Google Cloud

BigQuery ではじめる SQL

～SQL を覚えて BigQuery を使い倒そう～

2022 / 06 / 07

Google Cloud
Data Analytics Specialist
Tetsunori Nishimura

01

データ分析基盤のステークホルダー
前回のおさらい

データ分析のステーク ホルダー

ビジネスユーザー

データの在り処やそれに対する
アクセス方法が不明

できる人でも **Excel の関数やピボットテーブルが限界、SQL は無理**

IT 部門の用意してくれた BI ツールのダッシュボードでは要件が
満たされない、変更を依頼しても
待たされる

グロースハック/
デジタルマーケ

専門の分析ツールには慣れている (web なら GA / アプリなら Firebase など)

チャネル横断での分析まで手がけたい

業態によってリアルタイム データに
対する要求が強い

データ アナリスト
データ サイエンティスト

自分の得意とする言語でデータを触りたい

データの素性を重視する

クエリ速度に生産性が大きく
依存するが、IT 部門の管理する
インフラに手が出ない

分析結果を共有する方法がない

IT部門

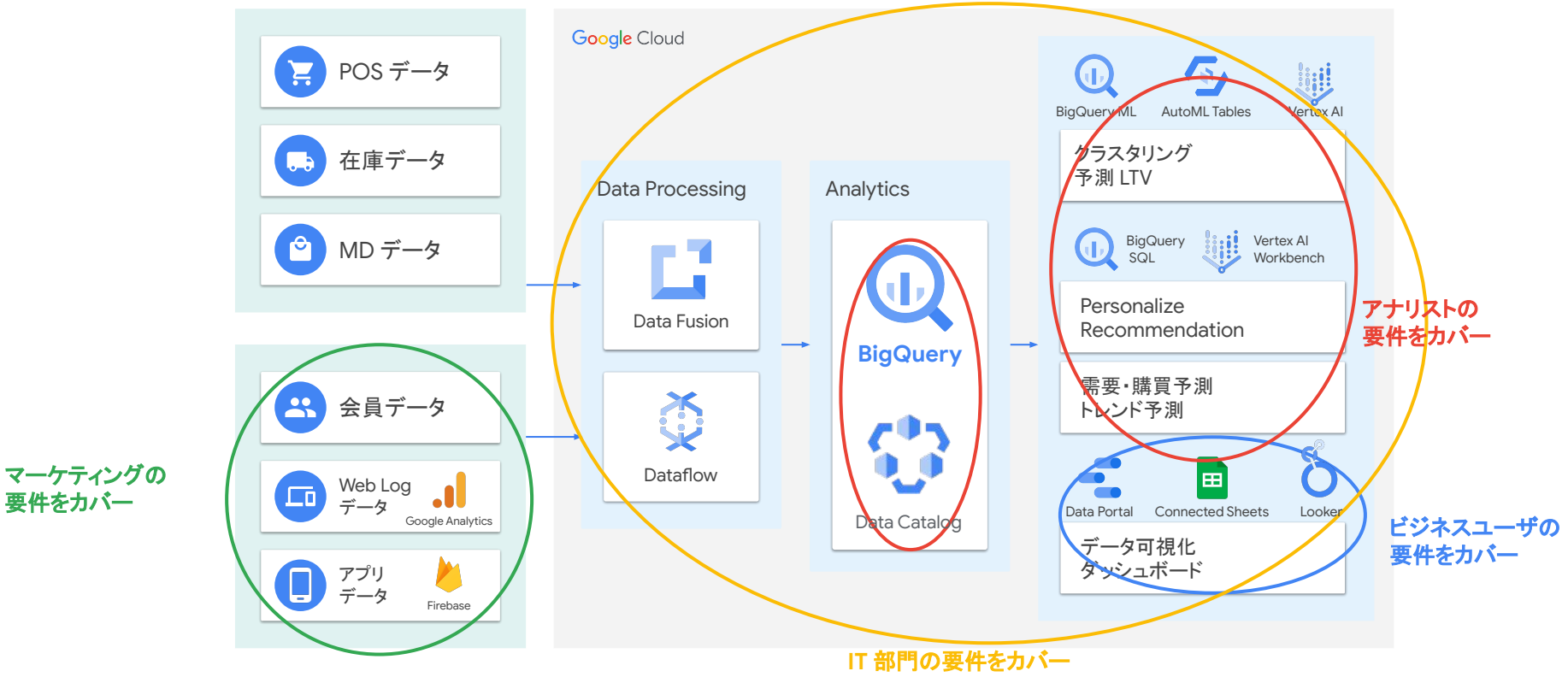
将来のデータ増加予測に基づく
インフラのサイジングが困難→サイロ化

