

Tony Nguyen

Dr. Shawn Bowers

CPSC 324 01

15 April 2024

## Homework 5

### 1. Question 1

- Model information and training statistics
  - The statistics provide information on the number of iterations run, the training and evaluation loss rate, the learning rate, and the duration in seconds.
  - Among the iterations, the run time is pretty consistent without any significant changes.
  - Between the training and evaluation loss rates, the two numbers are pretty consistent as well.
  - The two loss rates and learning rates are inversely correlated. This one can easily be seen using the graph features.
- Basic syntaxes
  - Create model

```
#standardSQL
CREATE OR REPLACE MODEL `bqml_lab.sample_model`
OPTIONS(model_type='logistic_reg') AS
SELECT
  IF(totals.transactions IS NULL, 0, 1) AS label,
  IFNULL(device.operatingSystem, "") AS os,
  device.isMobile AS is_mobile,
  IFNULL(geoNetwork.country, "") AS country,
  IFNULL(totals.pageviews, 0) AS pageviews
FROM
  `bigquery-public-data.google_analytics_sample.ga_sessions_*`
WHERE
  _TABLE_SUFFIX BETWEEN '20160801' AND '20170631'
```

```
LIMIT 100000;
```

- Predict model

```
#standardSQL
SELECT
  *
FROM
  ml.EVALUATE(MODEL `bqml_lab.sample_model`, (
  SELECT
    IF(totals.transactions IS NULL, 0, 1) AS label,
    IFNULL(device.operatingSystem, "") AS os,
    device.isMobile AS is_mobile,
    IFNULL(geoNetwork.country, "") AS country,
    IFNULL(totals.pageviews, 0) AS pageviews
  FROM
    `bigquery-public-data.google_analytics_sample.ga_sessions_*`
  WHERE
    _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'));
```

- Evaluate model

```
#standardSQL
SELECT
  country,
  SUM(predicted_label) AS total_predicted_purchases
FROM
  ml.PREDICT(MODEL `bqml_lab.sample_model`, (
  SELECT
    IFNULL(device.operatingSystem, "") AS os,
    device.isMobile AS is_mobile,
    IFNULL(totals.pageviews, 0) AS pageviews,
    IFNULL(geoNetwork.country, "") AS country
  FROM
    `bigquery-public-data.google_analytics_sample.ga_sessions_*`
  WHERE
    _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'))
  GROUP BY country
  ORDER BY total_predicted_purchases DESC
  LIMIT 10;
```

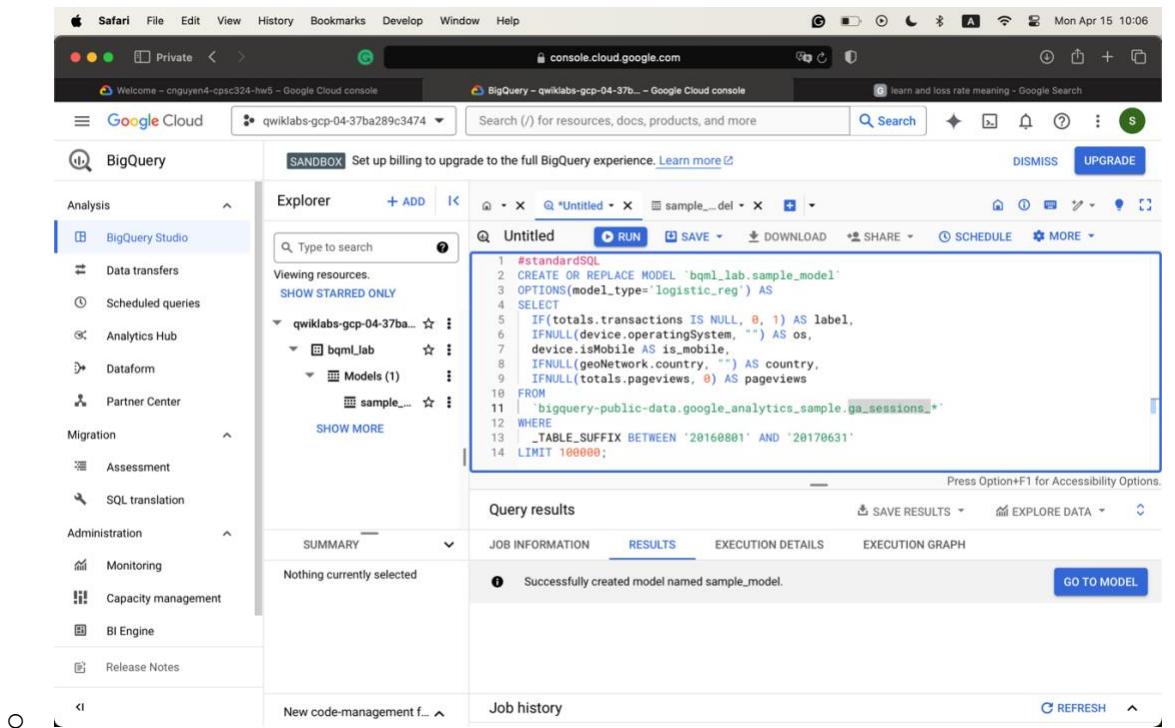
- Predicted features and features used to build the model

- Predicted features: visitor transactions

- Features used to build the model:
  - total.transactions
  - device.operatingSystem
  - device.isMobile
  - geoNetwork.country
  - totals.pageviews
- BigQuery ML returns
  - Create model
    - It does not really return anything other than saying the model has been created.
  - Evaluate model
    - It returns the statistics that are used to evaluate the model
    - In this case, we have:
      - Precision
      - Recall
      - Accuracy
      - F1\_score
      - Log\_loss
      - Roc\_auc
  - Prediction
    - The prediction step returns the predicted label for each instance.
      - The purchase per country prediction: All columns and the predicted label

- Return all schema
  - Row
  - predicted\_label
  - predicted\_label\_probs.label
  - predicted\_label\_probs.prob
  - os
  - is\_mobile
  - pageviews
  - country

- Screenshots



The screenshot shows the Google Cloud BigQuery Studio interface. On the left, there's a sidebar with navigation links for Analysis, Migration, Administration, and Release Notes. The main area has tabs for Explorer, Query results, and Job history. In the Explorer tab, under the 'Untitled' query, a SQL code editor contains the following query:

```

1 #standardSQL
2 CREATE OR REPLACE MODEL `bqml_lab.sample_model`
3   OPTIONS(model_type='logistic_reg') AS
4     SELECT
5       IF(totals.transactions IS NULL, 0, 1) AS label,
6       IFNULL(device.operatingSystem, '') AS os,
7       device.isMobile AS is_mobile,
8       IFNULL(geoNetwork.country, '') AS country,
9       IFNULL(totals.pageviews, 0) AS pageviews
10    FROM
11      `bigquery-public-data.google_analytics_sample.ga_sessions_*`
12    WHERE
13      _TABLE_SUFFIX BETWEEN '20160801' AND '20170631'
14    LIMIT 100000;

```

The Query results tab shows a summary message: "Successfully created model named sample\_model." There are tabs for JOB INFORMATION, RESULTS, EXECUTION DETAILS, and EXECUTION GRAPH. The RESULTS tab is selected.

The screenshot shows the Google Cloud BigQuery interface in a web browser. The left sidebar contains navigation links for Analysis, Migration, Administration, and Partner Center. The main area displays a query editor titled "Untitled". The query is as follows:

```

1 #standardSQL
2 SELECT
3   *
4   FROM
5     ml.EVALUATE(MODEL `bqml_lab.sample_model`, (
6       SELECT
7         IF(totals.transactions IS NULL, 0, 1) AS label,
8         IFNULL(device.operatingSystem, "") AS os,
9         device.isMobile AS is_mobile,
10        IFNULL(geoNetwork.country, "") AS country,
11        IFNULL(totals.pageviews, 0) AS pageviews
12      FROM
13        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
14      WHERE
15        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
));
  
```

Below the query editor is a "Query results" section. It includes tabs for JOB INFORMATION, RESULTS, CHART, JSON, EXECUTION DETAILS, and EXECUTION GRAPH. The RESULTS tab is selected, showing the following table:

Row	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	0.460606060...	0.070763500...	0.985383498...	0.122679580...	0.046928279...	0.977850149...

This screenshot shows the same Google Cloud BigQuery interface, but with a different query. The query is as follows:

```

1 #standardSQL
2 SELECT
3   country,
4   SUM(predicted_label) as total_predicted_purchases
5   FROM
6     ml.PREDICT(MODEL `bqml_lab.sample_model`, (
7       SELECT
8         IFNULL(device.operatingSystem, "") AS os,
9         device.isMobile AS is_mobile,
10        IFNULL(totals.pageviews, 0) AS pageviews,
11        IFNULL(geoNetwork.country, "") AS country
12      FROM
13        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
14      WHERE
15        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
16      GROUP BY country
17      ORDER BY total_predicted_purchases DESC
18  );
  
```

Below the query editor is a "Query results" section. It includes tabs for JOB INFORMATION, RESULTS, CHART, JSON, EXECUTION DETAILS, and EXECUTION GRAPH. The RESULTS tab is selected, showing the following table:

Row	country	total_predicted_purchases
1	United States	142
2	Taiwan	5
3	Canada	3
4	St. Lucia	2

The screenshot shows the Google Cloud BigQuery Studio interface. On the left, there's a sidebar with various options like Analysis, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Center, Migration, Assessment, SQL translation, Administration, Monitoring, Capacity management, BI Engine, and Release Notes. The main area has a 'Untitled' tab open with a query editor containing the following SQL code:

```

1: #standardSQL
2: SELECT
3:   -- fullVisitorId,
4:   -- SUM(predicted_label) as total_predicted_purchases
5:   *
6: FROM
7:   m1.PREDICT(MODEL `bqml_lab.sample_model`, (
8:     SELECT
9:       IFNULL(device.operatingSystem, "") AS os,
10:      device.isMobile AS is_mobile,
11:      IFNULL(totals.pageviews, 0) AS pageviews,
12:      IFNULL(geoNetwork.country, "") AS country,
13:      fullVisitorId
14:    FROM

```

Below the query editor is a 'Query results' table with the following data:

Row	label	predi...label	predi...prob	os	is_mobile	pageviews	country	fullVisitorId
1	0	1	0.00602614...	Macintosh	false	1	Netherlands	91177769...
		0	0.993973853...					
2	0	1	0.013696803...	Android	true	1	Timor-Leste	73403210...
		0	0.986303196...					
3	0	1	0.0030000532...	Android	true	1	United Kingdom	56964575...

## 2. Question 2

- Describe each task
  - Task 1
    - Run SQL codes to explore the dataset
    - The first command: What percentage of the total visitors who visited our website made a purchase?
    - The second command: The top 5 selling products.
  - Task 2
    - Identify an objective to know what we should do. This can help with feature engineering.
  - Task 3
    - Select features and create the training set

- Runs several queries to see which feature should be selected to develop the model
- Task 4
  - Create a BigQuery dataset to store models
- Task 5
  - Select BigQueryML model type
  - Build a model

```

CREATE OR REPLACE MODEL `ecommerce.classification_model`
OPTIONS
(
  model_type='logistic_reg',
  labels = ['will_buy_on_return_visit']
)
AS

#standardSQL
SELECT
  * EXCEPT(fullVisitorId)
FROM

  # features
  (SELECT
    fullVisitorId,
    IFNULL(totals.bounces, 0) AS bounces,
    IFNULL(totals.timeOnSite, 0) AS time_on_site
  FROM
    `data-to-insights.ecommerce.web_analytics` 
  WHERE
    totals.newVisits = 1
    AND date BETWEEN '20160801' AND '20170430') # train on first 9
months
  JOIN
  (SELECT
    fullvisitorid,
    IF(COUNTIF(totals.transactions > 0 AND totals.newVisits IS
    NULL) > 0, 1, 0) AS will_buy_on_return_visit
  FROM
    `data-to-insights.ecommerce.web_analytics` 
  GROUP BY fullvisitorid)
  USING (fullVisitorId)

```

;

## o Task 6

- Evaluate classification model performance

```

SELECT
    roc_auc,
    CASE
        WHEN roc_auc > .9 THEN 'good'
        WHEN roc_auc > .8 THEN 'fair'
        WHEN roc_auc > .7 THEN 'decent'
        WHEN roc_auc > .6 THEN 'not great'
        ELSE 'poor' END AS model_quality
FROM
    ML.EVALUATE(MODEL ecommerce.classification_model, (
        SELECT
            * EXCEPT(fullVisitorId)
        FROM

            # features
            (SELECT
                fullVisitorId,
                IFNULL(totals.bounces, 0) AS bounces,
                IFNULL(totals.timeOnSite, 0) AS time_on_site
            FROM
                `data-to-insights.ecommerce.web_analytics`
            WHERE
                totals.newVisits = 1
                AND date BETWEEN '20170501' AND '20170630') # eval on 2
months
            JOIN
            (SELECT
                fullvisitorid,
                IF(COUNTIF(totals.transactions > 0 AND totals.newVisits IS
NULL) > 0, 1, 0) AS will_buy_on_return_visit
            FROM
                `data-to-insights.ecommerce.web_analytics`
            GROUP BY fullvisitorid)
            USING (fullVisitorId)

        ))));

```

## o Task 7

- Improve performance with feature engineering.

