	-0.23298129984038170606

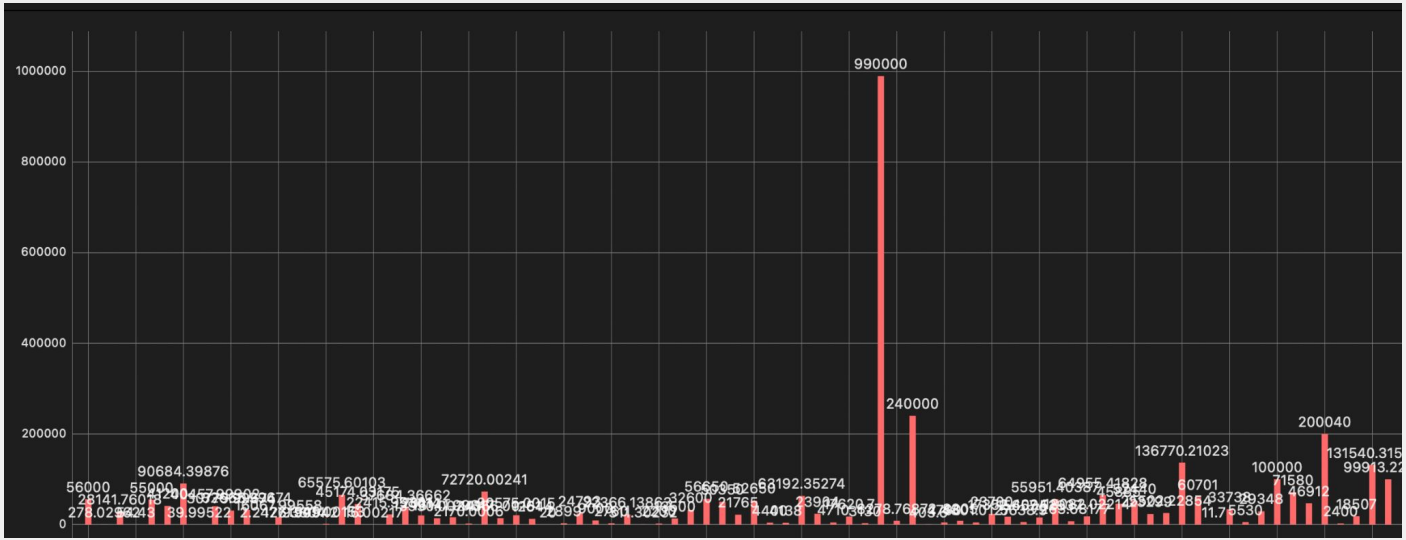
VERSUS

hsn_group	avg
93	50560.918168064516

10. Building Queries By Iterating

```sql
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2012 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT *
FROM (
    SELECT *,
           (sum - AVG(sum) OVER(PARTITION BY hsn_group)) / STDDEV_POP(sum) OVER(PARTITION BY hsn_group) AS hsn_zscore -- zscore
    FROM base_cte
)
WHERE hsn_zscore > 2.576 OR hsn_zscore < -2.576; -- zscore threshold
```

10b. Building Queries By Iterating



in this case, the z-score allows us to mathematically do what our eyeballs did with this chart - single out the outlier

month	hsn_group	sum	hsn_zscore
2018-06	93	990000.00000	7.2991511591560238

11. Building Queries By Iterating

```
WITH base_cte AS (
    -- facet query for base table (cte)
  SELECT year || '-' || lpad(month::text, 2, '0') AS month,
         hsn_group,
         SUM(value_usd)
  FROM (
    SELECT *, substring(hsn, 1, 2) AS hsn_group -- hsn granularity
    FROM records
    WHERE regime_id = 2 -- exports only
  ) AS sq1
  GROUP BY year, month, hsn_group
  HAVING year >= 2012 -- filter by years after
     AND hsn_group = '93'
  ORDER BY year, month
)
--
SELECT zscore_q.*,
       groups.name_en
FROM (
  SELECT *,
         (sum - AVG(sum) OVER(PARTITION BY hsn_group))/STDDEV_POP(sum) OVER(PARTITION BY hsn_group) AS hsn_zscore -- zscore
  FROM base_cte
) AS zscore_q
LEFT JOIN groups ON CAST(zscore_q.hsn_group AS INTEGER) = groups.id
WHERE hsn_zscore > 2.576 OR hsn_zscore < -2.576 -- zscore threshold;
```

11b. Building Queries By Iterating

month	hsn_group	sum	hsn_zscore	name_en
2018-06	93	990000.00000	7.2991511591560238	weapons and ammunition; their parts and belongings

Thank You!