# sample_babyweight

December 16, 2020

# 1 Creating a Sampled Dataset

### Learning Objectives

1. Setup up the environment
2. Sample the natality dataset to create train, eval, test sets
3. Preprocess the data in Pandas dataframe

## 1.1 Introduction

In this notebook, we'll read data from BigQuery into our notebook to preprocess the data within a Pandas dataframe for a small, repeatable sample.

We will set up the environment, sample the natality dataset to create train, eval, test splits, and preprocess the data in a Pandas dataframe.

Each learning objective will correspond to a **#TODO** in this student lab notebook – try to complete this notebook first and then review the solution notebook.

## 1.2 Set up environment variables and load necessary libraries

[1]:
```
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

[2]:
```
!pip install --user google-cloud-bigquery==1.25.0
```

```
Collecting google-cloud-bigquery==1.25.0
  Downloading google_cloud_bigquery-1.25.0-py2.py3-none-any.whl (169 kB)
    ␣
 ↪|âŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰŁâŰ
 ↪169 kB 8.3 MB/s eta 0:00:01
Requirement already satisfied: protobuf>=3.6.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(3.13.0)
Requirement already satisfied: google-auth<2.0dev,>=1.9.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(1.23.0)
```

```
Collecting google-resumable-media<0.6dev,>=0.5.0
  Downloading google_resumable_media-0.5.1-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: google-cloud-core<2.0dev,>=1.1.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(1.3.0)
Requirement already satisfied: six<2.0.0dev,>=1.13.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(1.15.0)
Requirement already satisfied: google-api-core<2.0dev,>=1.15.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(1.22.4)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from protobuf>=3.6.0->google-cloud-bigquery==1.25.0) (50.3.2)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-
cloud-bigquery==1.25.0) (4.1.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/opt/conda/lib/python3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-
cloud-bigquery==1.25.0) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.5" in
/opt/conda/lib/python3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-
cloud-bigquery==1.25.0) (4.6)
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in
/opt/conda/lib/python3.7/site-packages (from google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (1.52.0)
Requirement already satisfied: pytz in /opt/conda/lib/python3.7/site-packages
(from google-api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2020.4)
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in
/opt/conda/lib/python3.7/site-packages (from google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2.24.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/opt/conda/lib/python3.7/site-packages (from pyasn1-modules>=0.2.1->google-
auth<2.0dev,>=1.9.0->google-cloud-bigquery==1.25.0) (0.4.8)
Requirement already satisfied: chardet<4,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0dev,>=2.18.0->google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2020.11.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (1.25.11)
Installing collected packages: google-resumable-media, google-cloud-bigquery
```

```
ERROR: After October 2020 you may experience errors when installing or
updating packages. This is because pip will change the way that it resolves
dependency conflicts.
We recommend you use --use-feature=2020-resolver to test your packages with the
new resolver before it becomes the default.
tfx 0.23.0 requires attrs<20,>=19.3.0, but you'll have attrs 20.3.0 which is
incompatible.
tfx 0.23.0 requires google-resumable-media<0.7.0,>=0.6.0, but you'll have
google-resumable-media 0.5.1 which is incompatible.
tfx 0.23.0 requires kubernetes<12,>=10.0.1, but you'll have kubernetes 12.0.0
which is incompatible.
tfx 0.23.0 requires pyarrow<0.18,>=0.17, but you'll have pyarrow 2.0.0 which is
incompatible.
google-cloud-storage 1.30.0 requires google-resumable-media<2.0dev,>=0.6.0, but
you'll have google-resumable-media 0.5.1 which is incompatible.
Successfully installed google-cloud-bigquery-1.25.0 google-resumable-media-0.5.1
```

**Note**: Restart your kernel to use updated packages.

Kindly ignore the deprecation warnings and incompatibility errors related to google-cloud-storage.

Import necessary libraries.

[1]:
```
from google.cloud import bigquery
import pandas as pd
```

**Lab Task #1:** Set up environment variables so that we can use them throughout the notebook

[2]:
```
%%bash
# TODO 1
# TODO -- Your code here.
echo "Your current GCP Project Name is: "$PROJECT
```