- Use the same code as above, but create another model to feature engineer different features and control.
- Run the evaluation again to confirm/see the same metric.
- Task 8
  - Predict which new visitor will come back and purchase
  - Similar to the run prediction method
    - This code has been applied
- Task 9:
  - Analyze results and additional information
  - Based on the prediction result

The screenshot shows a Google Cloud BigQuery results page. The query is:

```

SELECT
    COUNT(DISTINCT user_id) AS total_newvisits
    FROM `ecommerce.events` WHERE
    date BETWEEN '20170701' AND '20170801' # test 1 month
    GROUP BY
        unique_session_id,
        will_buy_on_return_visit,
        bounces,
        time_on_site,
        pageview
    ORDER BY
        total_newvisits DESC
    LIMIT 10
  
```

The results table shows 5 rows of data:

Row	predicted_will_buy	pred_label	pred_prob	unique_session_id	will_buy_on_return_visit	latest_ecommerce	bounces	time_on_site	pageview
1	1	1	0.538374972...	5506722155080405236-1...	1	5	0	1193	
2	1	1	0.563634412...	0603906456861015666-1...	0	6	0	638	
3	1	1	0.563629305...	7355617495218387227-1...	0	6	0	519	
4	1	1	0.536973708...	1099669200376994515-1...	0	6	0	291	
5	1	1	0.507051973...	793338158599147304-15...	0	5	0	796	

- Task 10
  - Data ranges for Task 5 and Task 6. Instances used for testing
    - Task 5

- Train: 20160801 – 20170430. Used 563,076 rows for training
- Task 6
  - Evaluate: 20170501 – 20170630. Used 9,978 instances for testing
- Screenshots

The screenshot shows a Google Cloud BigQuery interface in a Safari browser window. The URL is `console.cloud.google.com`. The query titled "Untitled 2" is as follows:

```

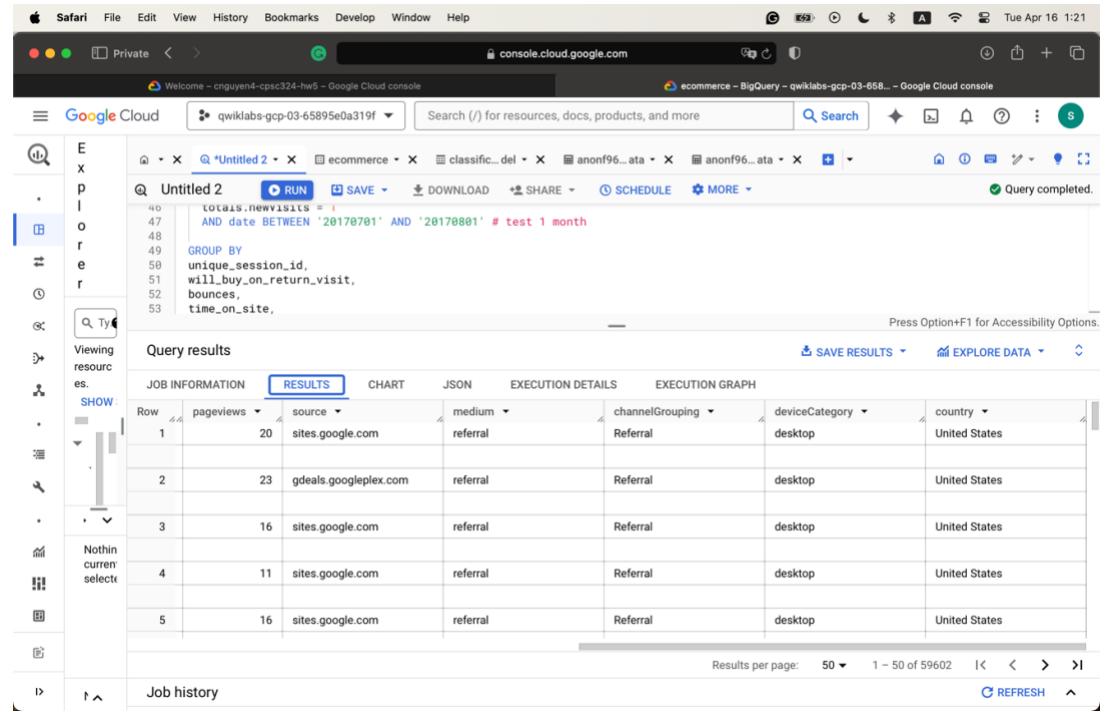
SELECT
    COUNTIF(newvisits = 1)
    AND date BETWEEN '20170701' AND '20170801' # test 1 month
GROUP BY
    unique_session_id,
    will_buy_on_return_visit,
    bounces,
    time_on_site,

```

The results table has the following columns and data:

Row	predicted_will_buy	predi...label	predi... prob	unique_session_id	will_buy_on_return	latest_ecommerce	bounces	time_on_site	pageview
1	1	1	0.538374972...	5506722155080405236-1...	1	5	0	1193	
2	1	1	0.563634412...	0603906456861015666-1...	0	6	0	638	
3	1	1	0.563629305...	7355617495218387227-1...	0	6	0	519	
4	1	1	0.536973708...	1099669200376994515-1...	0	6	0	291	
5	1	1	0.507051973...	793338158599147304-15...	0	5	0	796	

Job history: Nothin currrently selected.



Screenshot of the Google Cloud BigQuery interface. The query in the editor window is:

```

46 totals.newvisits = 1
47 AND date BETWEEN '20170701' AND '20170801' # test 1 month
48
49 GROUP BY
50 unique_session_id,
51 will_buy_on_return_visit,
52 bounces,
53 time_on_site,

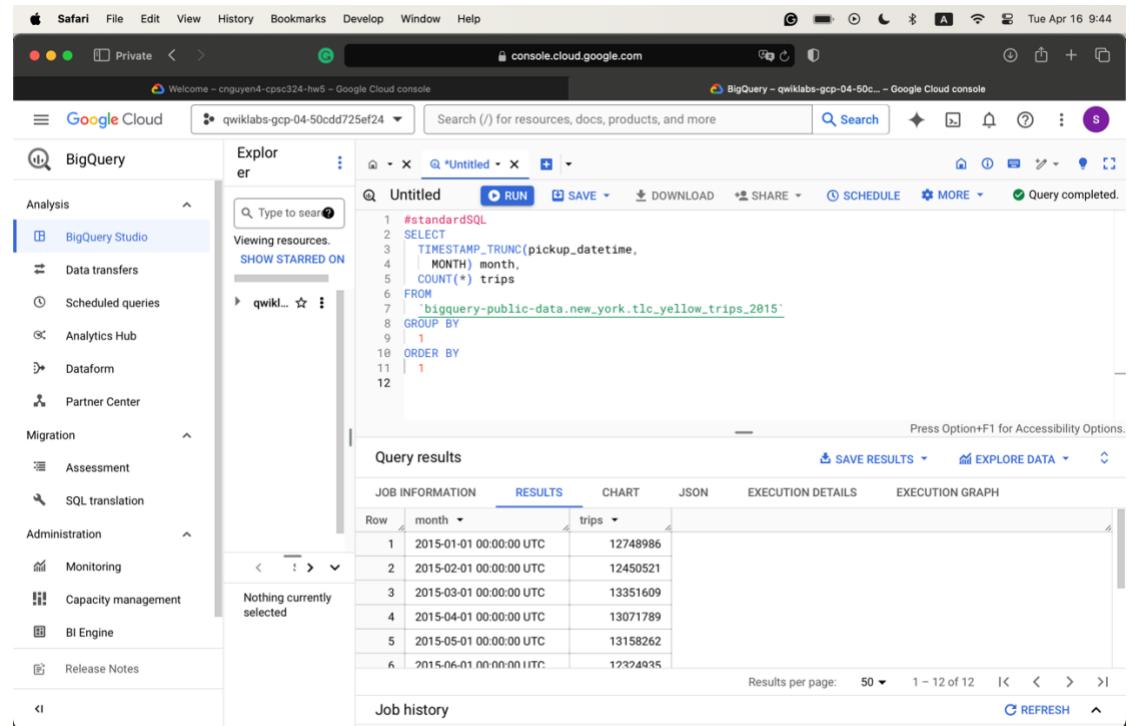
```

The results table shows the following data:

Row	pageviews	source	medium	channelGrouping	deviceCategory	country
1	20	sites.google.com	referral	Referral	desktop	United States
2	23	gdeals.googleplex.com	referral	Referral	desktop	United States
3	16	sites.google.com	referral	Referral	desktop	United States
4	11	sites.google.com	referral	Referral	desktop	United States
5	16	sites.google.com	referral	Referral	desktop	United States

### 3. Question 3

- Screenshots



Screenshot of the Google Cloud BigQuery interface. The query in the editor window is:

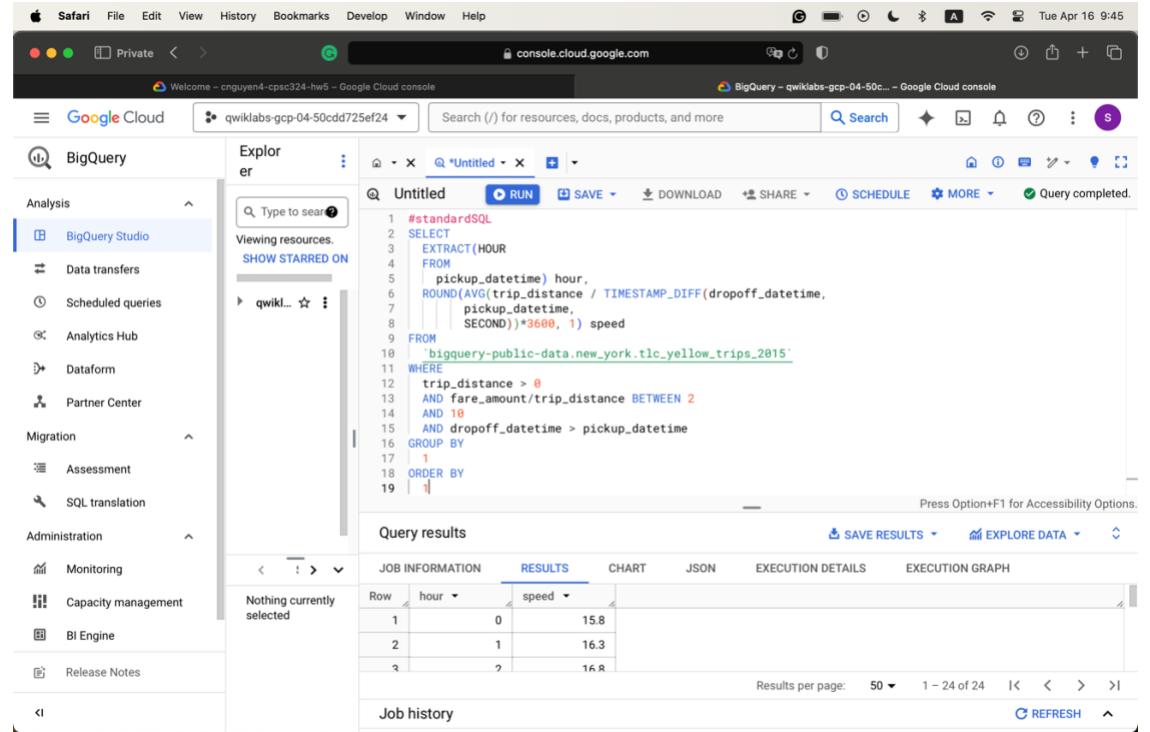
```

1 #standardSQL
2 SELECT
3   TIMESTAMP_TRUNC(pickup_datetime,
4     MONTH) month,
5   COUNT(*) trips
6 FROM
7   `bigquery-public-data.new_york.tlc_yellow_trips_2015`
8 GROUP BY
9   1
10 ORDER BY
11   1
12

```

The results table shows the following data:

Row	month	trips
1	2015-01-01 00:00:00 UTC	12748986
2	2015-02-01 00:00:00 UTC	12450521
3	2015-03-01 00:00:00 UTC	13351609
4	2015-04-01 00:00:00 UTC	13071789
5	2015-05-01 00:00:00 UTC	13158262
6	2015-06-01 00:00:00 UTC	12324935



