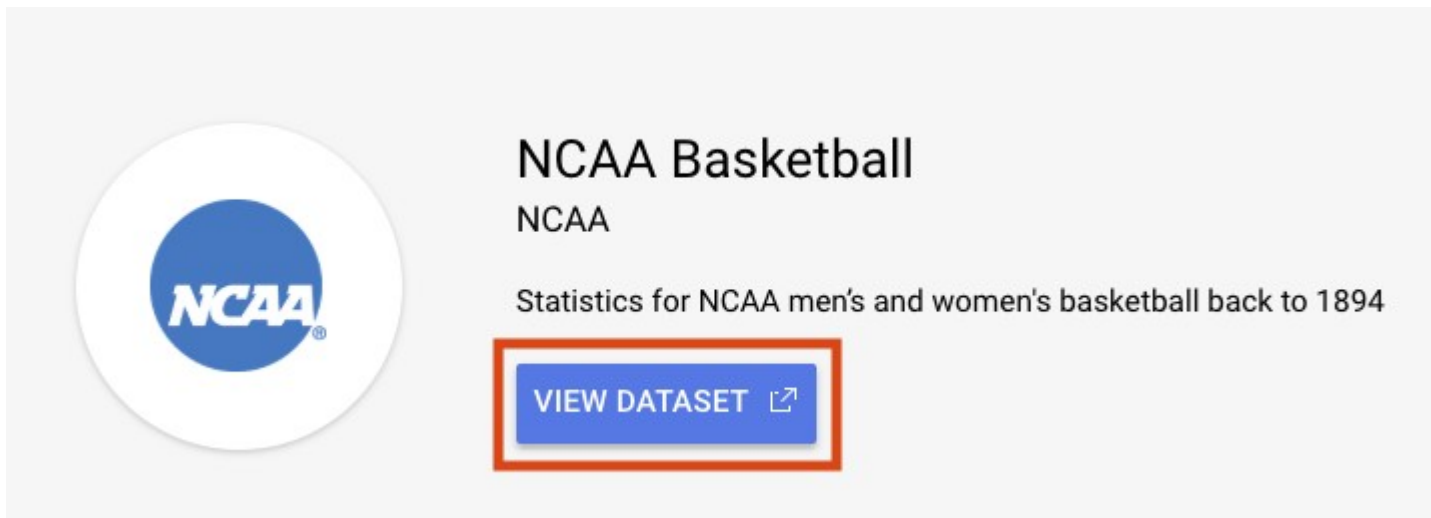


Find the NCAA public dataset in BigQuery

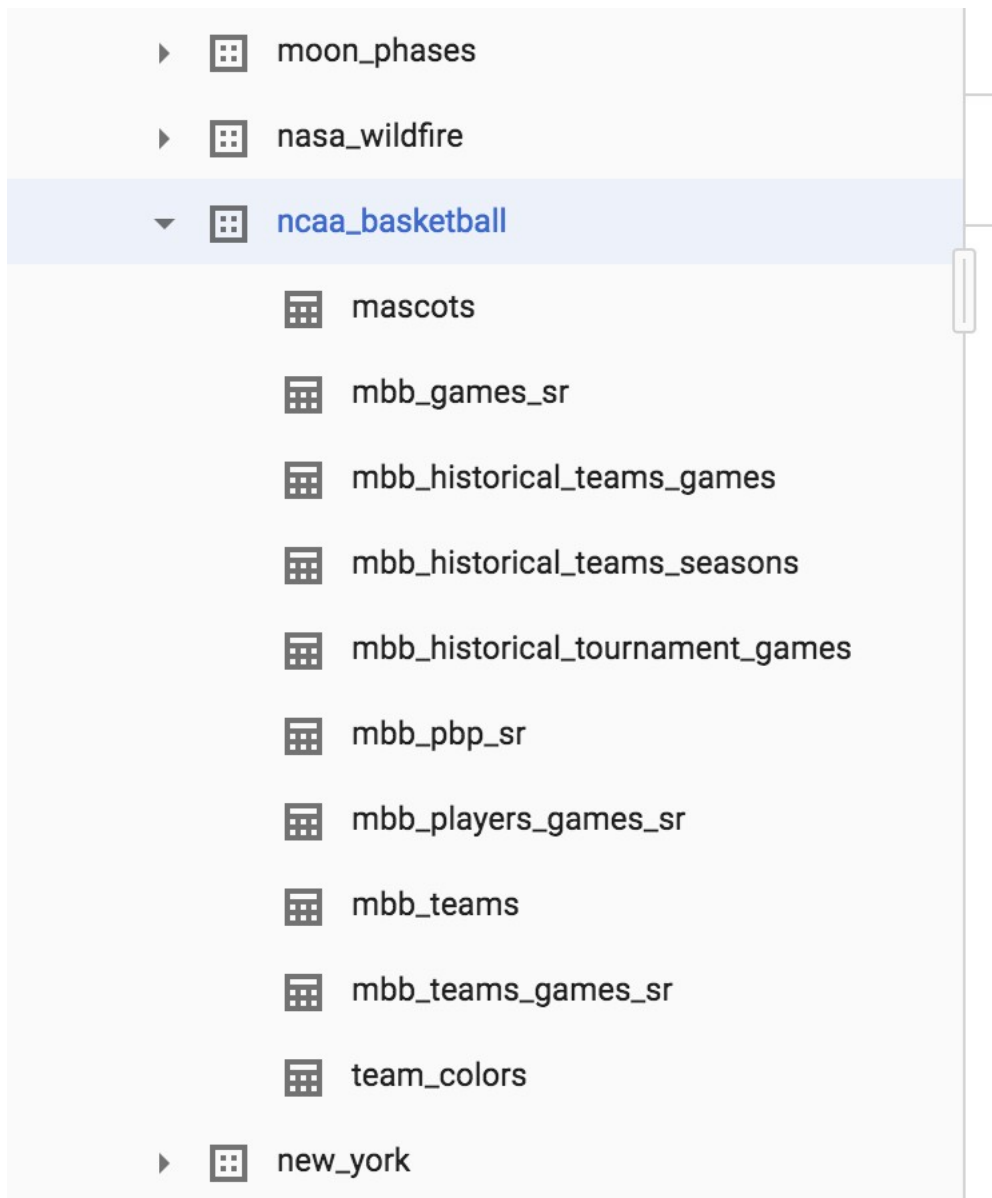
Make sure that you are still in the BigQuery Console for this step. In the Explorer tab, click the + **ADD DATA** button then select **Explore public datasets**.

