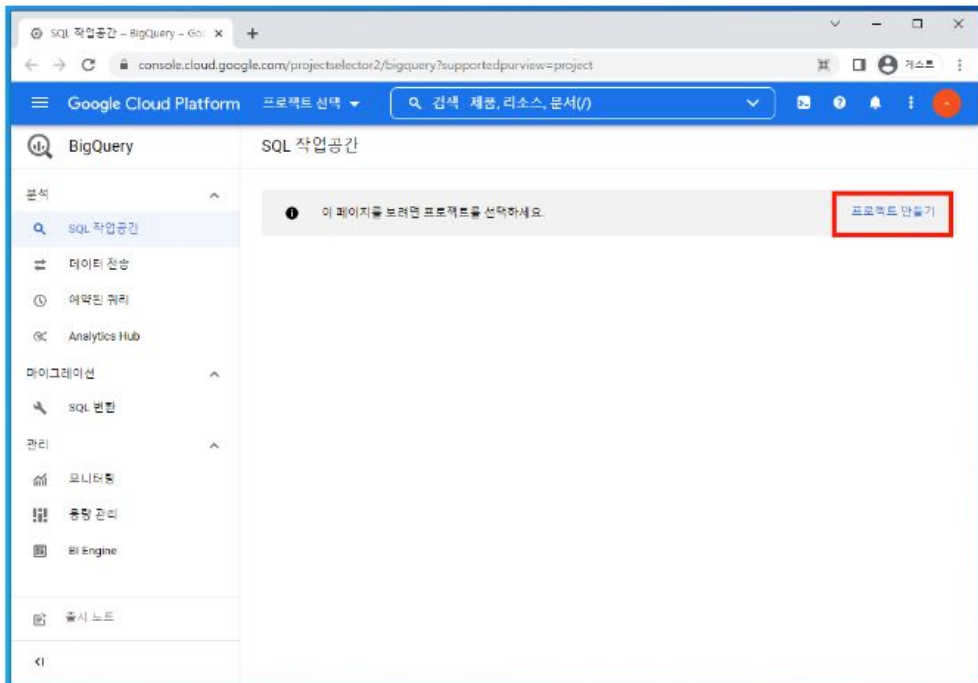


BigQuery

프로젝트 및 데이터 세트 만들기

- 수업시간에 다룬 아래 링크로 들어가 Google Cloud Platform Project를 하나 생성합니다.
- 단, 이때 프로젝트 이름에 언더바(underscore, _)가 들어가면 안 됩니다!