The screenshot shows the Google Cloud BigQuery interface. On the left, the navigation sidebar includes sections for Analysis, Migration, Administration, and Partner Center. The main area displays a query editor with an untitled query. The query is as follows:

```

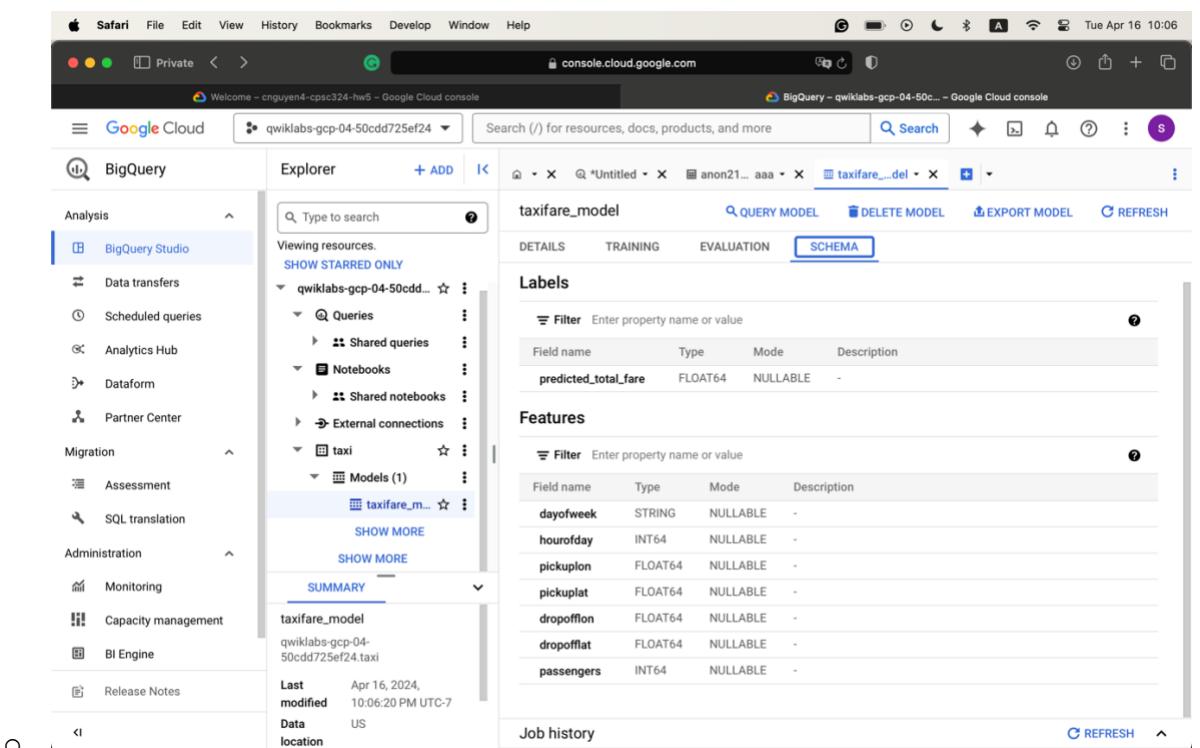
1 #standardSQL
2 SELECT
3   EXTRACT(HOUR
4   FROM
5     pickup_datetime) hour,
6   ROUND(AVG(trip_distance / TIMESTAMP_DIFF(dropoff_datetime,
7     pickup_datetime,
8     SECOND))*3600, 1) speed
9   FROM
10   `bigquery-public-data.new_york.tlc_yellow_trips_2015`
11 WHERE
12   trip_distance > 0
13   AND fare_amount/trip_distance BETWEEN 2
14   AND 18
15   AND dropoff_datetime > pickup_datetime
16 GROUP BY
17   hour
18 ORDER BY
19   hour
  
```

The results table shows three rows of data:

Row	hour	speed
1	0	15.8
2	1	16.3
3	2	16.8

Below the results, there are tabs for Job history, Save Results, and Explore Data.

- The number of rows returned in Task 3 is 1120077
- Screenshot of the schema of the training model



The screenshot shows the Google Cloud BigQuery interface focusing on a specific model named `taxifare_model`. The schema tab is selected, displaying the following fields:

Field name	Type	Mode	Description
<code>predicted_total_fare</code>	FLOAT64	NULLABLE	-

Below the schema, the Features section lists additional fields:

Field name	Type	Mode	Description
<code>dayofweek</code>	STRING	NULLABLE	-
<code>hourofday</code>	INT64	NULLABLE	-
<code>pickuplon</code>	FLOAT64	NULLABLE	-
<code>pickuplat</code>	FLOAT64	NULLABLE	-
<code>dropofflon</code>	FLOAT64	NULLABLE	-
<code>dropofflat</code>	FLOAT64	NULLABLE	-
<code>passengers</code>	INT64	NULLABLE	-

At the bottom, the Job history tab is visible.

```

CREATE OR REPLACE MODEL taxi.taxifare_model
OPTIONS
  (model_type='linear_reg', labels=['total_fare']) AS

WITH params AS (
  SELECT
    1 AS TRAIN,
    2 AS EVAL
  ),
  daynames AS
  (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'] AS
daysofweek),
  taxitrips AS (
  SELECT
    (tolls_amount + fare_amount) AS total_fare,
    daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] AS
dayofweek,
    EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
    pickup_longitude AS pickuplon,
    pickup_latitude AS pickuplat,
    dropoff_longitude AS dropofflon,
    dropoff_latitude AS dropofflat,
    passenger_count AS passengers
  FROM
    `nyc-tlc.yellow.trips`, daynames, params
  WHERE
    trip_distance > 0 AND fare_amount > 0
    AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS
STRING))), 1000) = params.TRAIN
  )
  SELECT *
  FROM taxitrips

```

- Evaluation query
  - Number of rows returned: 1

```

#standardSQL
SELECT
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL taxi.taxifare_model,
  (
    WITH params AS (

```

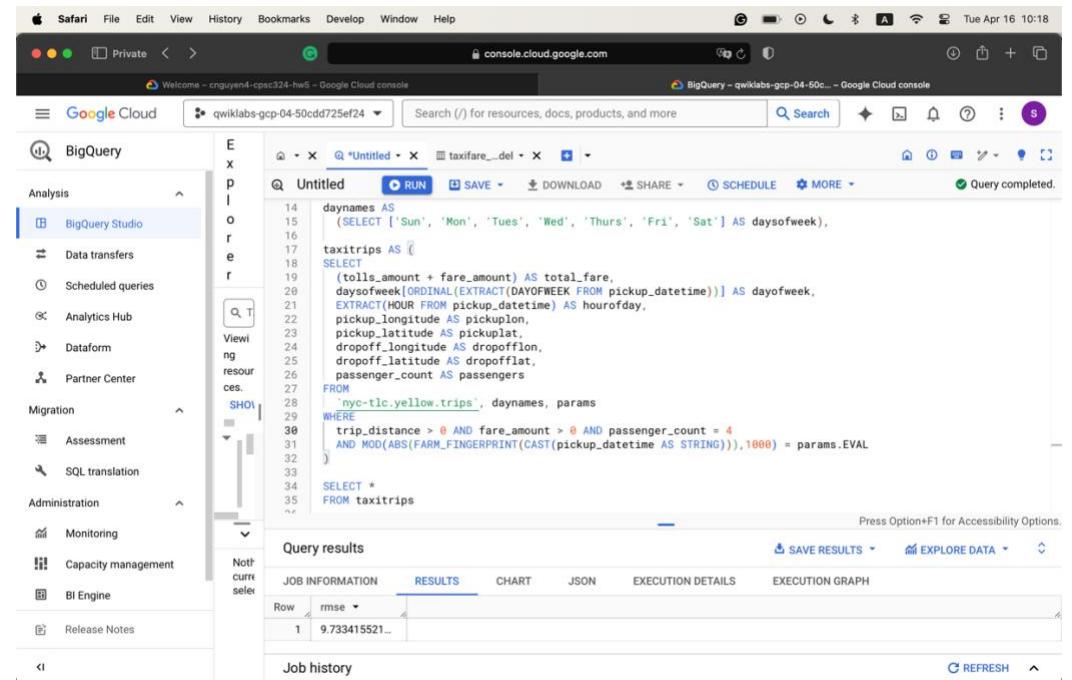
```

SELECT
  1 AS TRAIN,
  2 AS EVAL
),
daynames AS
  (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
AS daysofweek),
taxitrips AS (
SELECT
  (tolls_amount + fare_amount) AS total_fare,
  daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))]
AS dayofweek,
  EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
  pickup_longitude AS pickuplon,
  pickup_latitude AS pickuplat,
  dropoff_longitude AS dropofflon,
  dropoff_latitude AS dropofflat,
  passenger_count AS passengers
FROM
  `nyc-tlc.yellow.trips`, daynames, params
WHERE
  trip_distance > 0 AND fare_amount > 0
  AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS
STRING))),1000) = params.EVAL
)
SELECT *
FROM taxitrips
))

```

- 

- Modifying the query
  - passengers\_count = 4



The screenshot shows the Google Cloud BigQuery console interface. On the left, there's a sidebar with navigation links like Analysis, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Center, Migration, Assessment, SQL translation, and Administration. The main area has tabs for Untitled, Analysis, and BigQuery Studio. A search bar at the top right says "Search (/) for resources, docs, products, and more". Below the search bar, there's a "RUN" button and other options like SAVE, DOWNLOAD, SHARE, SCHEDULE, MORE, and a status message "Query completed". The code editor contains a query:

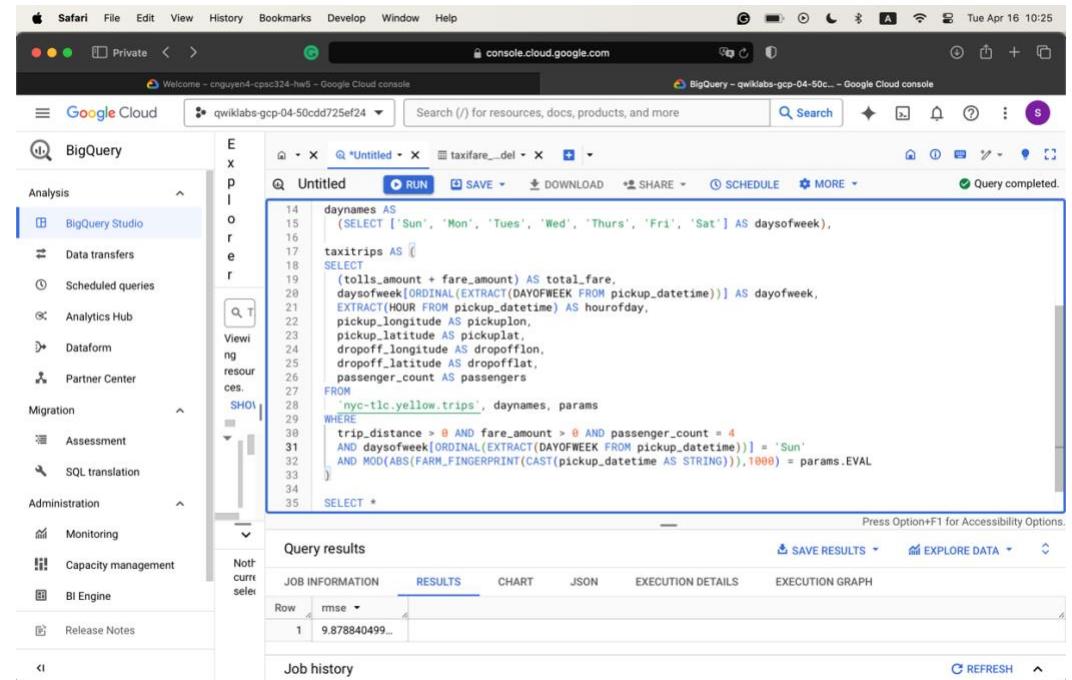
```

14 daynames AS
15   (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'] AS daysofweek,
16
17   taxitrips AS (
18     SELECT
19       (tolls_amount + fare_amount) AS total_fare,
20       daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] AS dayofweek,
21       EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
22       pickup_longitude AS pickuplon,
23       pickup_latitude AS pickuplat,
24       dropoff_longitude AS dropofflon,
25       dropoff_latitude AS dropofflat,
26       passenger_count AS passengers
27     FROM
28      `nyc-tlc.yellow.trips`, daynames, params
29     WHERE
30       trip_distance > 0 AND fare_amount > 0 AND passenger_count = 4
31       AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), 1000) = params.EVAL
32     )
33
34   SELECT *
35   FROM taxitrips