In the search bar, type in "NCAA Basketball" and hit enter. One result will pop up—select it and then click **VIEW DATASET**:



This will open a new BigQuery tab with the dataset loaded. You can continue working in this tab, or close it and refresh your BigQuery Console in the other tab to reveal your public dataset.

Expand **bigquery-public-data** > **ncaa\_basketball** dataset to reveal its tables:



You should see 10 tables in the dataset. Click on the `mbb_historical_tournament_games` and then click **PREVIEW** to see sample rows of data. Then click **DETAILS** to get metadata about the table. Your page should resemble the following:

# Write a query to determine available seasons and games

You will now write a simple SQL query to determine how many seasons and games are available to explore in our `mbb_historical_tournament_games` table.

In query EDITOR, which is located above the table details section, copy and paste the following into that field:

```
SELECT
  season,
  COUNT(*) as games_per_tournament
FROM
  `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games`
GROUP BY season
ORDER BY season # default is Ascending (low to high)
```

Copied!

Click **RUN**. Soon after, you should receive a similar output:

Scroll through the output and take note of the amount of seasons and the games played per season—you will use that information to answer the following questions. Additionally, you can quickly see how many rows were returned by looking in the lower right near the pagination arrows.

Click Check my progress to verify the objective.

## Test your understanding

The following multiple choice questions are used to reinforce your understanding of the concepts covered so far. Answer them to the best of your abilities.

# Understand machine learning features and labels

The end goal of this lab is to predict the winner of a given NCAA Men's basketball match-up using past knowledge of historical games. In machine learning, each column of data that will help us determine the outcome (win or loss for a tournament game) is called a feature.

The column of data that you are trying to predict is called the label. Machine learning models "learn" the association between features to predict the outcome of a label.

Examples of features for your historical dataset could be:

- Season
- Team name
- Opponent team name
- Team seed (ranking)
- Opponent team seed

The label you will be trying to predict for future games will be the game outcome—whether or not a team wins or loses.

## Test your understanding

The following multiple choice questions are used to reinforce your understanding of the concepts covered so far. Answer them to the best of your abilities.

# Create a labeled machine learning dataset

Building a machine learning model requires a lot of high-quality training data. Fortunately, our NCAA dataset is robust enough where we can rely upon it to build an effective model. Return to the BigQuery Console—you should have left off on the result of the query you ran.

From the left-hand menu, open the `mbb_historical_tournament_games` table by clicking on the table name. Then once it loads, click **PREVIEW**. Your page should resemble the following:

## Test your understanding

The following multiple choice questions are used to reinforce your understanding of the concepts covered so far. Answer them to the best of your abilities.

After inspecting the dataset, you'll notice that one row in the dataset has columns for both `win_market` and `lose_market`. You need to break the single game record into a record for each team so you can label each row as a "winner" or "loser".

In the query EDITOR, copy and paste the following query and then click **RUN**:

```
# create a row for the winning team
SELECT
  # features
  season, # ex: 2015 season has March 2016 tournament games
  round, # sweet 16
  days_from_epoch, # how old is the game
  game_date,
  day, # Friday
  'win' AS label, # our label
  win_seed AS seed, # ranking
  win_market AS market,
  win_name AS name,
  win_alias AS alias,
```

```

win_school_ncaa AS school_ncaa,
# win_pts AS points,
lose_seed AS opponent_seed, # ranking
lose_market AS opponent_market,
lose_name AS opponent_name,
lose_alias AS opponent_alias,
lose_school_ncaa AS opponent_school_ncaa
# lose_pts AS opponent_points
FROM `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games`
UNION ALL
# create a separate row for the losing team
SELECT
# features
season,
round,
days_from_epoch,
game_date,
day,
'loss' AS label, # our label
lose_seed AS seed, # ranking
lose_market AS market,
lose_name AS name,
lose_alias AS alias,
lose_school_ncaa AS school_ncaa,
# lose_pts AS points,
win_seed AS opponent_seed, # ranking
win_market AS opponent_market,
win_name AS opponent_name,
win_alias AS opponent_alias,
win_school_ncaa AS opponent_school_ncaa
# win_pts AS opponent_points
FROM
`bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games`
Copied!