<https://console.cloud.google.com/bigquery>



Personal
Google
Account

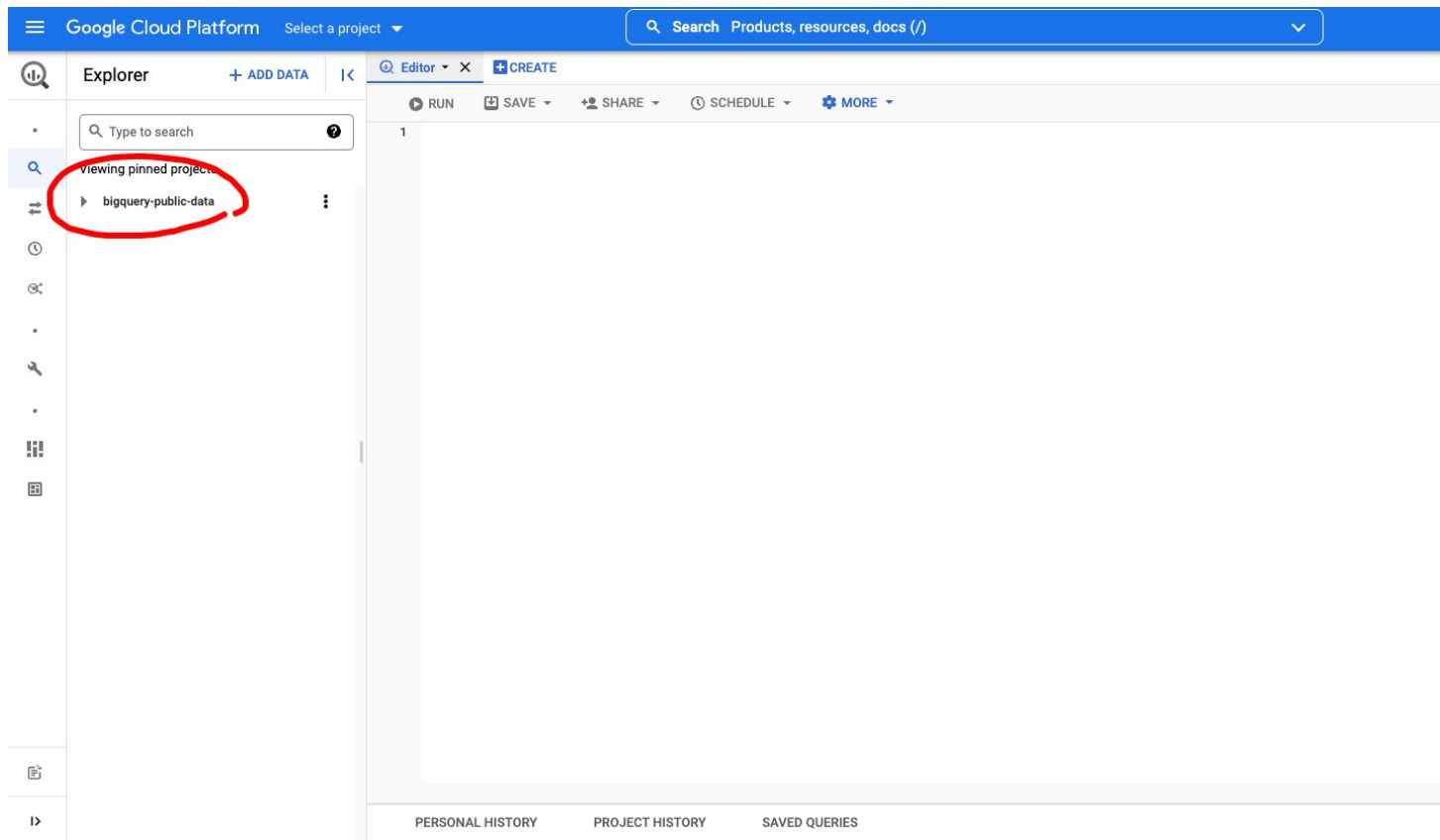


SNU
G Suite
Account

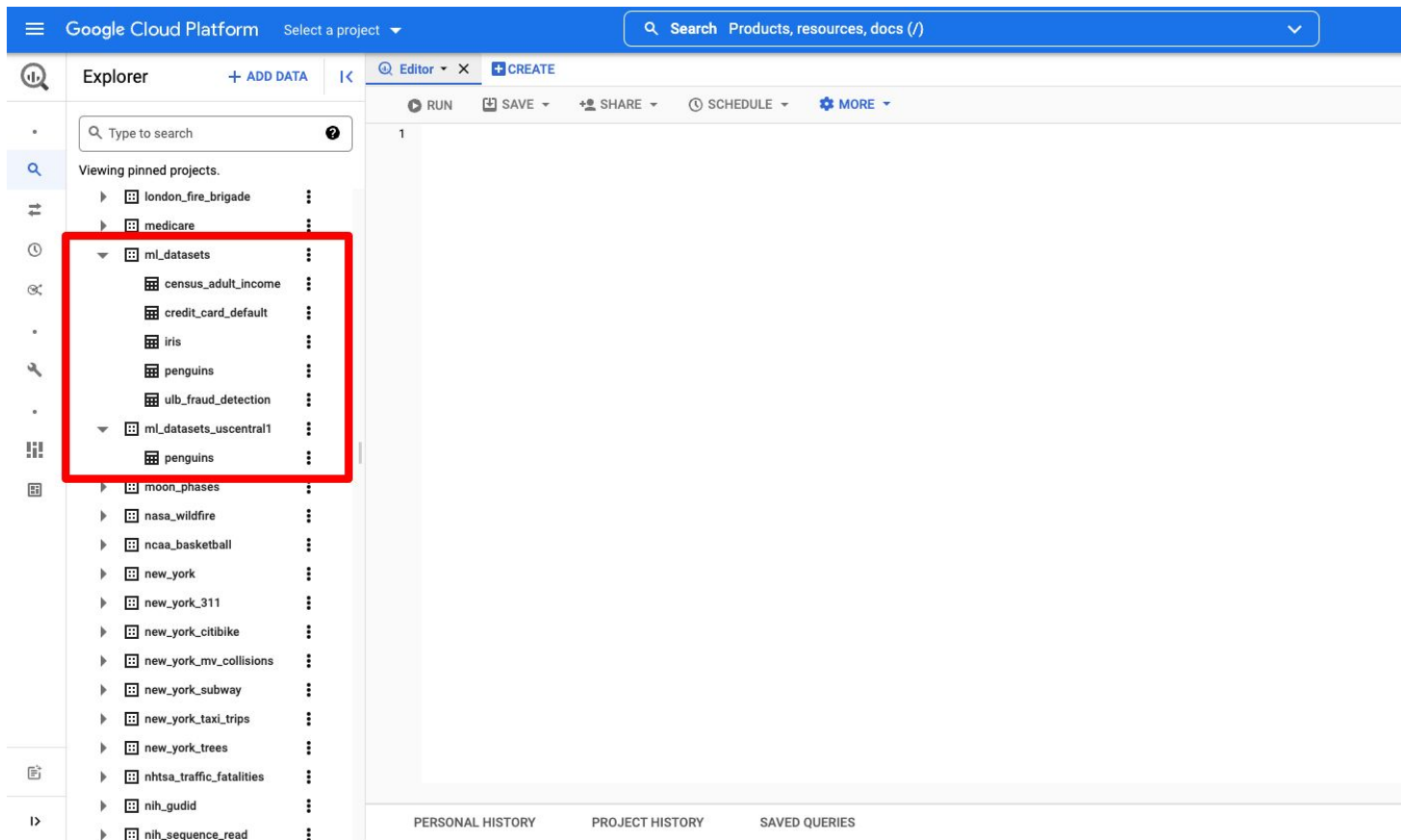


- 그런 다음 아래 링크를 클릭하면 아래 이미지처럼 왼쪽 탭에 **bigquery-public-data**가 하나 생성됩니다.