```

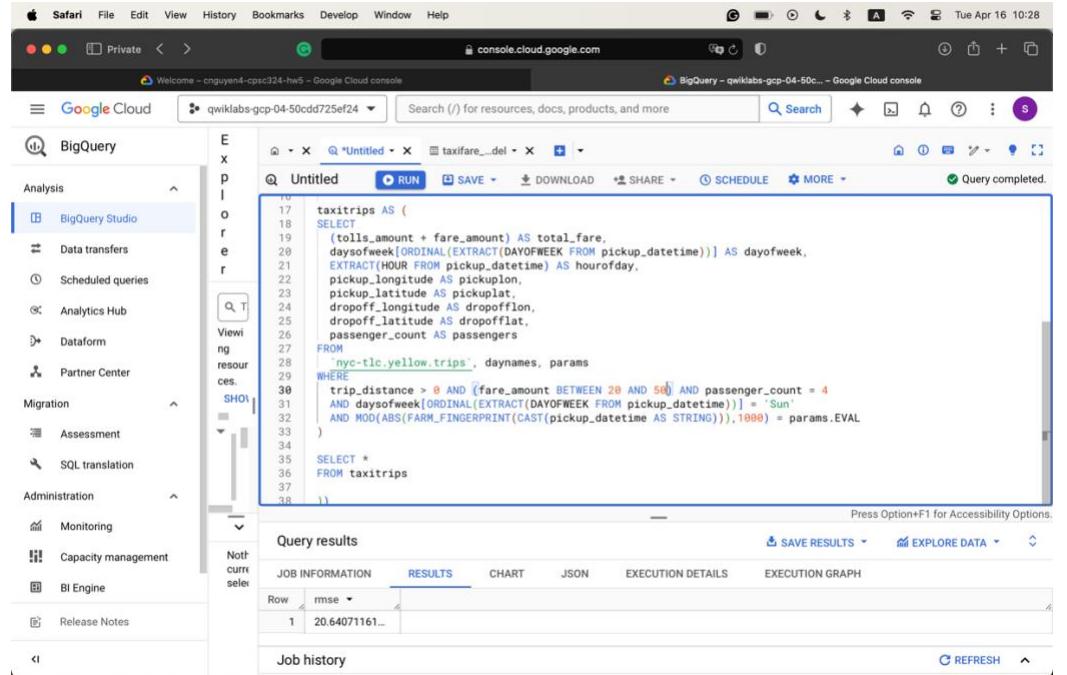
The "Query results" section shows a single row of data with an "rmse" column value of 9.733415521.

- daysofweek = 'Sun'



This screenshot is similar to the one above, showing the same BigQuery interface and code editor with the same query. However, the "Query results" section now shows a single row with an "rmse" column value of 9.878840499, indicating a change in the execution or data set.

- Fare\_amount between 20 and 50



```

17 taxitrips AS (
18   SELECT
19     (tolls_amount + fare_amount) AS total_fare,
20     daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] AS dayofweek,
21     EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
22     pickup_longitude AS pickuplon,
23     pickup_latitude AS pickuplat,
24     dropoff_longitude AS dropofflon,
25     dropoff_latitude AS dropofflat,
26     passenger_count AS passengers
27   FROM
28     `nyc-tlc.yellow.trips`, daynames, params
29   WHERE
30     trip_distance > 0 AND fare_amount BETWEEN 20 AND 50 AND passenger_count = 4
31     AND daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] = 'Sun'
32     AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), 1000) = params.EVAL
33   )
34
35   SELECT *
36   FROM taxitrips
37
38 )

```

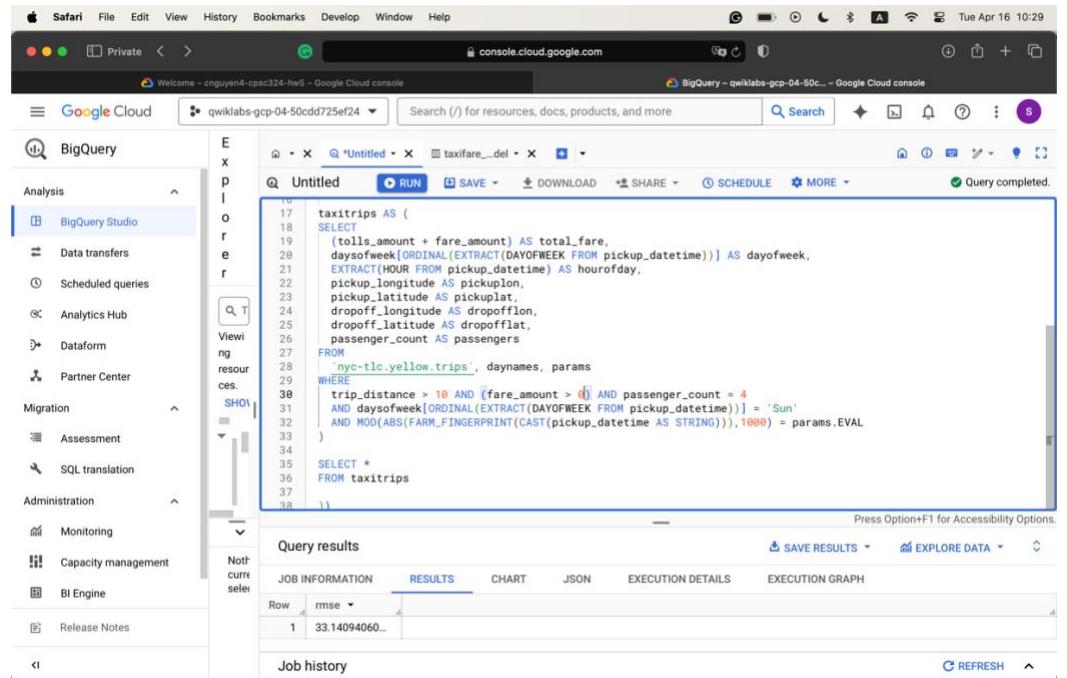
Press Option+F1 for Accessibility Options.

**Query results**

Row	rmse
1	20.64071161...

**Job history**

- Trip\_distance > 10



```

17 taxitrips AS (
18   SELECT
19     (tolls_amount + fare_amount) AS total_fare,
20     daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] AS dayofweek,
21     EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
22     pickup_longitude AS pickuplon,
23     pickup_latitude AS pickuplat,
24     dropoff_longitude AS dropofflon,
25     dropoff_latitude AS dropofflat,
26     passenger_count AS passengers
27   FROM
28     `nyc-tlc.yellow.trips`, daynames, params
29   WHERE
30     trip_distance > 10 AND fare_amount > 10 AND passenger_count = 4
31     AND daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] = 'Sun'
32     AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), 1000) = params.EVAL
33   )
34
35   SELECT *
36   FROM taxitrips
37
38 )

```

Press Option+F1 for Accessibility Options.

**Query results**

Row	rmse
1	33.14094060...

**Job history**

- Prediction process

```

#standardSQL
SELECT
*
FROM

```

```

ml.PREDICT(MODEL `taxi.taxifare_model`,
(
    WITH params AS (
        SELECT
            1 AS TRAIN,
            2 AS EVAL
    ),
    daynames AS
        (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri',
'Sat'] AS daysofweek),
    taxitrips AS (
        SELECT
            (tolls_amount + fare_amount) AS total_fare,
            daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM
pickup_datetime))] AS dayofweek,
            EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
            pickup_longitude AS pickuplon,
            pickup_latitude AS pickuplat,
            dropoff_longitude AS dropofflon,
            dropoff_latitude AS dropofflat,
            passenger_count AS passengers
        FROM
            `nyc-tlc.yellow.trips`, daynames, params
        WHERE
            trip_distance > 0 AND fare_amount > 0
            AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS
STRING))), 1000) = params.EVAL
    )
        SELECT *
        FROM taxitrips
    );
);

```

- The first model uses the raw coordinates, while the second uses the derived Euclidian distance from the coordinates. The second model also limits the fare\_amount to be between 6 and 200.
  - Using the initial parameters as in the tutorial, the second model results in a lower RMSE score (5 and 9), which suggests that it is more accurate as interpreted by the RSME.

#### 4. Question 4

- There are 33 seasons in the seasons. The data went back to 1985.
- The data used to build the model.
  - There are 2,117 instances and 27 attributes in the dataset.
  - The model only uses these attributes to train the data
    - Season
    - Win\_seed
    - Win\_seed\_ncaa
    - Lose\_seed
    - Lose\_seed\_ncaa
  - It covers data on NCAA men's games from 1985 to 2019.
- The training table generated from Task 7

Iteration	Training Data Loss	Evaluation Data Loss	Learn Rate	Duration (seconds)
3	0.5041	0.5784	1.6	2.68
2	0.5507	0.5929	0.8	2.65
1	0.6090	0.6298	0.4	2.49
0	0.6595	0.6671	0.2	2.39

○

- The modified Task 7 query

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Safari', 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Develop', 'Window', 'Help', and the date 'Wed Apr 17 12:22'. Below the bar are three tabs: 'Welcome - cngruyen4-cpsc324-hw5 - Google Cloud console', 'BigQuery - quicklabs-gcp-04-f34... - Google Cloud console', and 'BigQuery - quicklabs-gcp-04-f34... - Google Cloud console'. The main area has a sidebar titled 'Explorer' with a search bar and a 'Viewing resources' section. A query editor window titled 'Untitled' is open, showing the following SQL code:

```

category,
weight
FROM

```

The 'RESULTS' tab is selected in the query editor. The results table has columns 'Row', 'category', and 'weight'. The data rows are:

Row	category	weight
1	Loyola Chicago	1.014132345...
2	Tulane	0.685367979...
3	UConn	0.611630255...
4	North Carolina	0.581984224...
5	Kentucky	0.547159846...
6	Northwestern St.	0.530241361...
7	Duke	0.516683259...
8	Kansas	0.430801105...
9	Florida	0.418577485...
10	Mt. St. Mary's	0.406413579...
11	Syracuse	0.383629668...

Below the results table is a 'Job history' section with a refresh button.

- The results are the seed of each school (there are 286 of them). The lower the weight is, the more likely the team will win.
- Gonzaga's weight is 0.039
- The query result for Task 10

○

Row	predicted_label	predicted_label_p_label	predi... prob	season	round	days_from_epoch	game_date	day
1	win	win	0.631865055...	2018	16	17612	2018-03-22	Thursday
		loss	0.368134944...					
2	loss	win	0.466684445...	2018	32	17607	2018-03-17	Saturday
		loss	0.533315554...					

○

Row	day	label	seed	market	name	alias
1	Thursday	loss	04	Gonzaga	Bulldogs	GONZ
2	Saturday	win	04	Gonzaga	Bulldogs	GONZ

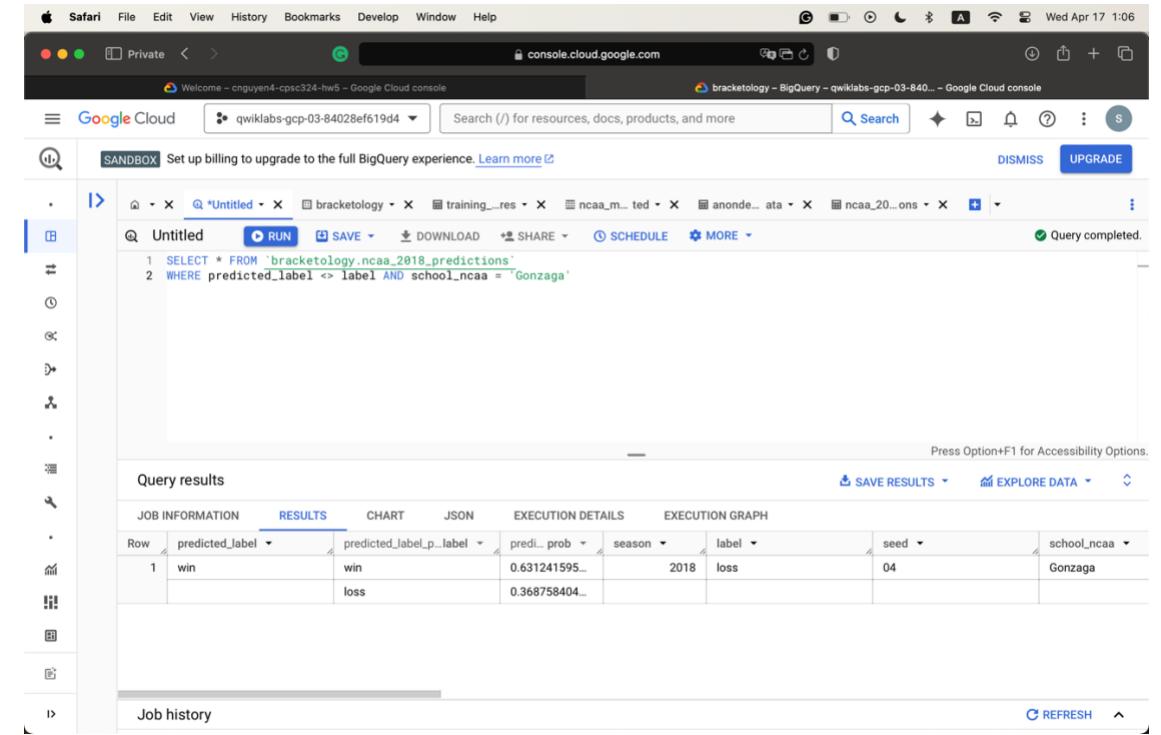
○

Row	name	alias	school_ncaa	points	opponent_seed	opponent_market
1	Bulldogs	GONZ	Gonzaga	60	09	Florida State
2	Bulldogs	GONZ	Gonzaga	90	05	Ohio State

○

Row	opponent_seed	opponent_market	opponent_name	opponent_alias	opponent_school_ncaa	opponent_points
1	60	09	Florida State	Seminole	FSU	75
2	90	05	Ohio State	Buckeyes	OSU	84

- The number of rows resulting from the Task 12 query is 664.
- The number of rows used in the Task 15 training model is 438.
- Modify Task 19 query to include Gonzaga



The screenshot shows a Google Cloud BigQuery interface in a Safari browser window. The URL is console.cloud.google.com. The query in the editor is:

```
1 SELECT * FROM `bracketology.ncaa_2018_predictions`
2 WHERE predicted_label <> label AND school_ncaa = 'Gonzaga'
```

The results table shows one row for Gonzaga:

Row	predicted_label	predicted_label_p_label	predi... prob	season	label	seed	school_ncaa
1	win	win	0.631241595...	2018	loss	04	Gonzaga
		loss	0.368758404...				