```

You should receive the following output:

Click Check my progress to verify the objective.

Now that you know what features are available from the result, answer the following question to reinforce your understanding of the dataset.

This feature is only available at the END of the game and for future games we are making predictions before a game begins.

# Part 1: Create a machine learning model to predict the winner based on seed and team name

Now that we have explored our data, it's time to train a machine learning model. Using your best judgment, answer the question below to orient yourself with this section.

~~We should build a forecasting model to predict how many points a team will score in a particular game.~~

We should build a classification model to predict whether a team will win or lose a particular game.

## Choosing a model type

For this particular problem, you will be building a classification model. Since we have two classes, win or lose, it's also called a binary classification model. A team can either win or lose a match.

If you wanted to, after the lab, you could forecast the total number of points a team will score using a forecasting model but that isn't going to be our focus here.

An easy way to tell if you're forecasting or classifying is to look at the type of label (column) of data you are predicting:

- If it's a numeric column (like units sold or points in a game), you're doing forecasting
- If it's a string value you're doing classification (this row is either this in class or this other class)

- ... and If you have more than two classes (like win, lose, or tie) you are doing multi-class classification

Our classification model will be doing machine learning is with a widely used statistical model called [Logistic Regression](#). We need a model that generates a probability for each possible discrete label value, which in our case is either a 'win' or a 'loss'. Logistic regression is a good model type to start with for this purpose. The good news for you is that the ML model will do all the math and optimization for you during model training -- it's what computers are really good at!

**Note:** There are many other machine learning models that vary in complexity to perform classification tasks. One commonly used at Google is [Deep Learning](#) with [neural networks](#). You can learn the TensorFlow code needed to build neural networks in this [Data Science](#) quest.

## Creating a machine learning model with BigQuery ML

To create our classification model in BigQuery we simply need to write the SQL statement CREATE MODEL and provide a few options.

But, before we can create the model, we need a place in our project to store it first.

In the Explorer tab, click on **View actions** icon next to your GCP Project ID and select **Create dataset**.

This will open a Create dataset dialog. Set your Dataset ID to bracketology and click **CREATE DATASET**.

Now run the following command in the query EDITOR

```
CREATE OR REPLACE MODEL  
  `bracketology.ncaa_model`
```



```

OPTIONS
  ( model_type='logistic_reg') AS
# create a row for the winning team
SELECT
  # features
  season,
  'win' AS label, # our label
  win_seed AS seed, # ranking
  win_school_ncaa AS school_ncaa,
  lose_seed AS opponent_seed, # ranking
  lose_school_ncaa AS opponent_school_ncaa
FROM `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games`
WHERE season <= 2017
UNION ALL
# create a separate row for the losing team
SELECT
  # features
  season,
  'loss' AS label, # our label
  lose_seed AS seed, # ranking
  lose_school_ncaa AS school_ncaa,
  win_seed AS opponent_seed, # ranking
  win_school_ncaa AS opponent_school_ncaa
FROM
  `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games`
# now we split our dataset with a WHERE clause so we can train on a subset of data and
then evaluate and test the model's performance against a reserved subset so the model
doesn't memorize or overfit to the training data.
# tournament season information from 1985 - 2017
# here we'll train on 1985 - 2017 and predict for 2018
WHERE season <= 2017
Copied!

```

In our code you'll notice that creating the model is just a few lines of SQL. One of the most important options is choosing the model type `logistic_reg` for our classification task.

[available model options](#) and settings. In our case we already have a field named "label" so we avoid having to specify our label column by using the model option: `input_label_cols`.

It will take between 3 and 5 minutes to train the model. You should receive the following output when the job is finished:

## Query results

---

Query complete (30.5 sec elapsed, 256.2 KB (ML) processed)

Job information   **Results**   JSON

---

✓ This statement created a new model named `qwiklabs-gcp-d4850eea17053489:bracketology.ncaa_model`.

---

Click the **Go to Model** button on the right-hand side of the console.

Click Check my progress to verify the objective.

## View model training details

Now that you are in the model details, scroll down to the **Training options** section and see the actual iterations the model performed to train. If you're experienced with machine learning, note that you can customize all of these hyperparameters (options set before the model runs) by defining their value in the `OPTIONS` statement. If you're new to machine learning, BigQuery ML will set smart default values for any options not set.

Refer to the [BigQuery ML model options list](#) to learn more.

## View model training stats

Machine learning models "learn" the association between known features and unknown labels. As you might intuitively guess, some features like "ranking seed" or "school name" may help determine a win or loss more than other data

columns (features) like the day of the week the game is played. Machine learning models start the training process with no such intuition and will generally randomize the weighting of each feature.

During the training process, the model will optimize a path to it's best possible weighting of each feature. With each run it is trying to minimize **Training Data Loss** and **Evaluation Data Loss**. Should you ever find that the final loss for evaluation is much higher than for training, your model is [overfitting](#) or memorizing your training data instead of learning generalizable relationships.

You can view how many training runs the model tooks by clicking the **TRAINING** tab and selecting **Table** under View as option. During our particular run, the model completed 3 training iterations in about 20 seconds. Yours will likely vary.