<https://console.cloud.google.com/bigquery?project=bigquery-public-data&page=project>



- 펼쳐서 사용할 데이터 하나를 골라줍니다
(웬만하면 **ml-dataset** 중 하나를 고르는게 편할 것 같습니다... 나머지는 전처리 고민이 많아 보여요)



- 데이터를 하나 골랐다면, 해당 데이터를 클릭한 뒤 **DETAIL**로 들어가 **Table ID**를 복사합니다.

The screenshot displays the Google Cloud Platform BigQuery interface. On the left, the 'Explorer' panel shows a list of pinned projects. The 'credit_card_default' table is highlighted with a red box. On the right, the 'DETAILS' tab is selected for the 'credit_card_default' table. The 'Table ID' is highlighted with a red box and is 'bigquery-public-data.ml_datasets.credit_card_default'. Other table information is also visible.

Google Cloud Platform Select a project ▼ Search Products, resources, docs (/)

Explorer + ADD DATA <

Type to search ?

Viewing pinned projects.

- ▶ london_fire_brigade
- ▶ medicare
- ▼ ml_datasets
 - census_adult_income
 - credit_card_default**
 - iris
 - penguins
 - ulb_fraud_detection
- ▶ ml_datasets_uscentral1
- ▶ moon_phases
- ▶ nasa_wildfire

credit_card_default QUERY SHARE COPY SNAPSHOT

SCHEMA **DETAILS** PREVIEW

Table info

Table ID	bigquery-public-data.ml_datasets.credit_card_default
Table size	557.85 KB
Long-term storage size	557.85 KB
Number of rows	2,965
Created	Mar 30, 2019, 1:51:54 AM UTC+9
Last modified	Mar 30, 2019, 1:55:15 AM UTC+9
Table expiration	NEVER
Data location	US
Default collation	[null]
Description	

- 만들어진 프로젝트로 돌아간 뒤, 데이터를 한 번 확인해봅니다.
- Query: `SELECT * FROM `TABLE_ID``
 - 주의: 여기서 Table ID 주변 첨자는 " (따옴표)가 아니라 숫자 키 1 옆에 있는 ` (grave accent)
 - Mac의 경우 한글 입력상태에서는 ~(option)과 동시에 눌러야 합니다.



▶ RUN
💾 SAVE ▾
👤 SHARE ▾
🕒 SCHEDULE ▾
⚙️ MORE ▾
✅ This query will process 557.85 KB when run

```
1 SELECT * FROM `bigquery-public-data.ml_datasets.credit_card_default`
```

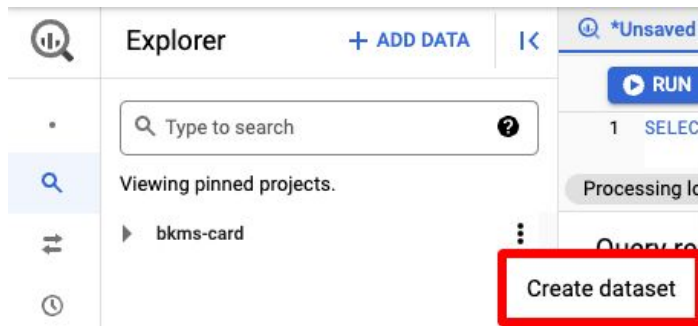
Processing location: US Press Alt+F1 for Accessibility Options

Query results

📄 SAVE RESULTS ▾
📊 EXPLORE DATA ▾
↕

JOB INFORMATION		RESULTS		JSON	EXECUTION DETAILS												
Row	id	limit_balance	sex	education_level	marital_status	age	pay_0	pay_2	pay_3	pay_4	pay_5	pay_6	bill_amt_1	bill_amt_2	bill_amt_3	bill_amt_4	
1	242.0	50000.0	1	1	2	39.0	0.0	0.0	0.0	0.0	0	0	47174.0	47974.0	48630.0	50803.0	↕
2	1822.0	110000.0	2	1	2	29.0	0.0	0.0	0.0	0.0	0	0	48088.0	45980.0	44231.0	32489.0	↕
3	5046.0	270000.0	1	1	2	36.0	0.0	0.0	0.0	2.0	0	0	78630.0	68921.0	46512.0	40335.0	↕

- 데이터 세트를 만들기 위해서는 프로젝트 옆 점 3개 버튼을 누르면 나오는 “Create dataset” 버튼을 눌러준 뒤, 나타나는 팝업창에 데이터 세트 아이디를 임의로 만들어서 입력해줍니다.
- 단 데이터 세트 아이디에는 **dash(-)**는 안 되고 언더바(_)만 사용 가능!



Create dataset

Project ID
bkms-card CHANGE

Dataset ID *
card_data
Letters, numbers, and underscores allowed

Data location ▼ ?

Default table expiration
☐ Enable table expiration ?