ビジネス部門の要件が曖昧で頻繁にかわる、それにあわせた DB のチューニングが辛い

要件を全て実現するとストレージ
枯渇やバッチ突き抜けリスクがある

セキュリティ・ガバナンスの
観点から LoB 側に勝手に BI ツールや
DWH を持ってほしくない

ステークホルダーと Google Cloud 機能の関連

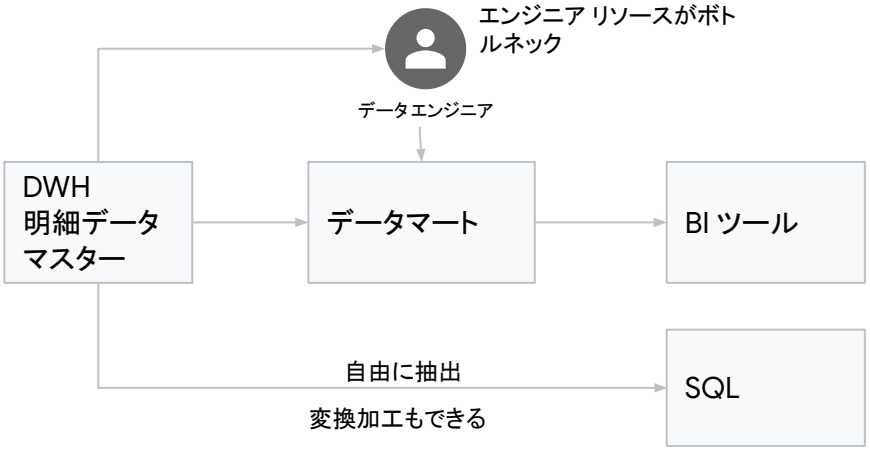


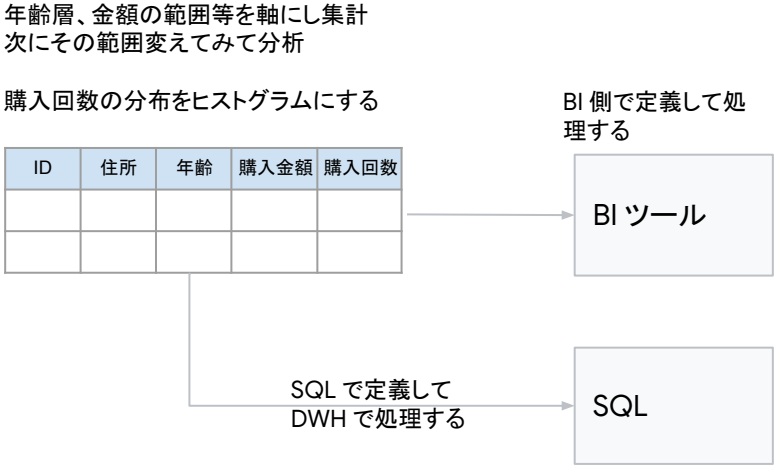
02

SQL で何ができるようになるのか？

SQL を使うメリット

- **思考にあわせて自由に分析できる**
  - データの抽出をエンジニアにお願いしなくてもいい
  - 自由にデータを加工、変換して分析に利用できる
  - データ探索がしやすい





SQL を使うメリット

- 過去から最新のデータまで明細レベルで分析ができる
  - ツール側では保持できないデータ
- 設計した分析作業を他者と共有しやすい
- どのデータ ウェアハウス製品に対しても分析できるスキルが身につく
  - 方言はあるものの SQL の基本的な考え方は同じ

SQL を使うメリット

- BigQuery を使っていると
  - **機械学習**ができる
  - PostGIS の関数を使って**位置情報の分析**ができる
  - **GA の生データを使って分析ができる**