```
Your current GCP Project Name is:
```

[3]:
```
PROJECT = "qwiklabs-gcp-03-5d3dc033c852"  # Replace with your PROJECT
```

## 1.3 Create ML datasets by sampling using BigQuery

We'll begin by sampling the BigQuery data to create smaller datasets. Let's create a BigQuery client that we'll use throughout the lab.

[4]:
```
bq = bigquery.Client(project = PROJECT)
```

We need to figure out the right way to divide our hash values to get our desired splits. To do that we need to define some values to hash within the module. Feel free to play around with these values to get the perfect combination.

[5]:
```
modulo_divisor = 100
train_percent = 80.0
eval_percent = 10.0

train_buckets = int(modulo_divisor * train_percent / 100.0)
eval_buckets = int(modulo_divisor * eval_percent / 100.0)
```

We can make a series of queries to check if our bucketing values result in the correct sizes of each of our dataset splits and then adjust accordingly. Therefore, to make our code more compact and reusable, let's define a function to return the head of a dataframe produced from our queries up to a certain number of rows.

[6]:
```
def display_dataframe_head_from_query(query, count=10):
    """Displays count rows from dataframe head from query.

    Args:
        query: str, query to be run on BigQuery, results stored in dataframe.
        count: int, number of results from head of dataframe to display.
    Returns:
        Dataframe head with count number of results.
    """
    df = bq.query(
        query + " LIMIT {limit}".format(
            limit=count)).to_dataframe()

    return df.head(count)
```

For our first query, we're going to use the original query above to get our label, features, and columns to combine into our hash which we will use to perform our repeatable splitting. There are only a limited number of years, months, days, and states in the dataset. Let's see what the hash values are. We will need to include all of these extra columns to hash on to get a fairly uniform spread of the data. Feel free to try less or more in the hash and see how it changes your results.

[7]:
```
# Get label, features, and columns to hash and split into buckets
hash_cols_fixed_query = """
SELECT
    weight_pounds,
```

```
    is_male,
    mother_age,
    plurality,
    gestation_weeks,
    year,
    month,
    CASE
        WHEN day IS NULL THEN
            CASE
                WHEN wday IS NULL THEN 0
                ELSE wday
            END
        ELSE day
    END AS date,
    IFNULL(state, "Unknown") AS state,
    IFNULL(mother_birth_state, "Unknown") AS mother_birth_state
FROM
    publicdata.samples.natality
WHERE
    year > 2000
    AND weight_pounds > 0
    AND mother_age > 0
    AND plurality > 0
    AND gestation_weeks > 0
"""

display_dataframe_head_from_query(hash_cols_fixed_query)
```

[7]:

```
   weight_pounds  is_male  mother_age  plurality  gestation_weeks  year  \
0       7.063611     True          32          1               37  2001
1       4.687028     True          30          3               33  2001
2       7.561856     True          20          1               39  2001
3       7.561856     True          31          1               37  2001
4       7.312733     True          32          1               40  2001
5       7.627994    False          30          1               40  2001
6       7.251004     True          33          1               37  2001
7       7.500126    False          23          1               39  2001
8       7.125340    False          33          1               39  2001
9       7.749249     True          31          1               39  2001

   month  date state mother_birth_state
0     12     3    CO                 CA
1      6     5    IN                 IN
2      4     5    MN                 MN
3     10     5    MS                 MS
4     11     3    MO                 MO
5     10     5    NY                 PA
```

```
6     11     5    WA              WA
7      9     2    OK              LA
8      1     4    TX              MS
9      1     1    TX         Foreign
```

Using COALESCE would provide the same result as the nested CASE WHEN. This is preferable when all we want is the first non-null instance. To be precise the CASE WHEN would become COALESCE(wday, day, 0) AS date. You can read more about it here.

Next query will combine our hash columns and will leave us just with our label, features, and our hash values.

[8]:
```
data_query = """
SELECT
    weight_pounds,
    is_male,
    mother_age,
    plurality,
    gestation_weeks,
    FARM_FINGERPRINT(
        CONCAT(
            CAST(year AS STRING),
            CAST(month AS STRING),
            CAST(date AS STRING),
            CAST(state AS STRING),
            CAST(mother_birth_state AS STRING)
        )
    ) AS hash_values
FROM
    ({CTE_hash_cols_fixed})
""".format(CTE_hash_cols_fixed=hash_cols_fixed_query)