- Task 21 query
  - The query may seem a bit intimidating at first, but it is, in fact, quite easy to understand as it is straightforward. Code comments are used liberally to denote their functions.
  - The advantage is that everything is laid out. We can control what is returned and so on.
  - The disadvantage: it makes the function complicated and hard to maintain. I can imagine users can easily fall through the logic why trying to understand it.
- 2019 prediction results
  - The model returns some pretty accurate results

- The game that Gonzaga lost to Texas Tech was correctly predicted, though the confidence rate is not high (51% chance of losing).
- For Gonzaga v. Florida State, Gonzaga v. Baylor, and Gonzaga v. Fairleigh Dickinson, the model correctly predicted that Gonzaga won.

## 5. Question 5

- Result from the first query of Task 4

The screenshot shows a Google Cloud BigQuery results page. The table displays the following data:

Row	type	isFraud	cnt
1	CASH_IN	0	1399284
2	CASH_OUT	0	2233384
3	CASH_OUT	1	4116
4	DEBIT	0	41432
5	PAYOUT	0	2151495
6	TRANSFER	0	528812
7	TRANSFER	1	4097

Below the table, a terminal window shows the command used to load the data:

```
student 01 65210b58995d@cloudshell:~ (qwiklabs-gcp-02-759816db3398)$ bq load --autodetect --source format=CSV --max bad records=100000 finance.fraud data gs://$PROJECT_ID/SDM/FILE/Google-cloud-sdk/platform/bq/third_party/requests/ init .py:103: RequestsDependencyWarning: urllib3 (2.0.7) or chardet (None)/charset normalizer (2.0.7) doesn't match a supported version!
  warnings.warn("urllib3 ({}) or chardet ({})/charset normalizer ({}) doesn't match a supported "
Waiting on bqjob_r ea248bc0e07b624_0000018eed76c304_1 ... (15s) Current status: DONE
student 01 65210b58995d@cloudshell:~ (qwiklabs-gcp-02-759816db3398)$
```

- Pause and reflect on Task 5
  - If there is no labeled fraud event in the data, I will probably use unsupervised learning to try to detect any abnormalities in the data. For skimming through, I can see that for CASH transactions, the amount is pretty big. I will also use association rules to try to find something that can be linked together.
- Metrics available for Task 6

Safari File Edit View History Bookmarks Develop Window Help

console.cloud.google.com

Welcome – cnguyen4-cpsc324-hw5 – Google Cloud console

BigQuery – qwiklabs-gcp-02-759816db3398 – Google Cloud console

Google Cloud Search (/) for resources, docs, products, and more

model\_unsupervised

EVALUATION

Metrics

Davies–Bouldin index	1.5503
Mean squared distance	4.5252

Numeric features

This table shows the centroid value for each feature. Use the select menu to view more numeric features.

Selected Features: amount, amountError, destzeroFlag, newbalanceDest, newbalanceOrig, old...

Centroid Id	Count	amount	amountError	destzeroFlag	newbalanceDest
1	1,492	7,978,375.4509	0.3606	0.2269	12,474,0
2	99,581	255,383.8624	0.0028	1.0000	1,831,40
3	62	9,542,731.1962	0.9718	0.0000	23,983,3
4	7,720	751,269.7974	0.0000	0.6606	15,951,5
5	118,677	282,443.0833	0.0280	0.0000	1,073,72

Job history

REFRESH

Safari File Edit View History Bookmarks Develop Window Help

console.cloud.google.com

Welcome – cnguyen4-cpsc324-hw5 – Google Cloud console

BigQuery – qwiklabs-gcp-02-759816db3398 – Google Cloud console

Google Cloud Search (/) for resources, docs, products, and more

model\_unsupervised

EVALUATION

4	7,720	751,269.7974	0.0000	0.6606	15,951,5
5	118,677	282,443.0833	0.0280	0.0000	1,073,72

Categorical features

Each chart below shows the category value distribution for a particular feature. Use the select menu to view more categorical features.

Selected Features: type

Centroid	TRANSFER	CASH_OUT
1	~1.491%	~98.5%
2	~99.580%	~0.42%
3	~61%	~39%
4	~7.719%	~92.28%
5	~118.676%	~81.32%

Job history

REFRESH

- Pause and reflect on Task 6

- The cluster that I think is the most interesting one is cluster 3. It is relatively small but has a very high error rate (near 100%).
- I think this is due to the fact that the instances in that cluster are really similar to each other, yet they are relatively different from the other ones.
- Pause and Reflect on Task 7
  - oldBalanceOrig and type are the two most important ones.
  - This is consistent with my response to Pause and Reflect of Task 5.
- Pause and Reflect on Task 9
  - There is a significant difference between the two models:

Row	model_name	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	Initial_reg	0.971905...	0.387878...	0.981919...	0.554472...	0.068082...	0.930771...
2	improved_reg	0.968573...	0.859242...	0.995108...	0.910638...	0.013089...	0.998735...

- The improved model, which is a decision tree, yields a much better result than the initial classification one, as seen in the f1-score.
- I noticed that the fields used are similar, and the only thing that is different is their model type.
- I was actually surprised by the difference in the results.

## 6. Question 6

- The evaluation table

The screenshot shows the Google Cloud Platform interface with three tabs open in the browser: 'Welcome - cnguyen4-cpsc324-hw5 - Google Cloud console', 'BigQuery - qwiklabs-gcp-01-5d6... - Google Cloud console', and 'BigQuery - qwiklabs-gcp-01-5d6... - Google Cloud console'. The main view is on the 'EVALUATION' tab for the 'bike\_model' dataset. The table contains the following data:

Time Series ID	Non Seasonal P	Non Seasonal D	Non Seasonal Q	Has Drift	Log Likelihood	AIC	Variance
293	0	1	5	False	-3,574.667	7,161.335	1,048.85
435	0	1	5	False	-3,408.346	6,828.692	667.92
497	0	1	5	False	-3,546.302	7,104.604	972.30
519	0	1	5	False	-3,772.511	7,557.022	1,810.79
521	2	1	2	False	-3,838.727	7,689.455	2,190.27

- Result and query from Task 5

The screenshot shows the Google Cloud Platform interface with three tabs open in the browser: 'Welcome - cnguyen4-cpsc324-hw5 - Google Cloud console', 'BigQuery - qwiklabs-gcp-01-5d6... - Google Cloud console', and 'BigQuery - qwiklabs-gcp-01-5d6... - Google Cloud console'. The main view is on the 'RESULTS' tab for an 'Untitled query'. The query code is:

```

DECLARE HORIZON STRING DEFAULT "2"; #number of values to forecast
DECLARE CONFIDENCE_LEVEL STRING DEFAULT "0.95";
EXECUTE IMMEDIATE format("""
SELECT
*
FROM
ML.FORECAST(MODEL bamlforecast.bike_model,

```

The results table shows 10 rows of forecast data:

Row	start_station_id	forecast_timestamp	forecast_value	standard_error	confidence_level	prediction_interval_min	prediction_interval_max	confidence_interval_min	confidence_interval_max
1	293	2016-01-01 00:00:00 UTC	126.4074521...	32.39120113...	0.95	63.03542462...	189.7794797...	63.03542462...	189.7794797...
2	293	2016-01-02 00:00:00 UTC	65.11916370...	32.93565671...	0.95	0.681931613...	129.5563957...	0.681931613...	129.5563957...
3	435	2016-01-01 00:00:00 UTC	119.8664732...	25.84422478...	0.95	69.30333053...	170.4296159...	69.30333053...	170.4296159...
4	435	2016-01-02 00:00:00 UTC	130.2501376...	26.44030342...	0.95	78.52079206...	181.9794832...	78.52079206...	181.9794832...
5	497	2016-01-01 00:00:00 UTC	134.4615819...	31.18192838...	0.95	73.45544582...	195.4677180...	73.45544582...	195.4677180...
6	497	2016-01-02 00:00:00 UTC	128.0010055...	31.90127576...	0.95	65.58749647...	190.4145146...	65.58749647...	190.4145146...
7	519	2016-01-01 00:00:00 UTC	153.1212924...	42.55340873...	0.95	69.86733162...	236.3752533...	69.86733162...	236.3752533...
8	519	2016-01-02 00:00:00 UTC	-101.7019894...	43.17558905...	0.95	-186.1732199...	-17.23075886...	-186.1732199...	-17.23075886...
9	521	2016-01-01 00:00:00 UTC	38.05031312...	46.80033711...	0.95	-53.51258512...	129.6132113...	-53.51258512...	129.6132113...
10	521	2016-01-02 00:00:00 UTC	-10.61408988...	49.23471990...	0.95	-106.9397560...	85.71157627...	-106.9397560...	85.71157627...

```

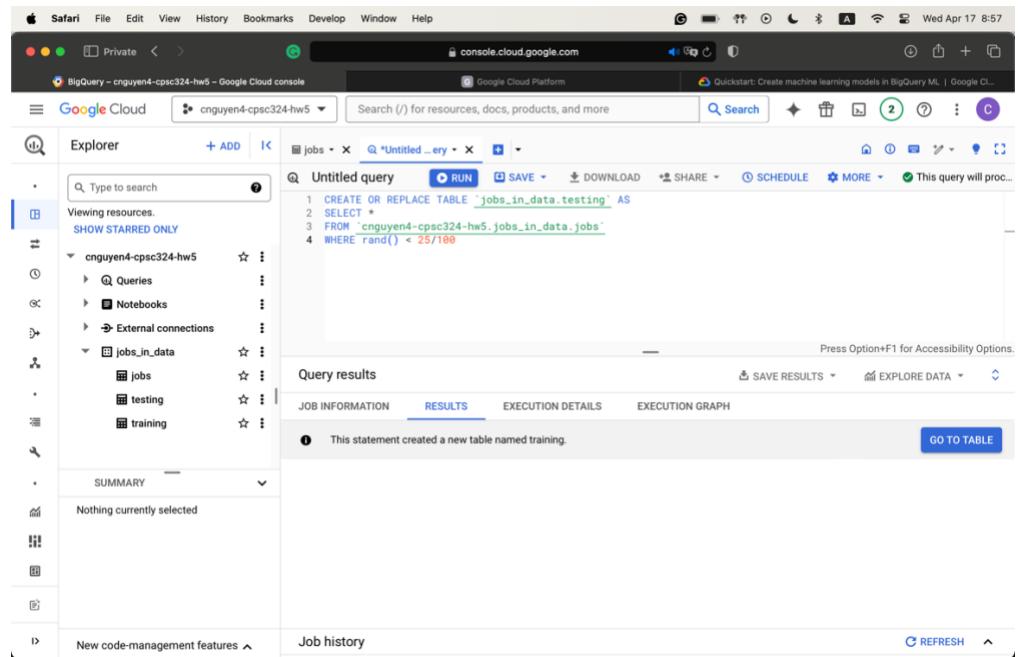
DECLARE HORIZON STRING DEFAULT "2"; #number of values to forecast
DECLARE CONFIDENCE_LEVEL STRING DEFAULT "0.95";