---

ncaa\_model

---

Model details   Model stats   Model schema

Iteration	Training Data Loss	Evaluation Data Loss	Learn Rate	Completion Time (seconds)
3	0.5038	0.5780	1.6000	2.73
2	0.5504	0.5930	0.8000	4.40
1	0.6089	0.6299	0.4000	4.79
0	0.6595	0.6671	0.2000	4.25

## See what the model learned about our features

After training, you can see which features provided the most value to the model by inspecting the weights. Run the following command in the query EDITOR:

```
SELECT
  category,
  weight
FROM
  UNNEST((
    SELECT
      category_weights
    FROM
      ML.WEIGHTS(MODEL `bracketology.ncaa_model`)
    WHERE
      processed_input = 'seed')) # try other features like 'school_ncaa'
  ORDER BY weight DESC
```

Copied!

Your output should resemble the following:

## Query results

 [SAVE RESULTS](#) ▼

Query complete (0.4 sec elapsed, 24.1 KB processed)

Job information [Results](#) JSON Execution details

Row	category	weight
1	01	0.5852092780176698
2	02	0.39297195138277996
3	03	0.25101465500186865
4	04	0.0715359016028045
5	06	0.057783806090708136
6	05	0.002518522994103156
7	07	-0.042499159644442154
8	08	-0.15113534855133096
9	11	-0.20631599390218447
10	10	-0.2129242010645865

As you can see, if the [seed](#) of a team is very low (1,2,3) or very high (14,15,16) the model gives it a significant weight (max is 1.0) in determining the win loss outcome. Intuitively this makes sense as we expect very low seed teams to perform well in the tournament.

The real magic of machine learning is that we didn't create a ton of hardcoded IF THEN statements in SQL telling the model IF the seed is 1 THEN give the team a 80% more chance of winning. Machine learning does away with hardcoded rules and logic and learns these relationships for itself. Check out the [BQML syntax weights documentation](#) to learn more.

# Evaluate model performance

To evaluate the model's performance you can run a simple `ML.EVALUATE` against a trained model. Run the following command in the query EDITOR:

```
SELECT
  *
FROM
  ML.EVALUATE(MODEL `bracketology.ncaa_model`)
```

Copied!

You should receive a similar output:

Query complete (0.4 sec elapsed, 0 B processed)

Job information [Results](#) JSON Execution details

Row	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	0.7176165803108808	0.6626794258373205	0.6924969249692496	0.6890547263681592	0.5779562229570154	0.765165

The value will be around 69% accurate. While it's better than a coin flip, there is room for improvement.

**Note:** For classification models, model accuracy isn't the only metric in the output you should care about. Because you performed a logistic regression, you can evaluate your model performance against all of the following metrics (the closer to 1.0 the better) [Precision](#) : A metric for classification models. Precision identifies the frequency with which a model was correct when predicting the positive class. [Recall](#) : A metric for classification models that answers the following question: Out of all the possible positive labels, how many did the model correctly identify? [Accuracy](#) : Accuracy is the fraction of predictions that a classification model got right. [f1\\_score](#) : A measure of the accuracy of the model. The f1 score is the harmonic average of the precision and recall. An f1 score's best value is 1. The worst value is 0. [log\\_loss](#) : The loss function used in a logistic regression. This is the measure of how far the model's predictions are from the correct labels. [roc\\_auc](#) : The area under the ROC curve. This is the probability that a classifier is more confident that a randomly chosen positive example is actually positive than that a randomly chosen negative example is positive.

# Making predictions

Now that you trained a model on historical data up to and including the 2017 season (which was all the data you had), it's time to make predictions for the 2018 season. Your data science team has just provided you with the tournament results for the 2018 tournament in a separate table which you don't have in your original dataset.

Making predictions is as simple as calling `ML.PREDICT` on a trained model and passing through the dataset you want to predict on.


Run the following command in the query EDITOR:

```
CREATE OR REPLACE TABLE `bracketology.predictions` AS (  
  SELECT * FROM ML.PREDICT(MODEL `bracketology.ncaa_model`,  
    # predicting for 2018 tournament games (2017 season)  
    (SELECT * FROM `data-to-insights.ncaa.2018_tournament_results`)  
  )  
)
```

Copied!

You should receive a similar output soon after:

Click [Check my progress](#) to verify the objective.

 **te:** You're storing your predictions in a table so you can query for insights later without having to keep re-running the above query.

You will now see your original dataset plus the addition of three new columns:

- Predicted label
- Predicted label options
- Predicted label probability

Since you happen to know the results of the 2018 March Madness tournament, let's see how the model did with it's predictions. (Tip: If you're predicting for this year's March Madness tournament, you would simply pass in a dataset with

2019 seeds and team names. Naturally, the label column will be empty as those games haven't been played yet -- that's what you're predicting!).

## How many did our model get right for the 2018 NCAA tournament?

Run the following command in the query EDITOR:

```
SELECT * FROM `bracketology.predictions`  
WHERE predicted_label <> label  
Copied!
```

You should receive a similar output:

Out of 134 predictions (67 March tournament games), our model got it wrong 38 times. 70% overall for the 2018 tournament matchup.



# Models can only take you so far...

There are many other factors and features that go into the close wins and amazing upsets of any March Madness tournament that a model would have a very hard time predicting.