  - 大量の**データを高速**に分析することができる
  - Web Console からすぐに SQL を始められる

SQL を使う場合の注意点

- ガバナンス
  - 同一の指標でも定義がバラバラになりやすい
  - 分析に必要なデータがどのテーブルにあるかわからないことがある

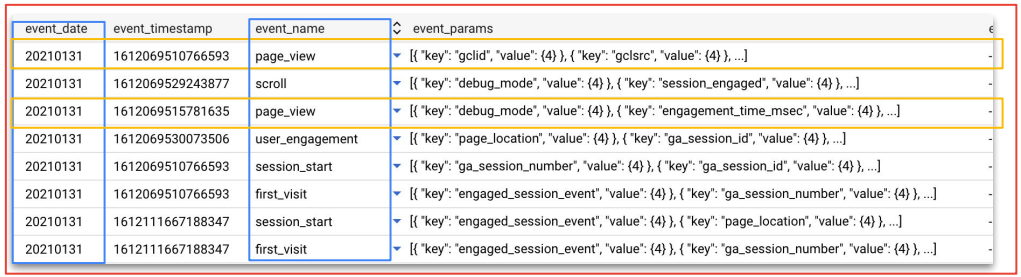
  - ツールだけでなくデータ ウェアハウス側でのアクセス制御も
    しっかりする必要がある

03

SQL を使った分析

SQL の基礎
テーブルからデータを抽出する

SELECT 取得する列を指定
FROM 検索対象のテーブルを指定
WHERE 検索条件を指定（行を指定）



event_date	event_timestamp	event_name	event_params
20210131	1612069510766593	page_view	[{ "key": "gclid", "value": {4} }, { "key": "gclsrc", "value": {4} }, ...]
20210131	1612069529243877	scroll	[{ "key": "debug_mode", "value": {4} }, { "key": "session_engaged", "value": {4} }, ...]
20210131	1612069515781635	page_view	[{ "key": "debug_mode", "value": {4} }, { "key": "engagement_time_msec", "value": {4} }, ...]
20210131	1612069530073506	user_engagement	[{ "key": "page_location", "value": {4} }, { "key": "ga_session_id", "value": {4} }, ...]
20210131	1612069510766593	session_start	[{ "key": "ga_session_number", "value": {4} }, { "key": "ga_session_id", "value": {4} }, ...]
20210131	1612069510766593	first_visit	[{ "key": "engaged_session_event", "value": {4} }, { "key": "ga_session_number", "value": {4} }, ...]
20210131	1612111667188347	session_start	[{ "key": "engaged_session_event", "value": {4} }, { "key": "page_location", "value": {4} }, ...]
20210131	1612111667188347	first_visit	[{ "key": "engaged_session_event", "value": {4} }, { "key": "ga_session_number", "value": {4} }, ...]

```
SELECT event_date, event_name

FROM
`bigquery-public-data.ga4_obfuscated_sample_ecommerce.events_20210131`

WHERE event_name = 'page_view'
```

SQL の基礎
テーブルを結合する

結合ができると例えば
購入履歴表と顧客表を結合して、どのユーザが購入したのか等を分析できる

表 1 INNER JOIN 表 2 ON（表 1 のキー = 表 2 のキー）

結合キー

o_orderkey	o_custkey	o_orderstatus	o_totalprice
358616487	8867686	F	224573.16
358625184	2992948	F	2013.07
358687397	7880134	F	52223.76
358767175	4195423	F	173464.79
21636	8665708	F	264330.92

c_custkey	c_name	c_address	c_nationkey
7455354	Customer#007455354	0RKKQGxbYdfUOtYE9UBvyGhFulE	0
12503661	Customer#012503661	s E8lQpssiyc,BGUaOtbY	0
4024604	Customer#004024604	o2s3aYYMSciQ2zcrw	0
8809228	Customer#008809228	ZFvua,hbfe	0
3444683	Customer#003444683	k6L9A7S8Rzxl,guB4DD,rmeO5LhuYeE	0
6846117	Customer#006846117	,Fy3Z7,36kUEJ04GXTrrUM18UluibG5Npwx	0

```
SELECT c_custkey, o_orderkey, o_totalprice,
c_nationkey, c_mktsegment
FROM orders INNER JOIN customer ON (o_custkey =
c_custkey)
WHERE c_custkey = 8867686
```

c_custkey	o_orderkey	o_totalprice	c_nationkey	c_mktsegment
8867686	363939173	219154.71	22	BUILDING
8867686	76866624	18903.95	22	BUILDING
8867686	32364512	63341.24	22	BUILDING