```

```
EXECUTE IMMEDIATE format("""
SELECT
  *
FROM
  ML.FORECAST(MODEL bqmlforecast.bike_model,
    STRUCT(%s AS horizon,
      %s AS confidence_level)
  )
""", HORIZON, CONFIDENCE_LEVEL)
```

## 7. Question 7

- The dataset I am using is the jobs\_in\_data.csv dataset from the previous homework.
  - There are 9,355 rows and 12 attributes.
- The label here is the yearly salary based on the position and other factor. This model should be able to suggest a recommended yearly salary so that it reflects the average salary in the market.
- I plan to split them up into training and testing sets using the handout method. I will keep 25% of the dataset as the testing set, and the remaining one will be the training set.
  - To create the testing set



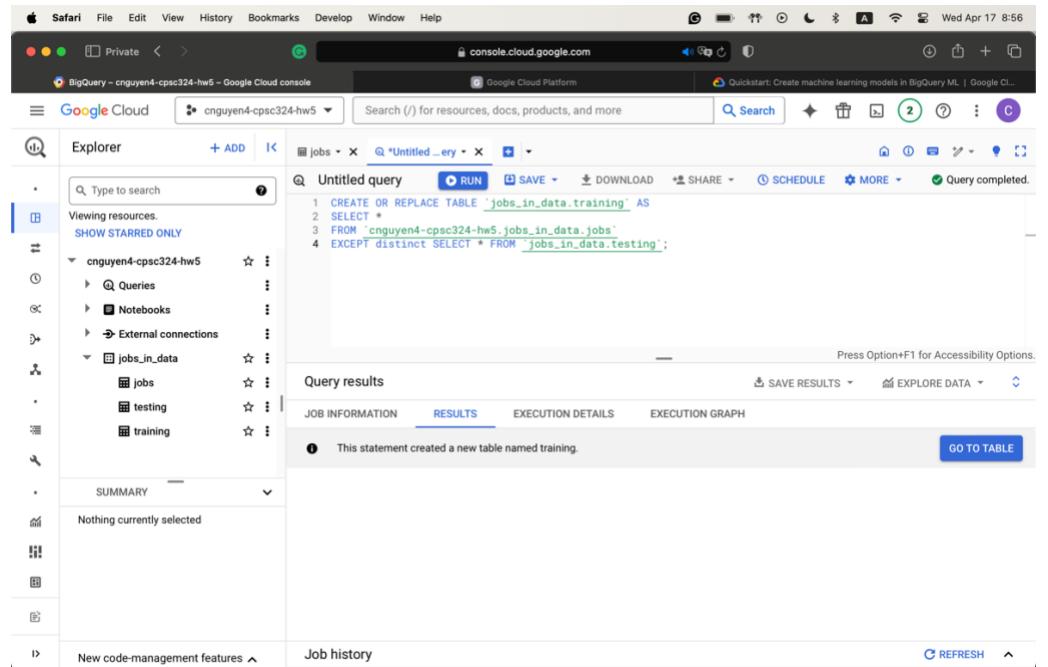
The screenshot shows the Google Cloud BigQuery interface in a web browser. The left sidebar displays the project structure under 'cnguyen4-cpsc324-hw5'. The main area shows an 'Untitled query' with the following SQL code:

```

1 CREATE OR REPLACE TABLE `jobs_in_data.testing` AS
2 SELECT *
3 FROM `cnguyen4-cpsc324-hw5.jobs_in_data.jobs`
4 WHERE rand() < 25/100
    
```

The 'RESULTS' tab is selected, showing the message: "This statement created a new table named training." A 'GO TO TABLE' button is visible.

- To create the training set



The screenshot shows the Google Cloud BigQuery interface in a web browser. The left sidebar displays the project structure under 'cnguyen4-cpsc324-hw5'. The main area shows an 'Untitled query' with the following SQL code:

```

1 CREATE OR REPLACE TABLE `jobs_in_data.training` AS
2 SELECT *
3 FROM `cnguyen4-cpsc324-hw5.jobs_in_data.jobs`
4 EXCEPT DISTINCT SELECT * FROM `jobs_in_data.testing`;
    
```

The 'RESULTS' tab is selected, showing the message: "This statement created a new table named training." A 'GO TO TABLE' button is visible.

- To train the model

The screenshot shows the Google Cloud BigQuery interface in a web browser. The left sidebar is titled 'Explorer' and lists 'External connections', 'jobs\_in\_data' (which contains 'Models (1)' and 'salary\_model'), and other datasets like 'testing' and 'training'. The main area is titled 'Untitled query' and contains the following SQL code:

```

1 CREATE OR REPLACE MODEL jobs_in_data.salary_model
2   OPTIONS (model_type='LINEAR_REG', INPUT_LABEL_COLS=['salary_in_usd']) AS
3   SELECT *
4   FROM `jobs_in_data.training`
5

```

The status bar at the bottom right indicates 'Press Option+F1 for Accessibility Options.'

- To evaluate the model

The screenshot shows the Google Cloud BigQuery interface in a web browser. The left sidebar is titled 'Explorer' and lists 'External connections', 'jobs\_in\_data' (which contains 'Models (1)' and 'salary\_model'), and other datasets like 'testing' and 'training'. The main area is titled 'Untitled query' and contains the following SQL code:

```

1 CREATE OR REPLACE TABLE jobs_in_data.model_eval AS
2   SELECT SQRT(mean_squared_error) AS rmse
3   FROM ML_EVALUATE(MODEL `cnguyen4-cpsc324-hw5.jobs_in_data.salary_model`,(
4     SELECT *
5     FROM `jobs_in_data.training`
6   ))

```

The status bar at the bottom right indicates 'Press Option+F1 for Accessibility Options.' A message in the bottom right corner says 'This statement created a new table named model\_eval.' and has a 'GO TO TABLE' button.

The screenshot shows the Google Cloud BigQuery interface. The left sidebar is titled "Explorer" and lists resources under "Viewing resources. SHOW STARRED ONLY". It includes sections for "External connections", "jobs\_in\_data" (with a "Models (1)" sub-section containing "salary\_model"), and other datasets like "jobs", "model\_eval", "testing", and "training". Below this is a "SUMMARY" section for "model\_eval" which shows it was last modified on April 17, 2024, at 9:19:18 PM UTC-7, with data located in the US. The main content area is titled "model\_eval" and shows a "PREVIEW" tab selected. A table has one row with the value "rmse" in the first column and "8703.57..." in the second column. There are tabs for "SCHEMA", "DETAILS", "PREVIEW", "LINEAGE", "DATA PROFILE", and "DATA QUALITY". At the top right, there are buttons for "SHARE", "COPY", "SNAPSHOT", "DELETE", "EXPORT", and "REFRESH". A search bar at the top is set to "cnguyen4-cpsc324-hw5".

- It seems like the performance of this model is not that good.
- Prediction

The screenshot shows the Google Cloud BigQuery interface. The left sidebar is identical to the previous screenshot, showing the "Explorer" with "model\_eval" selected. The main content area is titled "Untitled query" and contains the following SQL code:

```

1 CREATE OR REPLACE TABLE `jobs_in_data.prediction` AS
2 SELECT job_category, AVG(salary_in_usd) as avg_salary
3 FROM ML.PREDICT(MODEL `jobs_in_data.salary_model`, (
4   SELECT *
5     FROM `jobs_in_data.testing`
6   ))
7 GROUP BY job_category
  
```

Below the code, a "Query results" section is visible with tabs for "JOB INFORMATION", "RESULTS" (which is selected), "EXECUTION DETAILS", and "EXECUTION GRAPH". A message states "This statement created a new table named prediction." At the bottom right, there is a "GO TO TABLE" button. The top navigation bar shows "console.cloud.google.com", "Google Cloud Platform", and "Quickstart: Create machine learning models in BigQuery ML | Google Cl...".

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar lists datasets like 'jobs\_in\_data' and 'Models (1)'. The main pane displays the 'prediction' dataset with a preview table. The table has two columns: 'job\_category' and 'avg\_salary'. The data shows various job categories and their average salaries.

Row	job_category	avg_salary
1	Data Engineering	144618.6...
2	Leadership and Managem...	141711.8...
3	BI and Visualization	133859.6...
4	Machine Learning and AI	174432.2...
5	Data Science and Research	163228.9...
6	Data Quality and Operations	98587.68...
7	Data Architecture and Mo...	148055.1...
8	Data Analysis	107456.7...
9	Data Management and Str...	92538.235...

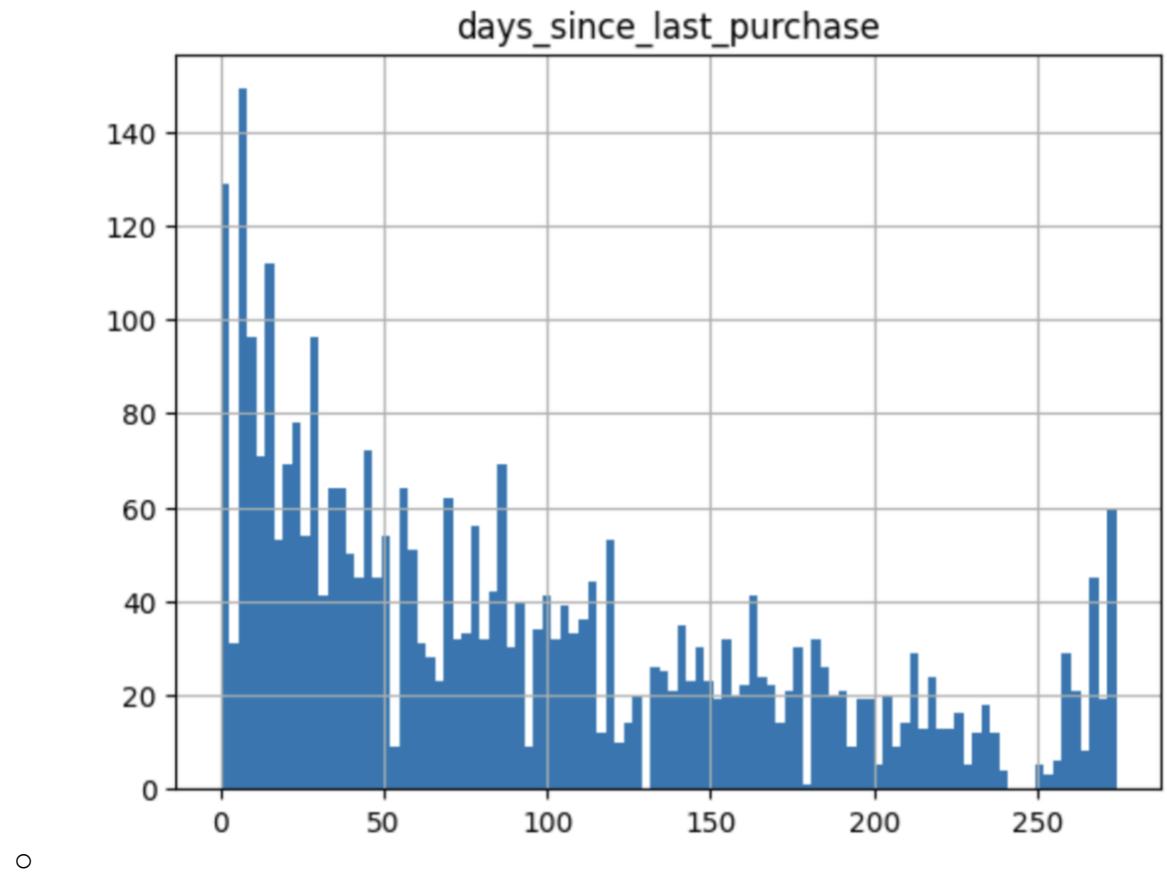
- The predicted result is pretty close to the actual average

The screenshot shows the Google Cloud BigQuery interface with a query editor and results viewer. The query runs a self-join between the 'jobs\_in\_data.prediction' and 'jobs\_in\_data.jobs' tables to calculate average salaries by job category. The results table shows the same data as the previous screenshot.

Row	job_category	avg_salary	cat	sal
1	Data Analysis	107456.7692...	Data Analysis	108505.7213...
2	Data Science and Research	163228.9823...	Data Science and Research	163758.5759...
3	Data Engineering	144618.6022...	Data Engineering	146197.6561...
4	Data Architecture and Mo...	148055.1346...	Data Architecture and Mo...	156002.3590...
5	Machine Learning and AI	174432.2155...	Machine Learning and AI	178925.8473...
6	BI and Visualization	133859.6923...	BI and Visualization	135092.1022...
7	Leadership and Managem...	141711.8120...	Leadership and Managem...	145476.0198...
8	Data Management and Str...	92538.23529...	Data Management and Str...	103139.9344...
9	Data Quality and Operations	98587.68749...	Data Quality and Operations	100879.4727...