Let's find biggest upset for the 2017 tournament according to the model. We'll look where the model predicts with 80%+ confidence and gets it WRONG.

Run the following command in the query EDITOR:

```
SELECT
  model.label AS predicted_label,
  model.prob AS confidence,
  predictions.label AS correct_label,
  game_date,
  round,
  seed,
  school_ncaa,
  points,
  opponent_seed,
  opponent_school_ncaa,
  opponent_points
FROM `bracketology.predictions` AS predictions,
UNNEST(predicted_label_probs) AS model
WHERE model.prob > .8 AND predicted_label <> predictions.label
Copied!
```

The outcome should look like this:

Query complete (0.3 sec elapsed, 12.5 KB processed)

Job information [Results](#) JSON Execution details

Row	predicted_label	confidence	correct_label	game_date	round	seed	school_ncaa	points	opponent_seed	opponent_school_ncaa	opponent_points
1	win	0.8647366355980848	loss	2018-03-15	64	04	Arizona	68	13	Buffalo	89
2	win	0.8528053004359508	loss	2018-03-16	64	04	Wichita St.	75	13	Marshall	81
3	win	0.8981839119836009	loss	2018-03-16	64	01	Virginia	54	16	UMBC	74
4	loss	0.8895132514511247	win	2018-03-15	64	13	Buffalo	89	04	Arizona	68
5	loss	0.8701799722994598	win	2018-03-16	64	13	Marshall	81	04	Wichita St.	75
6	loss	0.8744298173994041	win	2018-03-16	64	16	UMBC	74	01	Virginia	54
7	loss	0.8058240185589368	win	2018-03-18	32	05	Clemson	84	04	Auburn	53

**Prediction:** The model predicts Seed 1 Virginia to beat Seed 16 UMBC with 87% confidence. Seems reasonable right?

Take a look at [this video](#) to see what actually happened!

Coach Odom (UMBC) after the [game](#) said, "Unbelievable — it's really all you can say."

## Recap

- You created a machine learning model to predict game outcome.
- You evaluated the performance and got to 69% accuracy using seed and team name as primary features
- You predicted 2018 tournament outcomes
- You analyzed the results for insights

Our next challenge will be to build a better model WITHOUT using seed and team name as features.

## Part 2: Using skillful ML model features

In the second part of this lab you will be building a second ML model using newly provided and detailed features.

Now that you're familiar with building ML models using BigQuery ML, your data science team has provided you with a new play-by-play dataset where they have created new team metrics for your model to learn from. These include:

- Scoring efficiency over time based on historical play-by-play analysis.
- Possession of the basketball over time.

### Create a new ML dataset with these skillful features

Run the following command in the query EDITOR:

```
# create training dataset:
# create a row for the winning team
CREATE OR REPLACE TABLE `bracketology.training_new_features` AS
WITH outcomes AS (
SELECT
  # features
  season, # 1994
  'win' AS label, # our label
  win_seed AS seed, # ranking # this time without seed even
  win_school_ncaa AS school_ncaa,
  lose_seed AS opponent_seed, # ranking
  lose_school_ncaa AS opponent_school_ncaa
FROM `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games` t
WHERE season >= 2014
UNION ALL
# create a separate row for the losing team
SELECT
  # features
  season, # 1994
  'loss' AS label, # our label
  lose_seed AS seed, # ranking
  lose_school_ncaa AS school_ncaa,
  win_seed AS opponent_seed, # ranking
  win_school_ncaa AS opponent_school_ncaa
FROM
  `bigquery-public-data.ncaa_basketball.mbb_historical_tournament_games` t
WHERE season >= 2014
```

```

UNION ALL
# add in 2018 tournament game results not part of the public dataset:
SELECT
    season,
    label,
    seed,
    school_ncaa,
    opponent_seed,
    opponent_school_ncaa
FROM
    `data-to-insights.ncaa.2018_tournament_results`
)
SELECT
o.season,
label,
# our team
seed,
school_ncaa,
# new pace metrics (basketball possession)
team.pace_rank,
team.poss_40min,
team.pace_rating,
# new efficiency metrics (scoring over time)
team.efficiency_rank,
team.pts_100poss,
team.efficiency_rating,
# opposing team
opponent_seed,
opponent_school_ncaa,
# new pace metrics (basketball possession)
opp.pace_rank AS opp_pace_rank,
opp.poss_40min AS opp_poss_40min,
opp.pace_rating AS opp_pace_rating,
# new efficiency metrics (scoring over time)
opp.efficiency_rank AS opp_efficiency_rank,
opp.pts_100poss AS opp_pts_100poss,
opp.efficiency_rating AS opp_efficiency_rating,
# a little feature engineering (take the difference in stats)
# new pace metrics (basketball possession)
opp.pace_rank - team.pace_rank AS pace_rank_diff,
opp.poss_40min - team.poss_40min AS pace_stat_diff,
opp.pace_rating - team.pace_rating AS pace_rating_diff,
# new efficiency metrics (scoring over time)
opp.efficiency_rank - team.efficiency_rank AS eff_rank_diff,
opp.pts_100poss - team.pts_100poss AS eff_stat_diff,
opp.efficiency_rating - team.efficiency_rating AS eff_rating_diff
FROM outcomes AS o
LEFT JOIN `data-to-insights.ncaa.feature_engineering` AS team
ON o.school_ncaa = team.team AND o.season = team.season
LEFT JOIN `data-to-insights.ncaa.feature_engineering` AS opp
ON o.opponent_school_ncaa = opp.team AND o.season = opp.season
Copied!