Default maximum table age Days

Advanced options ▼

CREATE DATASET CANCEL



학습 데이터 생성

- 무료계정(Sandbox)의 특성상 **public dataset**마저도 그대로 사용할 수 없기 때문에 학습 데이터를 만들기 위해서는 **CREATE VIEW**를 통해 **VIEW**를 만들어줘야 합니다.
- 단, **VIEW**를 만들 때 주의할 점은
 - 한 번 만들어진 **VIEW**는 수정이 불가능합니다. 따라서 **VIEW**를 만든 이후에 **feature selection**을 할 수 없고, 먼저 사용할 **feature**를 고른 다음 **VIEW**로 불러와야 합니다.
 - 몇 종류의 전처리는 **VIEW**를 만들 때 수행해주는 것이 좋습니다.
 - 날짜 형식에서 연도와 월 변수를 뽑는 경우 (2022.05.15 -> 2022, 5)
 - NULL value 제거
 - Training - test - prediction용 데이터를 직접 구분해주는 경우
 - 실행해본 결과 **prediction**용 데이터는 직접 구분해주는 것이 좋을 것 같습니다...
안해주니까 전체 데이터를 대상으로 **prediction**하네요
 - 그런 의미에서 데이터 고를 때도 **prediction** 구분 기준을 만들 수 있는 데이터를 고르는 게 좋아 보입니다(ex. 데이터 고유 아이디가 부여되어 있다든지...)

- RUN

This query will process 23.16 KB when run.

1 SELECT distinct pay_0 FROM `bigquery-public-data.ml_datasets.credit_card_defa

Processing location: US

Press Alt+F1 for Accessibility Options.

Query results

SAVE RESULTS EXPLORE DATA

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

Row	pay_0
1	0.0
2	-1.0
3	2.0
4	1.0
5	3.0
6	-2.0
7	4.0
8	5.0
9	6.0
10	7.0
11	8.0

미리 걸러낼 수 있는
부분은 아예 걸러낸
다음에 **VIEW**로
만들어주는 것이
편할 것 같습니다...

pay_0 ~ pay_6은
범위를 벗어나는
값이 너무 많아 아예
배제!

- VIEW 만들 수 있는 쿼리

```
CREATE OR REPLACE VIEW `card_data.card_view` AS
```

```
SELECT limit_balance, sex, marital_status, age,
```

```
(bill_amt_1+bill_amt_2+bill_amt_3+bill_amt_4+bill_amt_5+bill_amt_6)/6 AS
```

```
avg_bill_amt,
```

```
(pay_amt_1+pay_amt_2+pay_amt_3+pay_amt_4+pay_amt_5+pay_amt_6)/6 AS
```

```
avg_pay_amt,
```

```
default_payment_next_month,
```

```
CASE
```

```
  WHEN education_level = '1' THEN '1'
```

```
  WHEN education_level = '2' THEN '2'
```

```
  WHEN education_level = '3' THEN '3'
```

```
  ELSE '0'
```

```
END AS education,
```

```
CASE
```

```
  WHEN MOD(CAST(id AS INTEGER), 10) < 7 THEN "training"
```

```
  WHEN MOD(CAST(id AS INTEGER), 10) >= 7 AND MOD(CAST(id AS INTEGER), 10) < 9 THEN "evaluation"
```

```
  ELSE "prediction"
```

```
END AS annotation
```

```
FROM `bigquery-public-data.ml_datasets.credit_card_default`
```


```
WHERE marital_status > '0' and education_level > '0'
```


전처리


Training / Validation / Test 구분


NULL Value 제거


- CSV file로 저장 및 확인 가능!

 RUN

 SAVE

 SHARE

 SCHEDULE

 MORE

✓ This query will process 382.21 KB when run.

1

select * from `card_data.card_view`

Processing location: US

Press Alt+F1 for Accessibility Options

Query results

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

Row	limit_balance	sex	marital_status	age	avg_bill_amt	avg_pay_amt	default_payment_next_month	educa
1	50000.0	1	2	39.0	40207.333333333336	2133.3333333333335	0	1
2	110000.0	2	2	29.0	36227.166666666664	2335.0	0	1
3	270000.0	1	2	36.0	48953.166666666664	3026.5	0	1
4	130000.0	1	1	45.0	61629.666666666664	2489.0	0	1
5	50000.0	1	2	24.0	26558.5	964.5	0	1
6	20000.0	1	2	29.0	17280.833333333332	1361.6666666666667	0	1
7	220000.0	1	2	38.0	207519.16666666666	7826.833333333333	0	1
8	50000.0	1	1	42.0	37647.666666666664	1612.1666666666667	0	1
9	170000.0	1	1	41.0	90769.5	5830.0	0	1
10	50000.0	2	2	24.0	42559.5	2012.6666666666667	0	1
11	170000.0	2	2	27.0	77753.0	2875.0	0	2
12	120000.0	2	2	37.0	99106.0	4100.1666666666667	0	2

SAVE RESULTS

EXPLORE DATA

CSV (Google Drive)

Save up to 1 GB of results to Google Drive.

CSV (local file)

Save up to 10 MB locally.

JSON (Google Drive)

Save up to 1 GB of results to Google Drive.

JSON (local file)

Save up to 10 MB locally.

BigQuery table

Save results as a BigQuery table.

Google Sheets

Save up to 10 MB to Google Sheets.

Copy to Clipboard

Copy up to 1 MB to the clipboard.