display_dataframe_head_from_query(data_query)
```

[8]:
```
   weight_pounds  is_male  mother_age  plurality  gestation_weeks  \
0       7.063611     True          32          1               37
1       4.687028     True          30          3               33
2       7.561856     True          20          1               39
3       7.561856     True          31          1               37
4       7.312733     True          32          1               40
5       7.627994    False          30          1               40
6       7.251004     True          33          1               37
7       7.500126    False          23          1               39
8       7.125340    False          33          1               39
9       7.749249     True          31          1               39

           hash_values
0  4762325092919148672
```

```
1  2341060194216507348
2 -8842767231851202242
3  7957807816914159435
4 -5961624242430066305
5  5493295634082918412
6 -2988893757655690534
7 -6735199252008114417
8 -3514093303120687641
9  2175328516857391398
```

The next query is going to find the counts of each of the unique 657484 `hash_values`. This will be our first step at making actual hash buckets for our split via the `GROUP BY`.

[9]:
```
# Get the counts of each of the unique hash of our splitting column
first_bucketing_query = """
SELECT
    hash_values,
    COUNT(*) AS num_records
FROM
    ({CTE_data})
GROUP BY
    hash_values
""".format(CTE_data=data_query)

display_dataframe_head_from_query(first_bucketing_query)
```

[9]:

| | hash_values | num_records |
|---|---|---|
| 0 | -1700820252994836306 | 741 |
| 1 | 7948408306271784936 | 883 |
| 2 | -1740303207227716653 | 794 |
| 3 | 5160546322234461939 | 2887 |
| 4 | -6766011436514751671 | 178 |
| 5 | -3974717950920322290 | 78 |
| 6 | -8405995084719745013 | 2 |
| 7 | 8641800065663952690 | 875 |
| 8 | -2290255568977475467 | 212 |
| 9 | -646460941575642964 | 463 |

The query below performs a second layer of bucketing where now for each of these bucket indices we count the number of records.

[10]:
```
# Get the number of records in each of the hash buckets
second_bucketing_query = """
SELECT
    ABS(MOD(hash_values, {modulo_divisor})) AS bucket_index,
    SUM(num_records) AS num_records
FROM
    ({CTE_first_bucketing})
```

```
GROUP BY
    ABS(MOD(hash_values, {modulo_divisor}))
""".format(
    CTE_first_bucketing=first_bucketing_query, modulo_divisor=modulo_divisor)

display_dataframe_head_from_query(second_bucketing_query)
```

[10]:

| | bucket_index | num_records |
|---|---|---|
| 0 | 62 | 426834 |
| 1 | 46 | 281627 |
| 2 | 76 | 354090 |
| 3 | 87 | 523881 |
| 4 | 0 | 277395 |
| 5 | 63 | 355283 |
| 6 | 58 | 209618 |
| 7 | 66 | 402627 |
| 8 | 34 | 379000 |
| 9 | 56 | 226752 |

The number of records is hard for us to easily understand the split, so we will normalize the count into percentage of the data in each of the hash buckets in the next query.

[11]:

```
# Calculate the overall percentages
percentages_query = """
SELECT
    bucket_index,
    num_records,
    CAST(num_records AS FLOAT64) / (
    SELECT
        SUM(num_records)
    FROM
        ({CTE_second_bucketing})) AS percent_records
FROM
    ({CTE_second_bucketing})
""".format(CTE_second_bucketing=second_bucketing_query)

display_dataframe_head_from_query(percentages_query)
```

[11]:

| | bucket_index | num_records | percent_records |
|---|---|---|---|
| 0 | 70 | 285539 | 0.008650 |
| 1 | 91 | 333267 | 0.010096 |
| 2 | 78 | 326758 | 0.009898 |
| 3 | 0 | 277395 | 0.008403 |
| 4 | 4 | 398118 | 0.012060 |
| 5 | 9 | 236637 | 0.007168 |
| 6 | 33 | 410226 | 0.012427 |
| 7 | 6 | 548778 | 0.016624 |

| | | | |
|---|---|---|---|
| 8 | 84 | 341155 | 0.010334 |
| 9 | 38 | 338150 | 0.010243 |

We'll now select the range of buckets to be used in training.

[12]:
```
# Choose hash buckets for training and pull in their statistics
train_query = """
SELECT
    *,
    "train" AS dataset_name
FROM
    ({CTE_percentages})
WHERE
    bucket_index >= 0
    AND bucket_index < {train_buckets}
""".format(
    CTE_percentages=percentages_query,
    train_buckets=train_buckets)