```

You should receive a similar output soon after:

Click **Check my progress** to verify the objective.

## Preview the new features

Click on the **Go to table** button on the right-hand side of the console. Then click on the **Preview** tab.

Your table should look similar to the following:

Don't worry if your output is not identical to the above screenshot.

## Interpreting selected metrics

You will now learn about some important labels that aid us in making predictions.

## opp\_efficiency\_rank

**Opponent's Efficiency Rank:** out of all the teams, what rank does our opponent have for scoring efficiently over time (points per 100 basketball possessions). Lower is better.

## opp\_pace\_rank

**Opponent's Pace Rank:** out of all teams, what rank does our opponent have for basketball possession (number of possessions in 40 minutes). Lower is better.

Now that you have insightful features on how well a team can score and how well it can hold on to the basketball lets train our second model.

As an additional measure to safe-guard your model from "memorizing good teams from the past", exclude the team's name and the seed from this next model and focus only on the metrics.

## Train the new model

Run the following command in the query EDITOR:

```
CREATE OR REPLACE MODEL
  `bracketology.ncaa_model_updated`
OPTIONS
  ( model_type='logistic_reg') AS
```

```

SELECT
  # this time, dont train the model on school name or seed
  season,
  label,
  # our pace
  poss_40min,
  pace_rank,
  pace_rating,
  # opponent pace
  opp_poss_40min,
  opp_pace_rank,
  opp_pace_rating,
  # difference in pace
  pace_rank_diff,
  pace_stat_diff,
  pace_rating_diff,
  # our efficiency
  pts_100poss,
  efficiency_rank,
  efficiency_rating,
  # opponent efficiency
  opp_pts_100poss,
  opp_efficiency_rank,
  opp_efficiency_rating,
  # difference in efficiency
  eff_rank_diff,
  eff_stat_diff,
  eff_rating_diff
FROM `bracketology.training_new_features`
# here we'll train on 2014 - 2017 and predict on 2018
WHERE season BETWEEN 2014 AND 2017 # between in SQL is inclusive of end points
Copied!

```

Soon after, your output should look similar to the following:

## Evaluate the new model's performance

To evaluate your model's performance, run the following command in the query EDITOR:

```
SELECT
*
FROM
ML.EVALUATE(MODEL `bracketology.ncaa_model_updated` )
Copied!
```

Your output should look similar to the followings:

Query complete (0.5 sec elapsed, 0 B processed)

Job information [Results](#) JSON Execution details

Row	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	0.7547169811320755	0.8	0.7653061224489796	0.7766990291262136	0.5529520315494507	0.796202

Wow! You just trained a new model with different features and increased your accuracy to around 75% or a 5% lift from the original model.

This is one of the biggest lessons learned in machine learning is that your high quality feature dataset can make a huge difference in how accurate your model is.

Click [Check my progress](#) to verify the objective.

## Inspect what the model learned

Which features does the model weigh the most in win / loss outcome? Find out by running the following command in the query EDITOR:

```
SELECT
*
```



```
FROM  
  ML.WEIGHTS(MODEL    `bracketology.ncaa_model_updated`)  
ORDER BY ABS(weight) DESC  
Copied!
```

Your output should look like this:

We've taken the absolute value of the weights in our ordering so the most impactful (for a win or a loss) are listed first.

As you can see in the results, the top 3 are `pace_stat_diff`, `eff_stat_diff`, and `eff_rating_diff`. Let's explore these a little bit more.

## `pace_stat_diff`

How different the actual stat for (possessions / 40 minutes) was between teams. According to the model, this is the largest driver in choosing the game outcome.

## `eff_stat_diff`

How different the actual stat for (net points / 100 possessions) was between teams.

## `eff_rating_diff`

How different the normalized rating for scoring efficiency was between teams.

What did the model not weigh heavily in its predictions? Season. It was last in the output of ordered weights above. What the model is saying is that the season (2013, 2014, 2015) isn't that useful in predicting game outcome. There wasn't anything magical about the year "2014" for all teams.

An interesting insight is that the model valued the pace of a team (how well they can control the ball) over how efficiently a team can score.

## Prediction time!

Run the following command in the query EDITOR:

```
CREATE OR REPLACE TABLE `bracketology.ncaa_2018_predictions` AS
# let's add back our other data columns for context
SELECT
  *
FROM
  ML.PREDICT(MODEL `bracketology.ncaa_model_updated`, (
  SELECT
    * # include all columns now (the model has already been trained)
  FROM `bracketology.training_new_features`
  WHERE season = 2018
  ))
```

Copied!

Your output should be similar to the following:

Query complete (1.6 sec elapsed, 97.2 KB processed)

Job information [Results](#) JSON Execution details

✓ This statement created a new table named `qwiklabs-gcp-220e4ec0b4fa40bc:bracketology.ncaa_2017_predictions`.

Click [Check my progress](#) to verify the objective.