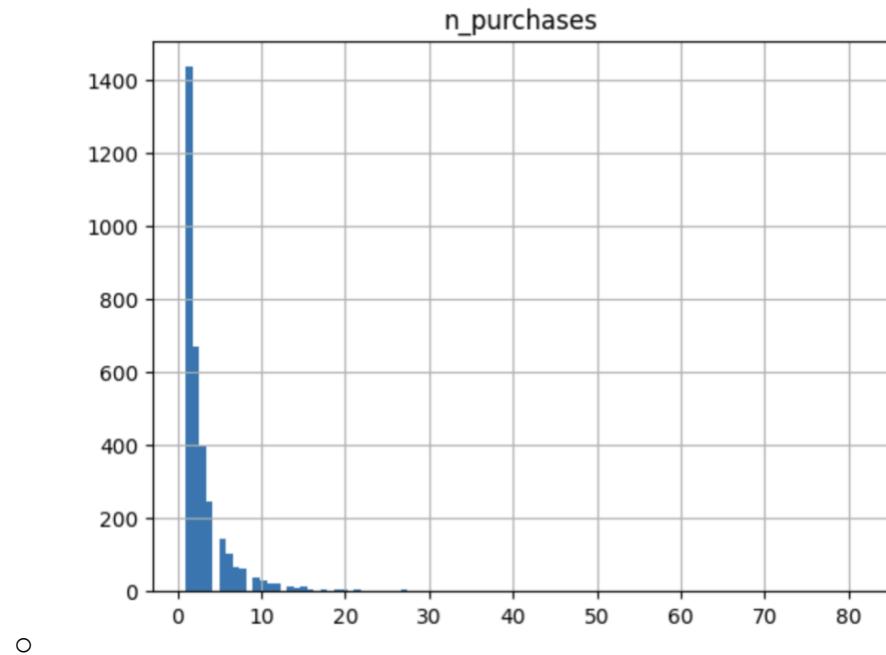
## 8. Question 8

- Cell 11 histogram



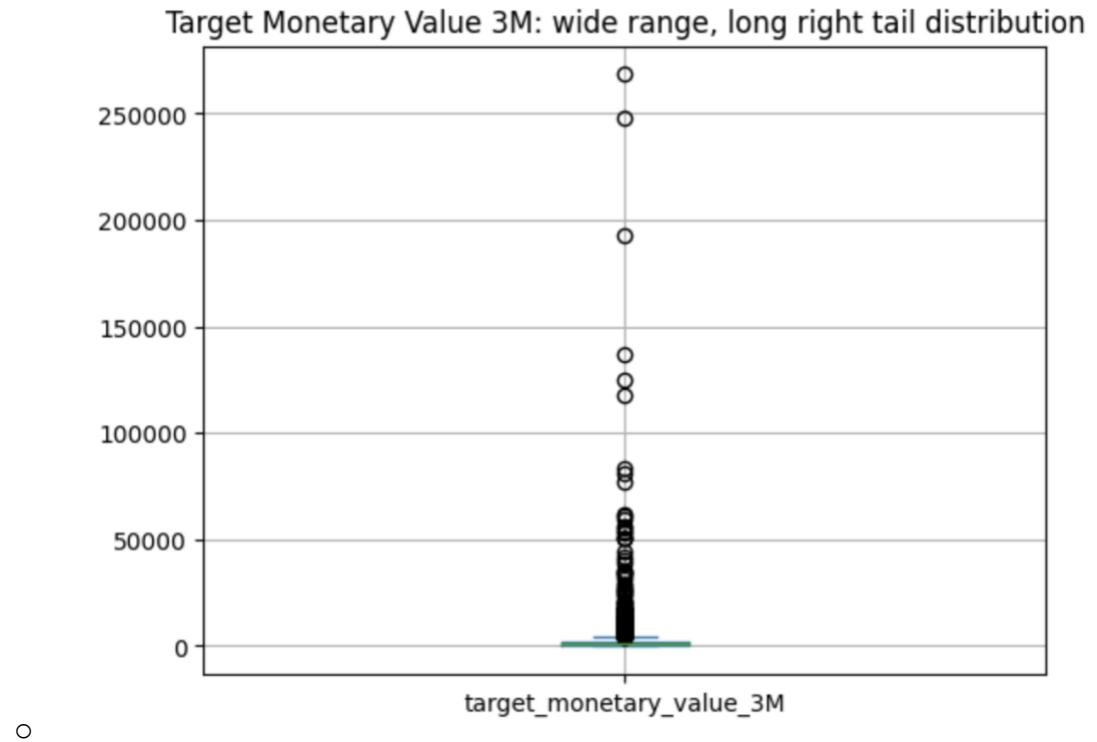
○

- Cell 14 histogram

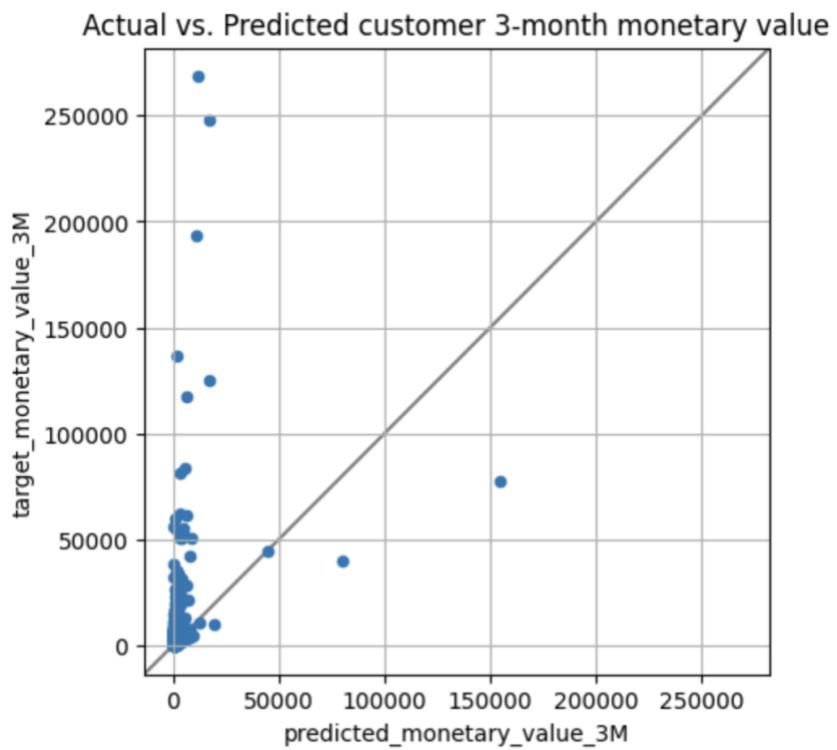


○

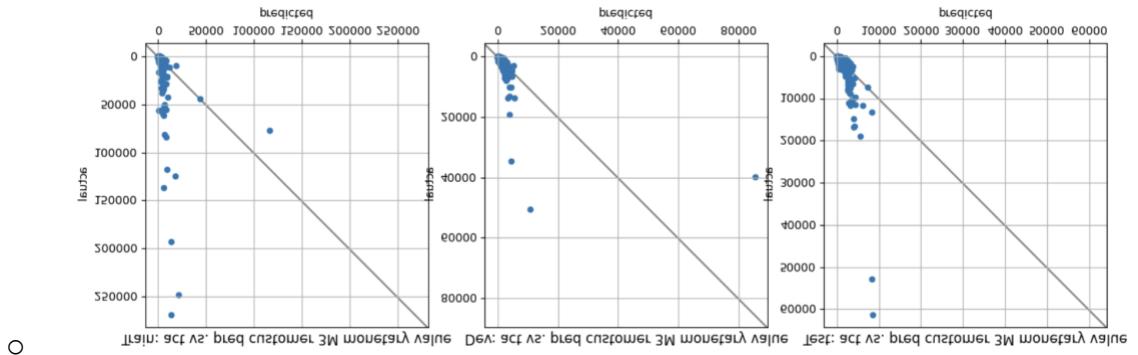
- Cell 17 plot



- Cell 21 plot



- Cell 23 plot



## 9. Question 9

- Describe queries
  - Cell 10
    - Return all rows
  - Cell 11
    - Count the total number of events and the total number of unique user
  - Cell 12
    - This query returns all users that are considered bounced or churned
  - Cell 13
    - Returns a matrix of the total count of whether someone bounced, churned, or both.
  - Cell 14
    - Returns the percentage of users who have churned
  - Cell 15
    - This query returns the metadata of each user event. It shows which country they accessed it from, the operating system, and language and region.

- Cell 16
  - This query counts the total number of events that occurred.
- Cell 17
  - This query counts the total number of events per user.
- Cell 18
  - This query combines these three intermediate views into one table. It also returns if an instance is used in the training and testing set.
- Cell 22

```
[22]: %%bigquery --project $PROJECT_ID
SELECT *
FROM
ML.TRIAL_INFO(MODEL `bqmlga4.churn_xgb`);
```

	trial_id	hyperparameters	hparam_tuning_evaluation_metrics	training_loss	eval_loss	status	error_message
0	1	{"learn_rate": 0.1, "max_tree_depth": 6}	{"roc_auc": 0.7873576423576424}	0.436812	0.461768	SUCCEEDED	None
1	2	0.0802623307307415, "max_tree_...	{"roc_auc": 0.7713226773226773}	0.463238	0.477591	SUCCEEDED	None
2	3	0.05867949061178646, "max_tree_...	{"roc_auc": 0.7804395604395604}	0.470266	0.488379	SUCCEEDED	None
3	4	0.05689253515969765, "max_tree_...	{"roc_auc": 0.7658771228771228}	0.483619	0.494980	SUCCEEDED	None
4	5	0.08409442404044874, "max_tree_...	{"roc_auc": 0.7738431568431569}	0.458928	0.473466	SUCCEEDED	None
5	6	{"learn_rate": 0.1, "max_tree_depth": 5}	{"roc_auc": 0.77594005994006}	0.450466	0.465411	SUCCEEDED	None
6	7	0.08420465492095495, "max_tree_...	{"roc_auc": 0.7868551448551449}	0.446473	0.467997	SUCCEEDED	None

- Cell 23

The screenshot shows a Jupyter Notebook interface running in a browser. The notebook is titled "lab\_exercise.ipynb". The code cell [24] contains a BigQuery query to calculate a confusion matrix:

```
%bqquery --project $PROJECT_ID
SELECT
    expected_label,
    _0 AS predicted_0,
    _1 AS predicted_1
FROM
    ML.CONFUSION_MATRIX(MODEL bqmlga4.churn_xgb)
WHERE trial_id=1;
```

The output shows the confusion matrix:

	expected_label	predicted_0	predicted_1
0	0	585	19
1	1	166	29

Cell [25] contains a command to plot an ROC curve:

```
[ ]: %bqquery df_roc --project $PROJECT_ID
SELECT * FROM ML.ROC_CURVE(MODEL bqmlga4.churn_xgb)
```

- Cell 26

The screenshot shows a Jupyter Notebook interface running in a browser. The notebook is titled "lab\_exercise.ipynb". The code cell [27] contains a BigQuery query to explain the model's global attribution:

```
[27]: %bqquery --project $PROJECT_ID
SELECT
    *
FROM
    ML.GLOBAL_EXPLAIN(MODEL bqmlga4.churn_xgb)
ORDER BY
    attribution DESC;
```