display_dataframe_head_from_query(train_query)
```

[12]:

| | bucket_index | num_records | percent_records | dataset_name |
|---|---|---|---|---|
| 0 | 9 | 236637 | 0.007168 | train |
| 1 | 30 | 333513 | 0.010103 | train |
| 2 | 20 | 432535 | 0.013103 | train |
| 3 | 15 | 263367 | 0.007978 | train |
| 4 | 39 | 224255 | 0.006793 | train |
| 5 | 2 | 492473 | 0.014918 | train |
| 6 | 45 | 265930 | 0.008056 | train |
| 7 | 33 | 410226 | 0.012427 | train |
| 8 | 53 | 230298 | 0.006976 | train |
| 9 | 73 | 411771 | 0.012474 | train |

We'll do the same by selecting the range of buckets to be used evaluation.

[13]:
```
# Choose hash buckets for validation and pull in their statistics
eval_query = """
SELECT
    *,
    "eval" AS dataset_name
FROM
    ({CTE_percentages})
WHERE
    bucket_index >= {train_buckets}
    AND bucket_index < {cum_eval_buckets}
""".format(
    CTE_percentages=percentages_query,
    train_buckets=train_buckets,
```

```
    cum_eval_buckets=train_buckets + eval_buckets)

display_dataframe_head_from_query(eval_query)
```

[13]:

| | bucket_index | num_records | percent_records | dataset_name |
|---|---|---|---|---|
| 0 | 88 | 423809 | 0.012838 | eval |
| 1 | 85 | 368045 | 0.011149 | eval |
| 2 | 87 | 523881 | 0.015870 | eval |
| 3 | 89 | 256482 | 0.007770 | eval |
| 4 | 82 | 468179 | 0.014182 | eval |
| 5 | 84 | 341155 | 0.010334 | eval |
| 6 | 80 | 312489 | 0.009466 | eval |
| 7 | 83 | 411258 | 0.012458 | eval |
| 8 | 81 | 233538 | 0.007074 | eval |
| 9 | 86 | 274489 | 0.008315 | eval |

Lastly, we'll select the hash buckets to be used for the test split.

[15]:

```
# Choose hash buckets for testing and pull in their statistics
test_query = """
SELECT
    *,
    "test" AS dataset_name
FROM
    ({CTE_percentages})
WHERE
    bucket_index >= {cum_eval_buckets}
    AND bucket_index < {modulo_divisor}
""".format(
    CTE_percentages=percentages_query,
    cum_eval_buckets=train_buckets + eval_buckets,
    modulo_divisor=modulo_divisor)

display_dataframe_head_from_query(test_query)
```

[15]:

| | bucket_index | num_records | percent_records | dataset_name |
|---|---|---|---|---|
| 0 | 96 | 529357 | 0.016036 | test |
| 1 | 93 | 215710 | 0.006534 | test |
| 2 | 98 | 374697 | 0.011351 | test |
| 3 | 97 | 480790 | 0.014564 | test |
| 4 | 91 | 333267 | 0.010096 | test |
| 5 | 94 | 431001 | 0.013056 | test |
| 6 | 90 | 286465 | 0.008678 | test |
| 7 | 95 | 313544 | 0.009498 | test |
| 8 | 92 | 336735 | 0.010201 | test |
| 9 | 99 | 223334 | 0.006765 | test |

In the below query, we'll `UNION ALL` all of the datasets together so that all three sets of hash buckets will be within one table. We added `dataset_id` so that we can sort on it in the query after.

[20]:
```
# Union the training, validation, and testing dataset statistics
union_query = """
SELECT
    0 AS dataset_id,
    *
FROM
    ({CTE_train})
UNION ALL
SELECT
    1 AS dataset_id,
    *
FROM
    ({CTE_eval})
UNION ALL
SELECT
    2 AS dataset_id,
    *
FROM
    ({CTE_test})
""".format(CTE_train=train_query, CTE_eval=eval_query, CTE_test=test_query)