## Prediction analysis:

Since you know the correct game outcome, you can see where your model made an incorrect prediction using the new testing dataset. Run the following command in the query EDITOR:

```
SELECT * FROM `bracketology.ncaa_2018_predictions`  
WHERE predicted_label <> label  
Copied!
```

As you can see from the number of records returned from the query, the model got 48 matchups wrong (24 games) out of the total number of matchups in the tournament for a 2018 accuracy of 64%. 2018 must have been a wild year, let's look at what upsets happened.

## Where were the upsets in March 2018?

Run the following command in the query EDITOR:

```
SELECT
```

```

CONCAT(school_ncaa, " was predicted to ",IF(predicted_label="loss","lose","win"),"
",CAST(ROUND(p.prob,2)*100 AS STRING), "% but ", IF(n.label="loss","lost","won")) AS
narrative,
predicted_label, # what the model thought
n.label, # what actually happened
ROUND(p.prob,2) AS probability,
season,
# us
seed,
school_ncaa,
pace_rank,
efficiency_rank,
# them
opponent_seed,
opponent_school_ncaa,
opp_pace_rank,
opp_efficiency_rank
FROM `bracketology.ncaa_2018_predictions` AS n,
UNNEST(predicted_label_probs) AS p
WHERE
    predicted_label <> n.label # model got it wrong
    AND p.prob > .75 # by more than 75% confidence
ORDER BY prob DESC
Copied!

```

Your outcome should look like:

The major upset was the same one our previous model found: UMBC vs Virginia. 2018 overall was a year of [huge upsets](#). Will 2019 be just as wild?

## Comparing model performance

What about where the naive model (comparing seeds) got it wrong but the advanced model got it right? Run the following command in the query EDITOR:

```
SELECT
```

```

CONCAT(opponent_school_ncaa, " (", opponent_seed, ") was
",CAST(ROUND(ROUND(p.prob,2)*100,2) AS STRING),"% predicted to upset ", school_ncaa,
" (", seed, ") and did!") AS narrative,
predicted_label, # what the model thought
n.label, # what actually happened
ROUND(p.prob,2) AS probability,
season,
# us
seed,
school_ncaa,
pace_rank,
efficiency_rank,
# them
opponent_seed,
opponent_school_ncaa,
opp_pace_rank,
opp_efficiency_rank,
(CAST(opponent_seed AS INT64) - CAST(seed AS INT64)) AS seed_diff
FROM `bracketology.ncaa_2018_predictions` AS n,
UNNEST(predicted_label_probs) AS p
WHERE
    predicted_label = 'loss'
    AND predicted_label = n.label # model got it right
    AND p.prob >= .55 # by 55%+ confidence
    AND (CAST(opponent_seed AS INT64) - CAST(seed AS INT64)) > 2 # seed difference
magnitude
ORDER BY (CAST(opponent_seed AS INT64) - CAST(seed AS INT64)) DESC
Copied!

```

Your outcome should look like:

The model predicted a Florida St. (09) upset of Xavier (01) and they did!

The upset was correctly predicted by the new model (even when the seed ranking said otherwise) based on your new skillful features like pace and shooting efficiency. Watch the [game highlights on YouTube](#).

# Predicting for the 2019 March Madness tournament

Now that we know the teams and seed rankings for March 2019, let's predict the outcome of future games.

## Explore the 2019 data

Run the below query to find the top seeds

```
SELECT * FROM `data-to-insights.ncaa.2019_tournament_seeds` WHERE seed = 1
```

Copied!

Your outcome should look like:

Query complete (0.4 sec elapsed, 1.8 KB processed)

Job information [Results](#) JSON Execution details

Row	school_ncaa	seed	season
1	North Carolina	1	2019
2	Gonzaga	1	2019
3	Duke	1	2019
4	Virginia	1	2019

## Create a matrix of all possible games

Since we don't know which teams will play each other as the tournament progresses, we'll simply have them all face each other.

In SQL, an easy way to have a single team play every other team in a table is with a CROSS JOIN.

Run the below query to get all possible team games in the tournament.

```
SELECT
  NULL AS label,
  team.school_ncaa AS team_school_ncaa,
  team.seed AS team_seed,
  opp.school_ncaa AS opp_
Copied!
```