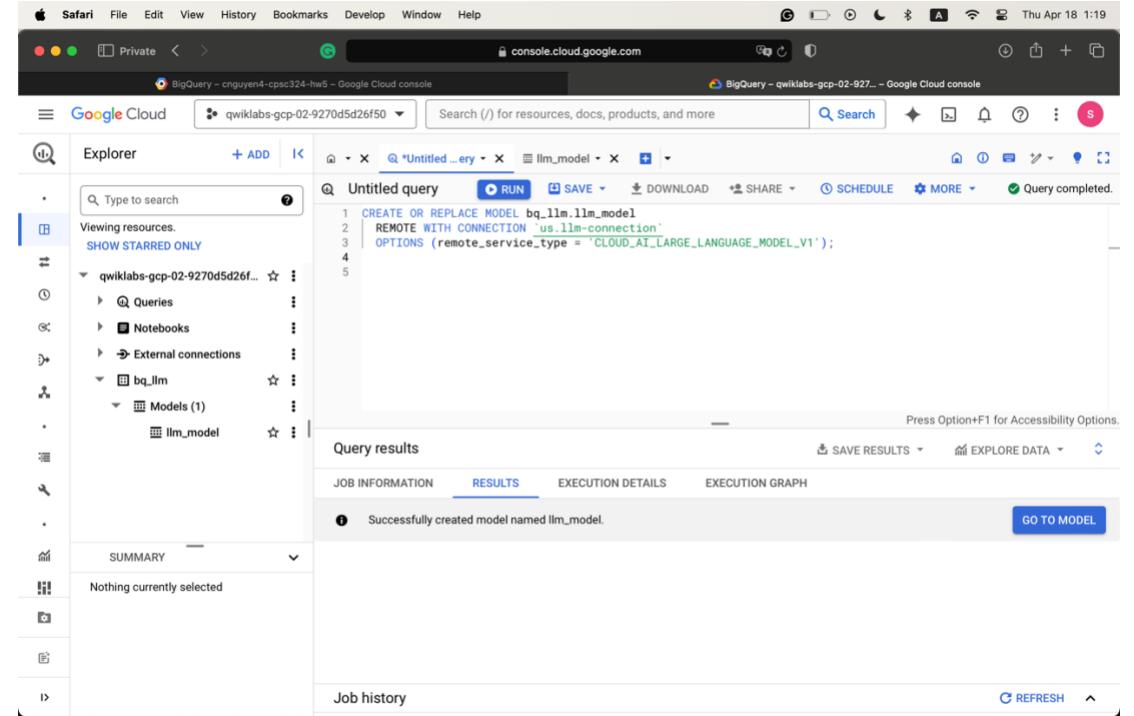
The output shows the attribution weights for each feature:

trial_id	feature	attribution
0	cnt_user_engagement	0.191482
1	user_first_engagement	0.090093
2	julianday	0.068122
3	operating_system	0.053907
4	cnt_post_score	0.015645
5	cnt_level_start_quickplay	0.014644
6	cnt_level_end_quickplay	0.013826
7	cnt_level_reset_quickplay	0.005394
8	language	0.005119
9	cnt_spend_virtual_currency	0.004703
10	dayofweek	0.004442
11	cnt_level_complete_quickplay	0.004008

- The table above shows how much weight the model puts into each feature that is used to predict the result.

## 10. Question 10

- Create a model



The screenshot shows the Google Cloud BigQuery interface. In the left sidebar, under 'Explorer', there is a tree view of resources. Under 'bq\_llm', there is a single item 'ilm\_model'. In the main area, an 'Untitled query' tab is open with the following SQL code:

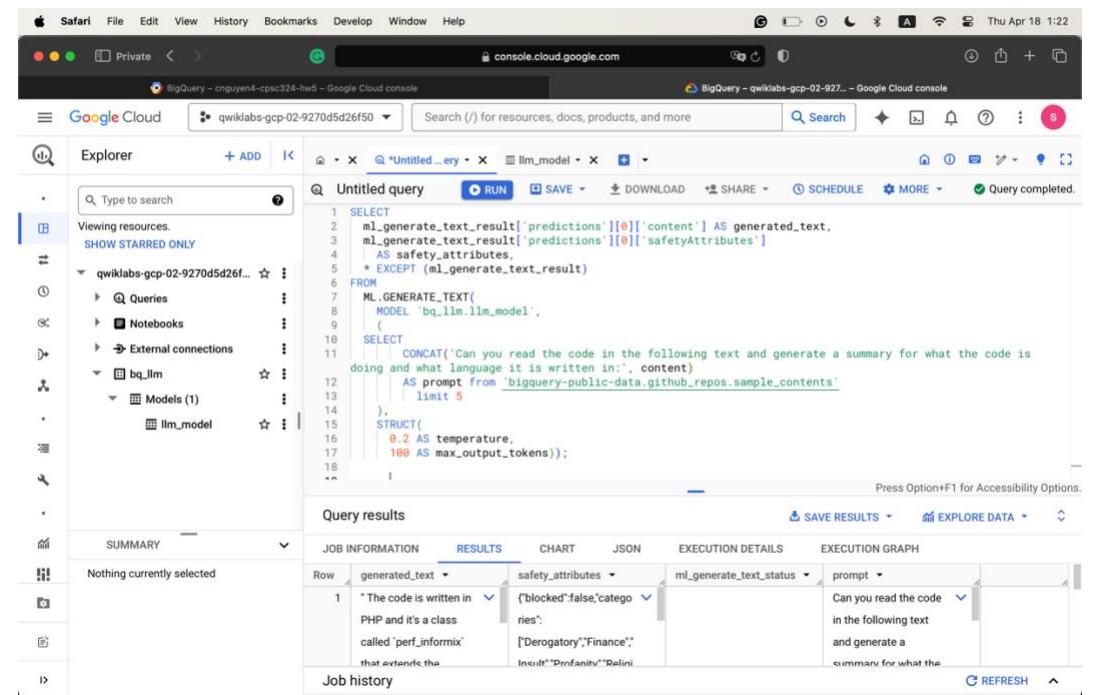
```

CREATE OR REPLACE MODEL bq_llm.llm_model
REMOTE WITH CONNECTION 'us_llm-connection'
OPTIONS (remote_service_type = 'CLOUD_AI_LARGE_LANGUAGE_MODEL_V1');

```

The 'RESULTS' tab is selected, showing the message: 'Successfully created model named ilm\_model.' A 'GO TO MODEL' button is available.

- Generate text using the model.



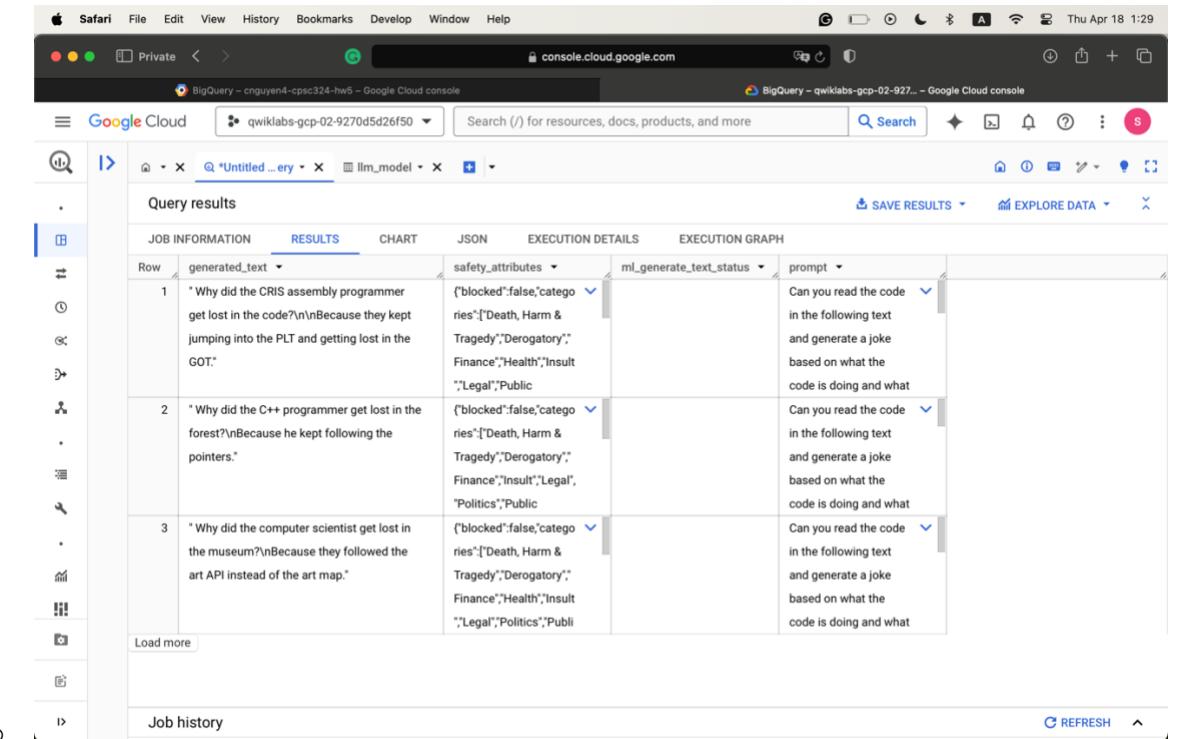
The screenshot shows the Google Cloud BigQuery interface. In the left sidebar, under 'Explorer', there is a tree view of resources. Under 'bq\_llm', there is a single item 'ilm\_model'. In the main area, an 'Untitled query' tab is open with the following SQL code:

```

SELECT
  ml_generate_text_result['predictions'][0]['content'] AS generated_text,
  ml_generate_text_result['safety_attributes'],
  * EXCEPT (ml_generate_text_result)
FROM
  ML.GENERATE_TEXT(
    MODEL 'bq_llm.llm_model',
    (
      SELECT
        CONCAT('Can you read the code in the following text and generate a summary for what the code is doing and what language it is written in:', content)
        AS prompt
        FROM `bigquery-public-data.github_repos.sample_contents`
        LIMIT 5
    ),
    STRUCT(
      0.2 AS temperature,
      100 AS max_output_tokens));
  
```

The 'RESULTS' tab is selected, showing the generated text and its safety attributes. The generated text is: 'The code is written in PHP and it's a class called "perf\_informix" that extends the'. The safety attributes are: 'blocked':false, 'categories': ["Derogatory", "Finance", "Insult", "Profanity", "Religious"]. The ml\_generate\_text\_status is: 'OK'. The prompt is: 'Can you read the code in the following text and generate a summary for what the'. A 'REFRESH' button is at the bottom right.

- The generated\_text column is the result returned by the model after passing in the prompt, as well as the data that we want to use. It returns the language in which the code was written, as well as a summary of what is going on inside.
- I tried the joke prompt, and it returned the following:



The screenshot shows a Google Cloud BigQuery results page in a Safari browser. The query results table has columns: Row, generated\_text, safety\_attributes, ml\_generate\_text\_status, and prompt. The table contains three rows of data:

Row	generated_text	safety_attributes	ml_generate_text_status	prompt
1	"Why did the C++ programmer get lost in the forest?\nBecause he kept following the pointers."	{"blocked":false,"categories":["Death, Harm & Tragedy","Derogatory","Finance","Health","Insult","Legal","Public"]}		Can you read the code in the following text and generate a joke based on what the code is doing and what
2	"Why did the computer scientist get lost in the museum?\nBecause they followed the art API instead of the art map."	{"blocked":false,"categories":["Death, Harm & Tragedy","Derogatory","Finance","Insult","Legal","Politics","Public"]}		Can you read the code in the following text and generate a joke based on what the code is doing and what
3	"Why did the CRIS assembly programmer get lost in the GOT?"	{"blocked":false,"categories":["Death, Harm & Tragedy","Derogatory","Finance","Health","Insult","Legal"]}		Can you read the code in the following text and generate a joke based on what the code is doing and what

- This is not particularly very funny, though.
- I tried another one: Can you read the code in the following text and generate a critique of that language?
  - Some examples:

Row	generated_text	safety_attributes	ml_generate_text_status	prompt
1	" **Critique of the JavaScript code:**\n\n**1. Lack of Modular Structure:**\n\n The code lacks a clear modular structure, making it difficult to understand and maintain. It would be better to organize the code into distinct modules or functions to improve readability and reusability.\n\n**2. Excessi...	{"blocked":false,"categories":["Death, Harm & Tragedy","Derogatory","Finance","Health","Illicit Drugs","Insult","Legal"]}	Can you read the code in the following text and generate critique of that language:module("supp	Can you read the code in the following text and generate critique of that language:#
2	" The code is a Makefile for the BCM47XX specific kernel interface routines under Linux. It defines a list of object files that will be built when the Makefile is run.\n\nThe Makefile is well-written and easy to understand. It uses a consistent indentation style and the comments are clear and concis...	{"blocked":false,"categories":["Derogatory"],"safetyRatings":[]}	Can you read the code in the following text and generate critique of that language:# Makefile for the	Can you read the code in the following text and generate critique of that language:{ "apiVersion": "0.4.0",
3	" **Critique of the language used in the provided API documentation:**\n\n**1. Inconsistent terminology:**\n\n - The term 'edge' is used inconsistently throughout the documentation. In some places, it refers to the edge resource itself, while in others, it refers to the edge object within a respons...	{"blocked":false,"categories":["Death, Harm & Tragedy","Derogatory","Finance","Health","Insult","Legal"],"safetyRatings":[]}	Can you read the code in the following text and generate critique of that language:{ "apiVersion": "0.4.0",	Can you read the code in the following text and generate critique
4	" **Critique of the given Python code:**\n\n**1. Lack of Docstrings:**\n\n The code lacks proper docstrings for the 'FlaskTestClient' class and its methods. Docstrings provide essential information about the purpose,	{"blocked":false,"categories":["Derogatory"],"safetyRatings":[]}	Can you read the code in the following text and generate critique	