よく使う結合方式

orders

orderid	productid	total price
1	1	100
2	1	100
3	2	300
4	3	40
5	5	50

products

productid	productname
1	本
2	CD
3	ビデオ

内部結合

結合条件に指定した列の値が
両方のテーブルで一致する
レコードを連結

```
SELECT o.productid,
productname,totalprice
FROM orders o
    INNER JOIN products p
    ON (o.productid = p.productid)
```

productid	productname	total price
1	本	100
1	本	100
2	CD	300
3	ビデオ	40

外部結合

結合条件に指定した列の値が
両方のテーブルで一致する
レコードを連結し、一致しない
ものはそのままレコードを抽出

```
SELECT o.productid,
productname,totalprice
FROM orders o
    LEFT OUTER JOIN products p
    ON (o.productid = p.productid)
```

productid	productname	total price
1	本	100
1	本	100
2	CD	300
3	ビデオ	40
5	null	50

SQL の基礎
テーブルを集計する

集計ができると例えば
購入履歴表と顧客表を結合して、顧客セグメントごとの購入金額が分析できる

GROUP BY 列名 列の値ごとにグループ化する
SUM（列名）GROUP BY で指定した列の値ごとに集計する

o_orderkey	o_custkey	o_orderstatus	o_totalprice
358616487	8867686	F	224573.16
358625184	2992948	F	2013.07
358687397	7880134	F	52223.76
358767175	4195423	F	173464.79
21636	8665708	F	264330.92

```
SELECT c_mktsegment, SUM (o_totalprice) AS
totalprice
FROM orders INNER JOIN customer ON
(o_custkey = c_custkey)
GROUP BY c_mktsegment
```

c_custkey	c_name	c_address	c_mktsegment
7455354	Customer#007455354	0RKKQGxbYdfUOtYE9UBvyGhFulE	AUTOMOBILE
12503661	Customer#012503661	s E8lQpssiyc,BGUaOtbY	AUTOMOBILE
4024604	Customer#004024604	o2s3aYYMSciQ2zcrw	AUTOMOBILE
8809228	Customer#008809228	ZFvua,hbfe	AUTOMOBILE
3444683	Customer#003444683	k6L9A7S8Rzxl,guB4DD,rmeO5LhuYeE	AUTOMOBILE

c_mktsegment	totalprice
FURNITURE	4537422598685.3232
BUILDING	4532175220620.9922
MACHINERY	4532294919624.94
AUTOMOBILE	4533727886178.2393
HOUSEHOLD	4532547817309.1123

集計関数の一部

- COUNT（列名）列の値をカウントする
- COUNT（DISTINCT 列名）列の値のユニーク数をカウントする
- SUM（列名）列の値を合計する
- AVG（列名）列の値を平均する
- MAX（列名）列の値の最大値を返す
- MIN（列名）列の値の最小値を返す

集計関数

SELECT 文を組み合わせる サブクエリ

サブクエリを使うとクエリ内に複数の SELECT 文をネストしてより複雑な分析ができる
例えば、一定回数以上購入しているユーザ ID から、それに合致するユーザ情報を抽出

IN サブクエリ
5 回以上、購入しているユーザ

```
SELECT *
FROM customer
WHERE
 c_custkey IN (
   SELECT o_custkey
   FROM orders
   WHERE o_orderdate BETWEEN '1994-01-01' AND
'1994-01-31'
   GROUP BY o_custkey
   HAVING count (1) >= 5)
```

テーブル サブクエリ

```
SELECT customer.*
FROM customer INNER JOIN
    ( SELECT o_custkey
      FROM orders
      WHERE o_orderdate BETWEEN '1994-01-01' AND
'1994-01-31'
      GROUP BY o_custkey
      HAVING count (1) >= 5)
ON (c_custkey = o_custkey)
```