데이터 전처리 및 모델 생성

- 기본적으로 모델 생성과 동시에 전처리를 지정할 수 있습니다.
 - (개인적으로 전처리는 **label encoding**, 더미변수화와 표준화 정도만 해주면 충분할 것 같습니다.)
- 전처리에는 크게 2가지!
 - 모델을 생성할 때 수동 전처리하는 방법
 - SQL 구문을 활용한 전처리 (주로 더미변수화)
 - BigQuery에서 직접 제공하는 사전처리 함수를 사용한 전처리 (주로 표준화)
<https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-preprocessing-functions>
 - TRANSFORM 명령어를 사용한 자동 전처리
- 주의할 점
 - **TRANSFORM**을 사용하는 경우, **TRANSFORM**에 명시한 **Feature**만 모델에 포함되는 것 같습니다.
 - 모델 생성 후 **SCHEMA**를 통해서 확인할 수 있음
 - 이 말인즉슨 **TRANSFORM**을 사용하면 전처리가 필요 없는 값도 자동으로 전처리를 해줘야 하는 문제 발생
 - **TRANSFORM**을 사용할 경우, **target variable**이 반드시 **TRANSFORM** 안에 포함되어야 합니다!

- 모델 생성시 SQL을 통한 수동 전처리 전체 쿼리

```
CREATE OR REPLACE MODEL `card_data.card_model_manual2`  
OPTIONS (  
  model_type = "LOGISTIC_REG",  
  auto_class_weights=TRUE,  
  data_split_method="NO_SPLIT",  
  input_label_cols=["is_default"]  
) AS  
SELECT
```

Target에도 전처리를 해주었다면, 전처리 후
생성해준 column 이름을 Target에 넣어주어야 함!

```
IF (sex = '1', 1, 0) AS is_male,  
IF (marital_status = '1', 1, 0) AS is_married,  
IF (marital_status = '2', 1, 0) AS is_single,  
IF (education = '1', 1, 0) AS is_graduate,  
IF (education = '2', 1, 0) AS is_univ,  
IF (education = '3', 1, 0) AS is_high,  
IF (default_payment_next_month = '1', 1, 0) AS is_default,  
ML.STANDARD_SCALER(limit_balance) OVER() as std_balance,  
ML.STANDARD_SCALER(age) OVER() as std_age,  
ML.STANDARD_SCALER(avg_bill_amt) OVER() as std_bill,  
ML.STANDARD_SCALER(avg_pay_amt) OVER() as std_pay  
FROM `card_data.card_view`  
WHERE annotation = "training"
```

← One-hot Encoding SQL query

← 표준화 using 사전처리 함수

- TRANSFORM을 통한 자동 전처리
 - 그냥 변수만 집어넣으면 알아서 아래와 같이 해준다고 합니다.

Feature Type	TRANSFORM 변환 내역	비고
숫자형 (INT64, NUMERIC, BIGNUMERIC, FLOAT64)	Standardization	변수에서 평균을 빼고 표준편차로 나누는 작업
BOOLEAN, STRING, BYTES, DATE, DATETIME, TIME	One-hot Encoding	한 Feature에 K개의 서로 다른 값이 있다면 K-1개의 변수가 새로 생성
ARRAY	Multi-hot Encoding	마치 $\log_2 N$ 주처럼 N개의 feature를 개짜리의 bit로

- TRANSFORM을 통한 자동 전처리 전체 쿼리

```
CREATE OR REPLACE MODEL `card_data.card_model2`
```

```
TRANSFORM(  
sex, marital_status, education, default_payment_next_month,  
limit_balance, age, avg_bill_amt, avg_pay_amt  
)
```

← 여기다 넣어놓으면 자동으로 전처리 끝!

```
OPTIONS (
```

```
model_type = "LOGISTIC_REG",
```

```
auto_class_weights=TRUE,
```

```
data_split_method="NO_SPLIT",
```

```
input_label_cols: ["default_payment_next_month"]
```

← 원래 column name 그대로!

```
) AS
```

```
SELECT
```

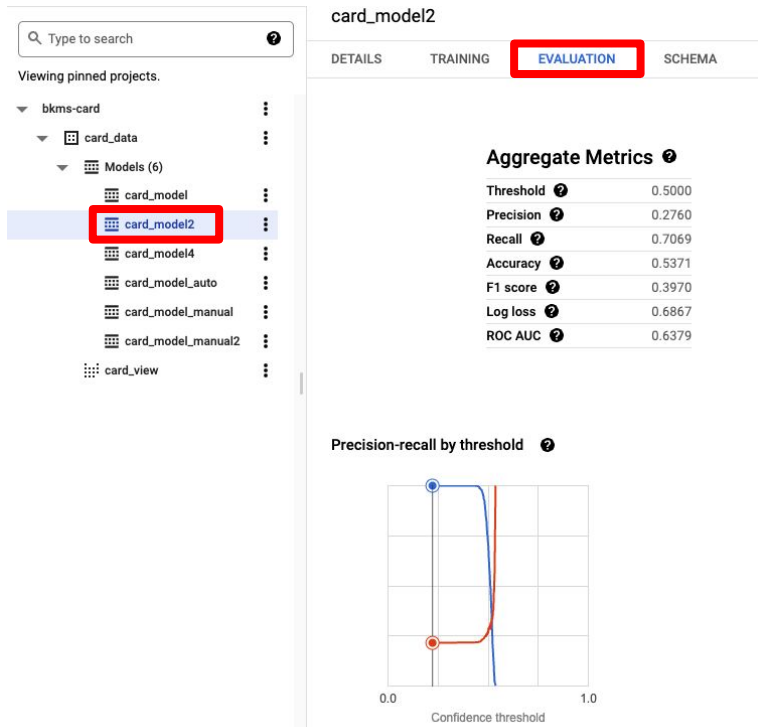
```
* EXCEPT(annotation)
```

← annotation를 제외한 모든 column을
모두 사용한다는 의미

```
FROM `card_data.card_view`
```

```
WHERE annotation = "training"
```