## Add in 2018 team stats (pace, efficiency)

```
CREATE OR REPLACE TABLE `bracketology.ncaa_2019_tournament` AS
WITH team_seeds_all_possible_games AS (
  SELECT
    NULL AS label,
    team.school_ncaa AS school_ncaa,
    team.seed AS seed,
    opp.school_ncaa AS opponent_school_ncaa,
    opp.seed AS opponent_seed
  FROM `data-to-insights.ncaa.2019_tournament_seeds` AS team
  CROSS JOIN `data-to-insights.ncaa.2019_tournament_seeds` AS opp
  # teams cannot play against themselves :)
  WHERE team.school_ncaa <> opp.school_ncaa
)
, add_in_2018_season_stats AS (
  SELECT
    team_seeds_all_possible_games.*,
    # bring in features from the 2018 regular season for each team
    (SELECT AS STRUCT * FROM `data-to-insights.ncaa.feature_engineering` WHERE
    school_ncaa = team AND season = 2018) AS team,
    (SELECT AS STRUCT * FROM `data-to-insights.ncaa.feature_engineering` WHERE
    opponent_school_ncaa = team AND season = 2018) AS opp
  FROM team_seeds_all_possible_games
)
# Preparing 2019 data for prediction
SELECT
  label,
```

```

2019 AS season, # 2018-2019 tournament season
# our team
seed,
school_ncaa,
# new pace metrics (basketball possession)
team.pace_rank,
team.poss_40min,
team.pace_rating,
# new efficiency metrics (scoring over time)
team.efficiency_rank,
team.pts_100poss,
team.efficiency_rating,
# opposing team
opponent_seed,
opponent_school_ncaa,
# new pace metrics (basketball possession)
opp.pace_rank AS opp_pace_rank,
opp.poss_40min AS opp_poss_40min,
opp.pace_rating AS opp_pace_rating,
# new efficiency metrics (scoring over time)
opp.efficiency_rank AS opp_efficiency_rank,
opp.pts_100poss AS opp_pts_100poss,
opp.efficiency_rating AS opp_efficiency_rating,
# a little feature engineering (take the difference in stats)
# new pace metrics (basketball possession)
opp.pace_rank - team.pace_rank AS pace_rank_diff,
opp.poss_40min - team.poss_40min AS pace_stat_diff,
opp.pace_rating - team.pace_rating AS pace_rating_diff,
# new efficiency metrics (scoring over time)
opp.efficiency_rank - team.efficiency_rank AS eff_rank_diff,
opp.pts_100poss - team.pts_100poss AS eff_stat_diff,
opp.efficiency_rating - team.efficiency_rating AS eff_rating_diff
FROM add_in_2018_season_stats
Copied!

```

## Make predictions

```

CREATE OR REPLACE TABLE `bracketology.ncaa_2019_tournament_predictions` AS
SELECT
*
FROM
# let's predicted using the newer model
ML.PREDICT(MODEL `bracketology.ncaa_model_updated`, (
# let's predict on March 2019 tournament games:
SELECT * FROM `bracketology.ncaa_2019_tournament`
))
Copied!

```

Click [Check my progress](#) to verify the objective.



# Get your predictions

```
SELECT
  p.label AS prediction,
  ROUND(p.prob,3) AS confidence,
  school_ncaa,
  seed,
  opponent_school_ncaa,
  opponent_seed
FROM `bracketology.ncaa_2019_tournament_predictions`,
UNNEST(predicted_label_probs) AS p
WHERE p.prob >= .5
AND school_ncaa = 'Duke'
ORDER BY seed, opponent_seed
Copied!
```

Query complete (0.7 sec elapsed, 289.7 KB processed)

Job information [Results](#) JSON Execution details

Row	prediction	confidence	school_ncaa	seed	opponent_school_ncaa	opponent_seed
1	win	0.605	Duke	1	North Carolina	1
2	win	0.515	Duke	1	Gonzaga	1
3	loss	0.694	Duke	1	Virginia	1
4	loss	0.585	Duke	1	Michigan	2
5	loss	0.542	Duke	1	Michigan St.	2
6	loss	0.53	Duke	1	Tennessee	2
7	loss	0.508	Duke	1	Kentucky	2
8	loss	0.554	Duke	1	Purdue	3
9	loss	0.517	Duke	1	Houston	3
10	loss	0.522	Duke	1	Texas Tech	3
11	win	0.619	Duke	1	LSU	3
12	loss	0.588	Duke	1	Virginia Tech	4
13	win	0.626	Duke	1	Florida St.	4
14	win	0.537	Duke	1	Kansas St.	4

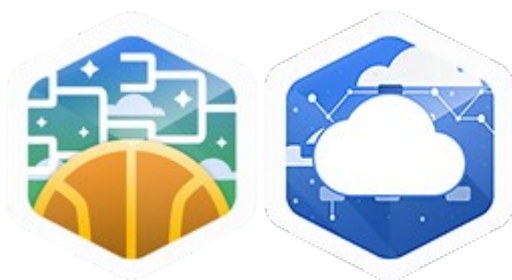
Here we filtered the model results to see all of Duke's possible games. Scroll to find the Duke vs North Dakota St. game.

Insight: Duke (1) is 88.5% favored to beat North Dakota St. (16) on 3/22/19.

Experiment by changing the school\_ncaa filter above to predict for the matchups in your bracket. Write down what the model confidence is and enjoy the games!

## Congratulations!

And there you have it! You used Machine Learning to predict winning teams for the NCAA Men's basketball tournament.



## Finish Your Quest

This self-paced lab is part of the Qwiklabs [NCAA® March Madness®: Bracketology with Google Cloud](#) and [BigQuery for BigQuery for Machine Learning](#) Quests. A Quest is a series of related labs that form a learning path.

Completing a Quest earns you a badge to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in either Quest and get immediate completion credit if you've taken this lab. [See other available Qwiklabs Quests](#).

## Take your next lab

We hope you enjoyed learning all about big data exploration, and how quickly you can create machine learning models inside of BigQuery. Try these out next:

- [Exploring NCAA Data with BigQuery](#)
- [Introduction to SQL for BigQuery and Cloud SQL](#)

## Next steps / learn more

- Want to learn more about basketball metrics and analysis? Check out [additional analysis](#) from team behind the Google Cloud NCAA tournament ads and predictions.
- See what you can do with ML with [Using Machine Learning on Compute Engine to Make Product Recommendations](#)
- [Data Science quest](#)

## Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. [Our classes](#) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level

training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](#) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated September 25, 2021

Lab Last Tested September 25, 2021

Copyright 2021 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.