サブクエリ

条件式や関数等を使って値を変換する

条件式や関数等を使うことで値を変換することができる。
例えば、購入金額から顧客ランクをつける

```
SELECT
 o_custkey,
 CASE
   WHEN totalprice <= 100000 THEN 'NORMAL'
   WHEN totalprice <= 1000000 THEN 'MEDIUM'
   WHEN totalprice > 1000000 THEN 'ROYAL'
 END AS user_rank
FROM (
 SELECT o_custkey, SUM (o_totalprice) AS totalprice
 FROM orders
 WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-01-31'
 GROUP BY o_custkey)
```

o_custkey	user_rank
8332000	ROYAL
8073208	ROYAL
13364107	ROYAL
8781157	ROYAL
6387151	ROYAL
10903121	MEDIUM
201529	MEDIUM

条件式や関数等を使って値を変換する

関数を使ってランク分けをする場合

```
SELECT
 o_custkey,
 RANGE_BUCKET (totalprice, [100000.0, 1000000.0]) AS user_rank
FROM (
 SELECT o_custkey, SUM (o_totalprice) AS totalprice
 FROM orders
 WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-01-31'
 GROUP BY o_custkey);
```

o_custkey	user_rank
8781157	2
13364107	2
6387151	2
8332000	2
8073208	2
10910896	0
14346352	0

分析関数（Window 関数）を使って高度な分析をする

分析関数（WINDOW 関数）を使うことで行のグループに対しての値を計算し、
各行に対して 1 つの結果を返すことができる。
例えば、前月比、移動平均、ランク付けなどの分析ができる

monthly_sales 表

month	totalprice
1994-01-01	292071034855.67004
1994-02-01	263744269826.45023
1994-03-01	291904061988.78986
1994-04-01	282901983538.46973
1994-05-01	292129896276.40009
1994-06-01	283177418168.96039

```
SELECT
 month,
 totalprice,
 LAG (totalprice,1) OVER (ORDER BY month)
AS last_month
FROM monthly_sales
```

1994-02-01 の行にとって 1 行前の
totalprice つまり、1 月の totalprice

month	totalprice	last_month
1994-01-01	292071034855.6701	*null*
1994-02-01	263744269826.45029	292071034855.6701
1994-03-01	291904061988.78986	263744269826.45029
1994-04-01	282901983538.46985	291904061988.78986

表全体をグループ化し、month 順に
並び替えて、1 行前の値をもってくる

分析関数のコンセプト

分析関数（Window 関数）を使って高度な分析をする

3 ヶ月移動平均の例

monthly_sales 表

month	totalprice
1994-01-01	292071034855.67004
1994-02-01	263744269826.45023
1994-03-01	291904061988.78986
1994-04-01	282901983538.46973
1994-05-01	292129896276.40009
1994-06-01	283177418168.96039

```
SELECT
 month,
 totalprice,
 AVG (totalprice) OVER (ORDER BY month ROWS
BETWEEN 2 PRECEDING AND CURRENT ROW) AS
moving_average
FROM monthly_sales
ORDER BY 1
```

1994-03-01 の行は、1 月、2 月、3 月の totalprice の平均値

month	totalprice	moving_average
1994-01-01	292071034855.6701	292071034855.6701
1994-02-01	263744269826.4502	277907652341.06018
1994-03-01	291904061988.78986	282573122223.63672
1994-04-01	282901983538.46973	279516771784.56995
1994-05-01	292129896276.40009	288978647267.88654

表全体をグループ化し、month 順に並び替えて、2 行前から現在行の値の平均を計算する

分析関数のコンセプト

機械学習を使って予測分析をする

SQL で変換、集計したデータを元に BigQuery ML で機械学習をすることができる。
例えば、firebase analytics のデータから解約予測をすることも可能

train 表

列名	説明
user_pseudo_id	ユーザーID
country	国
operating_system	OS
language	言語
cnt_user_engagement	user_engagement イベントの回数
cnt_level_start_quickplay	level_start_quickplayイベントの回数
cnt_level_end_quickplay	level_end_quickplay イベントの回数
cnt_level_complete_quickplay	level_complete_quickplayイベントの回数
cnt_level_reset_quickplay	level_reset_quickplay イベントの回数
cnt_post_score	post_score イベントの回数
cnt_spend_virtual_currency	spend_virtual_currenty イベントの回数
cnt_ad_reward	ad_rewardイベントの回数
cnt_challenge_a_friend	challenge_a_friendイベントの回数
cnt_completed_5_levels	completed_5_levels イベントの回数
cnt_use_extra_steps	use_extra_stepsイベントの回数
user_first_engagement	最初の user_engagementイベントのタイムスタンプ
month	最初の user_engagementイベントの月
julianday	最初の user_engagementイベントのユリウス日
dayofweek	最初の user_engagementイベントの曜日
churned	解約したかどうか