● 결과 비교



card_model2

[QUERY MODEL](#)

DETAILS TRAINING EVALUATION **SCHEMA**

Labels

Filter Enter property name or value

Field name	Type	Mode	Description
predicted_default_payment_next_month	STRING	NULLABLE	

Features

Filter Enter property name or value

Field name	Type	Mode	Description
education	STRING	NULLABLE	
sex	STRING	NULLABLE	
marital_status	STRING	NULLABLE	
age	FLOAT64	NULLABLE	
avg_bill_amt	FLOAT64	NULLABLE	
avg_pay_amt	FLOAT64	NULLABLE	
limit_balance	FLOAT64	NULLABLE	

- 결과 비교

- Score Table

- 자동 전처리

Aggregate Metrics ?

Threshold ?	0.5000
Precision ?	0.2760
Recall ?	0.7069
Accuracy ?	0.5371
F1 score ?	0.3970
Log loss ?	0.6867
ROC AUC ?	0.6379

Aggregate Metrics ?

Threshold ?	0.5000
Precision ?	0.2727
Recall ?	0.6846
Accuracy ?	0.5386
F1 score ?	0.3901
Log loss ?	0.6869
ROC AUC ?	0.6261

- 수동 전처리

- ROC Curve

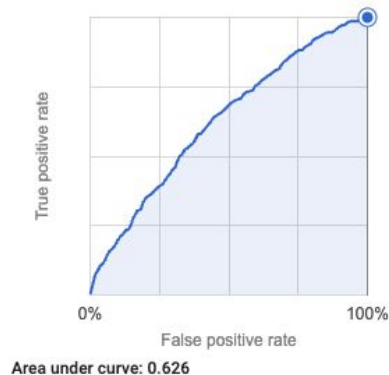
- 자동 전처리

- 수동 전처리

ROC curve ?



ROC curve ?



모델 options 설정

- 어떤 모델을 사용할지, Training-validation을 어떤 비율로 분할할지, hyper-parameter는 어떻게 할지를 모두 결정
 - OPTIONS()라는 명령 안에 넣어서 처리
- 모델 설정
 - `model_type="사용할_모델_이름"`의 형식으로 지정하며, 사용할 수 있는 모델에는 다음이 있습니다.
(하이라이트된 부분은 hyper-parameter tuning 가능 모델)
 - Regression:
 - **LINEAR_REG, BOOSTED_TREE_REGRESSOR, DNN_REGRESSOR, DNN_LINEAR_COMBINED_REGRESSOR**, AUTOML_REGRESSOR,
 - Classification:
 - **LOGISTIC_REG, BOOSTED_TREE_CLASSIFIER, DNN_CLASSIFIER, DNN_LINEAR_COMBINED_CLASSIFIER**, AUTOML_CLASSIFIER
 - Others
 - **KMEANS, MATRIX_FACTORIZATION**, TENSORFLOW, PCA, ARIMA, AUTOENCODER
 - 참고로, 여기서 사용하는 boosting 방법은 XGBoost를 따른다고 합니다.

- Data-split method
 - Training-Validation 비율 나누는 방법을 정하는 argument
 - 주로 “DATA_SPLIT_METHOD”와 “DATA_SPLIT_EVAL_FRACTION”를 사용합니다.
 - DATA_SPLIT_METHOD의 값으로는 'AUTO_SPLIT', 'RANDOM', 'CUSTOM', 'SEQ', 'NO_SPLIT'를 사용하며
 - 이중 AUTO_SPLIT, RANDOM, NO_SPLIT을 제일 많이 사용할 것 (같습니다...)
 - AUTO_SPLIT
 - 데이터가 500개보다 적은 경우 전체를 학습에 사용
 - 데이터가 500 ~ 50000개인 경우 데이터의 20%를 validation set으로 사용
 - 50000개보다 많은 경우 10000개를 validation set으로 사용
 - RANDOM
 - Scikit-learn train_test_split의 shuffle과 비슷하게 데이터를 랜덤으로 추출
 - 이 경우 DATA_SPLIT_EVAL_FRACTION에 몇 %를 validation으로 사용할지 지정해야 합니다.
 - NO_SPLIT
 - 이미 validation set으로 사용할 데이터까지 지정해둔 경우에 사용합니다.

Model Evaluation & Prediction

- Evaluation

- 기본적인 쿼리의 형태는 아래와 같습니다.

```
SELECT * FROM
```

```
ML.EVALUATION(MODEL model_name,
```

```
(query)
```

← 사용할 column을 선택하는 쿼리

- 이때, TRANSFORM을 통해 전처리를 해준 경우, 여기서 다시 TRANSFORM을 사용하지 않아도 됩니다.
 - 원칙적으로는 TRANSFORM 속에 있던 column들은 모두 SELECT query에 포함해주어야 한다고 하지만, 포함하든 포함하지 않든 결과는 동일한 것 같습니다.
- 한편, 수동으로 전처리를 한 경우 전처리에 사용한 query를 그대로 모두 옮겨 적어주어야 합니다.
 - 실제 dataset의 column과 전처리 도중에 새로 생겨난 column이 서로 달라지는 문제가 발생하기 때문
- 다만 자동 전처리의 경우 Evaluation과 Prediction을 할 때마다 전처리를 수행하기 때문에, 수동 전처리의 경우보다 시간이 더 오래 걸릴 수는 있습니다.


- 자동 전처리 사용했을 때의 쿼리

```
SELECT * FROM
ML.EVALUATE(MODEL
`card_data.card_model2`,
(
  SELECT *
  FROM `card_data.card_view`
  WHERE annotation="evaluation"
)
)
```


- 수동 전처리 사용했을 때의 쿼리

```
SELECT * FROM
ML.EVALUATE(MODEL `card_data.card_model1`,
(
  SELECT
  IF (sex = '1', 1, 0) AS is_male,
  IF (marital_status = '1', 1, 0) AS is_married,
  IF (marital_status = '2', 1, 0) AS is_single,
  IF (education = '1', 1, 0) AS is_graduate,
  IF (education = '2', 1, 0) AS is_univ,
  IF (education = '3', 1, 0) AS is_high,
  IF (default_payment_next_month = '1', 1, 0) AS is_default,
  ML.STANDARD_SCALER(limit_balance) OVER() as std_balance,
  ML.STANDARD_SCALER(age) OVER() as std_age,
  ML.STANDARD_SCALER(avg_bill_amt) OVER() as std_bill,
  ML.STANDARD_SCALER(avg_pay_amt) OVER() as std_pay
  FROM `card_data.card_view`
  WHERE annotation="evaluation"
)
)
```

- 자동 전처리 사용했을 때의 Validation 결과

Query results							 SAVE RESULTS ▾
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS			
Row	precision	recall	accuracy	f1_score	log_loss	roc_auc	
1	0.26315789473684209	0.73913043478260865	0.53146853146853146	0.38812785388127852	0.6878599702258581	0.64870429570429566	

- 수동 전처리 사용했을 때의 Validation 결과

Query results							 SAVE RESULTS ▾
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS			
Row	precision	recall	accuracy	f1_score	log_loss	roc_auc	
1	0.25233644859813081	0.70434782608695656	0.52097902097902093	0.37155963302752293	0.68890940373441678	0.63549550449550452	

- Prediction
- 자동 전처리 사용했을 때의 쿼리

```
SELECT * FROM
ML.PREDICT(MODEL `card_data.card_model2`,
(
  SELECT *
  FROM `card_data.card_view`
  WHERE annotation="prediction"
)
)
```

- 수동 전처리 사용했을 때의 쿼리

```
SELECT * FROM
ML.PREDICT(MODEL `card_data.card_model_manual`,
(
  SELECT
  IF (sex = '1', 1, 0) AS is_male,
  IF (marital_status = '1', 1, 0) AS is_married,
  IF (marital_status = '2', 1, 0) AS is_single,
  IF (education = '1', 1, 0) AS is_graduate,
  IF (education = '2', 1, 0) AS is_univ,
  IF (education = '3', 1, 0) AS is_high,
  IF (default_payment_next_month = '1', 1, 0) AS is_default,
  ML.STANDARD_SCALER(limit_balance) OVER() as std_balance,
  ML.STANDARD_SCALER(age) OVER() as std_age,
  ML.STANDARD_SCALER(avg_bill_amt) OVER() as std_bill,
  ML.STANDARD_SCALER(avg_pay_amt) OVER() as std_pay
  FROM `card_data.card_view`
  WHERE annotation="prediction"
)
)
```


- Prediction
- 자동 전처리 사용했을 때의 결과

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	predicted_default_payment_next_month	↕	predicted_default_payment_next_month_probs	
1	1	▼	[{"label": "1", "prob": "0.51157269938864525"}, {"label": "0", "prob": "0.48842730061135475"}]	
2	0	▼	[{"label": "1", "prob": "0.497601823353766"}, {"label": "0", "prob": "0.502398176646234"}]	
3	0	▼	[{"label": "1", "prob": "0.49201102678921171"}, {"label": "0", "prob": "0.50798897321078829"}]	
4	0	▼	[{"label": "1", "prob": "0.4825089987874272"}, {"label": "0", "prob": "0.51749100121257285"}]	
5	1	▼	[{"label": "1", "prob": "0.505870782333104"}, {"label": "0", "prob": "0.494129217666896"}]	

- 수동 전처리 사용했을 때의 쿼리

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	predicted_is_default	↕	predicted_is_default_probs	
1	1	▼	[{"label": "1", "prob": "0.53584485029369855"}, {"label": "0", "prob": "0.46415514970630145"}]	
2	1	▼	[{"label": "1", "prob": "0.50576087811048931"}, {"label": "0", "prob": "0.49423912188951069"}]	
3	1	▼	[{"label": "1", "prob": "0.53123538712964058"}, {"label": "0", "prob": "0.46876461287035942"}]	
4	1	▼	[{"label": "1", "prob": "0.532785386955933"}, {"label": "0", "prob": "0.467214613044067"}]	
5	0	▼	[{"label": "1", "prob": "0.49601932818982442"}, {"label": "0", "prob": "0.50398067181017558"}]	

- (Optional) Hyper-parameter tuning

- 역시 **OPTIONS()**에 계속해서 추가해줌으로써 수행할 수 있고, 들어갈 수 있는 **argument**로는 다음이 있다.