XGBoost の分類を使い解約予測をする
モデルを作成

```
CREATE OR REPLACE MODEL bqmlga4.churn_xgb
OPTIONS (
  MODEL_TYPE="BOOSTED_TREE_CLASSIFIER",
  INPUT_LABEL_COLS=["churned"]
) AS
SELECT
  * EXCEPT (user_pseudo_id)
FROM
 bqmlga4.train
```

機械学習を使って予測分析をする

モデルの評価、モデルを使った予測も SQL で実行することができる

モデルの評価

```
SELECT
 *
FROM
 ML.EVALUATE (MODEL bqmlga4.churn_xgb)
```

precision	recall	accuracy	f1_score	log_loss	roc_auc
0.60330578512396693	0.19783197831978319	0.78459611772072635	0.29795918367346941	0.44179696790300887	0.79088511488511493

モデルで予測

```
SELECT
 user_pseudo_id,
 churned,
 predicted_churned,
 predicted_churned_probs[OFFSET (0)].prob as
probability_churned
FROM
 ML.PREDICT (MODEL bqmlga4.churn_xgb,
 (SELECT * FROM bqmlga4.train))
```

user_pseudo_id	churned	predicted_churned	probability_churned
383F4B98EC31DD2A915427D36A30354E	0	0	0.311443030834198
648F91A3D0D868FC410FC4614FF11811	0	0	0.11782803386449814
D0E006B5E441969283C94713E308A11D	0	0	0.40376183390617371
84F55D39B331B4D51467A32FECD5F0A5	0	0	0.3510470986366272
469473B2E1C330A74286BA52BA3CACF5	1	0	0.36152374744415283
F81A7FACD9640F8169D8DE2E27D217E6	0	0	0.40376183390617371
7B4198B05CE36E11C2623428A090BE16	0	0	0.311443030834198

BigQuery GIS で地理空間の分析をする

位置情報を BigQuery の GEOGRAPHY 型に格納し、BigQuery GIS を使うことで
地理空間分析ができる

緯度経度（-122.021, 37.406）から半径 50 KM 以内にあるウェザー ステーションを
検索しその距離を計算

```
SELECT
   id,
   name,
   ST_GeogPoint (longitude, latitude) AS loc,
   ST_Distance (ST_GeogPoint (longitude,
latitude),ST_GeogPoint (-122.021,37.406)) AS dist_meters
 FROM
   `bigquery-public-data.ghcn_d.ghcnd_stations`
 WHERE ST_DWithin (ST_GeogPoint (longitude,
latitude),ST_GeogPoint (-122.021,37.406), 50*1000)
```

id	name	loc	dist_meters
USC00041206	BURLINGAME	POINT(-122.35 37.5833)	35087.780902089311
USC00046144	NEWARK	POINT(-122.0325 37.5147)	12129.452834746844
US1CASM0016	HALF MOON BAY 0.7 NW	POINT(-122.4454 37.4777)	38307.237462838872
US1CASZ0025	SANTA CRUZ 2.6 ESE	POINT(-121.9912 36.961)	49552.193255534607
USW00023293	SAN JOSE	POINT(-121.9239 37.3592)	10034.222760456691
USC00046642	PALO ALTO	POINT(-122.1667 37.4333)	13220.272376055093
US1CASZ0052	LIVE OAK 1.3 WNW	POINT(-122.0026 36.9879)	46519.228750611124

04

まとめ

まとめ

- BigQuery はユーザの使い慣れたツールで分析することができる
- さらに、SQL を使えると思考にあわせて自由に分析できるようになる
- 機械学習や地理空間分析などの高度な分析も可能に

Thank you.

Google Cloud