Parameter	의미	가능한 값들	예시 및 기타
NUM_TRIALS	Tuning에 사용할 모델의 최대 개수	정수형, 1 ~ 100	
MAX_PARALLEL_TRIALS	Tuning에서 동시에 돌아갈 모델의 개수	정수형, 1 ~ 5	
HPARAM_TUNING_ALGORITHM	Tuning 알고리즘	'VIZIER_DEFAULT', 'RANDOM_SEARCH', 'GRID_SEARCH' 중 1	VIZIER_DEFAULT가 default & recommended
hyperparameter	조정 및 실험해볼 hyperparameter 값의 후보들	HPARAM_RANGE(min, max), HPARAM_CANDIDATES([candidates]) 중 1	ex. boosting의 경우 MAX_TREE_DEPTH=HPARAM_RANGE(3, 6)
HPARAM_TUNING_OBJECTIVES	Tuning 모델 평가에 사용할 지표	STRING	"roc_auc", "mean_squared_error"

- <https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-hyperparameter-tuning>

- (Optional) Hyper-parameter tuning (continued)

- 전체 쿼리 및 best model 확인

- <https://medium.com/google-cloud/hyperparameter-tuning-directly-within-bigquery-ml-a0affb0991ae>

```
CREATE OR REPLACE MODEL `card_data.card_model15`  
TRANSFORM(  
sex, marital_status, education,  
default_payment_next_month,  
limit_balance, age, avg_bill_amt, avg_pay_amt  
)
```

```
OPTIONS (
```

```
model_type = "BOOSTED_TREE_CLASSIFIER",  
num_trials = 5,  
max_tree_depth=hparam_range(3, 6),  
learn_rate=hparam_candidates([0, 0.1, 0.25, 0.4]),
```

```
auto_class_weights=TRUE,  
data_split_method="NO_SPLIT",  
input_label_cols=["default_payment_next_month"]  
) AS
```

```
SELECT
```

```
* EXCEPT(annotation)
```

```
FROM `card_data.card_view`
```

```
WHERE annotation = "training"
```

```
SELECT * FROM
```

```
ML.EVALUATE(MODEL
```

```
`card_data.card_model15`)
```

```
ORDER BY log_loss ASC
```

The screenshot displays the Google Cloud BigQuery ML interface. On the left, the 'Explorer' pane shows a tree view of pinned projects, with 'card_data' expanded and 'Models (7)' listed. 'card_model15' is highlighted. On the right, the 'card_model15' details pane is open, showing the 'EVALUATION' tab. It contains a table titled 'ALL TRIALS' SUMMARY' with columns: Trial ID, Learn rate, Max tree depth, and Precision. Trial 5 is marked as optimal.

Trial ID	Learn rate	Max tree depth	Precision
1	0.2500	6.0000	0.4809
2	0.2500	5.0000	0.4045
3	0.1000	6.0000	0.4304
4	0.4000	5.0000	0.4253
5 (Optimal)	0.4000	6.0000	0.6536

- (Optional) Hyper-parameter tuning (continued)
 - Prediction의 경우 자동으로 best model을 사용해서 prediction함

```
SELECT * FROM  
ML.PREDICT(MODEL `card_data.card_model15`, (  
  SELECT * FROM `card_data.card_view`  
  WHERE annotation="prediction"  
)  
)
```

Query results			
JOB INFORMATION		RESULTS	JSON
EXECUTION DETAILS			
Row	trial_id	predicted_default_payment_next_month	predicted_default_payment_next_month_probs
1	5	0	{ "label": "1", "prob": "0.2036820650100708" }, { "label": "0", "prob": "0.796317994594574" }
2	5	0	{ "label": "1", "prob": "0.41474077105522156" }, { "label": "0", "prob": "0.585259199142456" }
3	5	0	{ "label": "1", "prob": "0.32745927572250366" }, { "label": "0", "prob": "0.67254072427749634" }
4	5	0	{ "label": "1", "prob": "0.02840907871723175" }, { "label": "0", "prob": "0.97159087657928467" }
5	5	1	{ "label": "1", "prob": "0.652901291847229" }, { "label": "0", "prob": "0.34709867835044861" }
6	5	1	{ "label": "1", "prob": "0.575492262840271" }, { "label": "0", "prob": "0.42450776696205139" }

- 느낀 점
 - 왜 쓰지...
 - 데이터 찾는 게 은근히 까다롭다...
 - 데이터 찾는 팁(?)
 - 텍스트 적고 숫자 많은 데이터
 - **Row** 수가 꽤 많은 데이터 (최소 3000개 이상)
 - ID 등 뭔가 임의의 식별자가 포함된 데이터 → **Training - Validation - Prediction** 나누기 쉬움
 - 검색해보면서 훑어본 거지만 뭔가 **trip**에 관련된 데이터나 **cms_medicare** 안에 들어 있는 데이터가 해볼 만해 보였습니다...