display_dataframe_head_from_query(union_query)
```

[20]:

| | dataset_id | bucket_index | num_records | percent_records | dataset_name |
|---|---|---|---|---|---|
| 0 | 0 | 36 | 246041 | 0.007453 | train |
| 1 | 0 | 3 | 196889 | 0.005964 | train |
| 2 | 0 | 67 | 372457 | 0.011283 | train |
| 3 | 0 | 5 | 449280 | 0.013610 | train |
| 4 | 0 | 35 | 250505 | 0.007588 | train |
| 5 | 0 | 34 | 379000 | 0.011481 | train |
| 6 | 0 | 15 | 263367 | 0.007978 | train |
| 7 | 0 | 71 | 260774 | 0.007900 | train |
| 8 | 0 | 0 | 277395 | 0.008403 | train |
| 9 | 0 | 22 | 257140 | 0.007789 | train |

Lastly, we'll show the final split between train, eval, and test sets. We can see both the number of records and percent of the total data. It is really close to that we were hoping to get.

[21]:
```
# Show final splitting and associated statistics
split_query = """
SELECT
    dataset_id,
    dataset_name,
    SUM(num_records) AS num_records,
    SUM(percent_records) AS percent_records
FROM
```

```
    ({CTE_union})
GROUP BY
    dataset_id,
    dataset_name
ORDER BY
    dataset_id
""".format(CTE_union=union_query)

display_dataframe_head_from_query(split_query)
```

[21]:

| | dataset_id | dataset_name | num_records | percent_records |
|---|---|---|---|---|
| 0 | 0 | train | 25873134 | 0.783765 |
| 1 | 1 | eval | 3613325 | 0.109457 |
| 2 | 2 | test | 3524900 | 0.106778 |

Now that we know that our splitting values produce a good global splitting on our data, here's a way to get a well-distributed portion of the data in such a way that the train, eval, test sets do not overlap and takes a subsample of our global splits.

[25]:
```
def dataframe_from_query(query, count=10):
    """Displays count rows from dataframe head from query.

    Args:
        query: str, query to be run on BigQuery, results stored in dataframe.
        count: int, number of results from head of dataframe to display.
    Returns:
        Dataframe head with count number of results.
    """
    df = bq.query(
        query + " LIMIT {limit}".format(
            limit=count)).to_dataframe()

    return df
```

[41]:
```
# Get the number of records in each of the hash buckets
get_train_df = """
SELECT
        weight_pounds,
    is_male,
    mother_age,
    plurality,
    gestation_weeks,
FROM
    ({CTE_first_bucketing})
WHERE
     ABS(MOD(hash_values, {modulo_divisor})) < {train_buckets}
""".format(
```

```
    CTE_first_bucketing=first_bucketing_query,␣
↪modulo_divisor=modulo_divisor,train_buckets=train_buckets)

# Get the number of records in each of the hash buckets
get_eval_df = """
SELECT
        weight_pounds,
    is_male,
    mother_age,
    plurality,
    gestation_weeks,
FROM
    ({CTE_first_bucketing})
WHERE
     ABS(MOD(hash_values, {modulo_divisor})) >= {train_buckets}
     AND ABS(MOD(hash_values, {modulo_divisor})) < {cum_eval_buckets}
""".format(
    CTE_first_bucketing=first_bucketing_query, modulo_divisor=modulo_divisor,␣
↪train_buckets=train_buckets,
    cum_eval_buckets=train_buckets + eval_buckets)

# Get the number of records in each of the hash buckets
get_test_df = """
SELECT
        weight_pounds,
    is_male,
    mother_age,
    plurality,
    gestation_weeks,
FROM
    ({CTE_first_bucketing})
WHERE
     ABS(MOD(hash_values, {modulo_divisor})) >= {cum_eval_buckets}
     AND ABS(MOD(hash_values, {modulo_divisor})) < {modulo_divisor}
""".format(
    CTE_first_bucketing=first_bucketing_query,␣
↪modulo_divisor=modulo_divisor,cum_eval_buckets=train_buckets + eval_buckets)
```

**Lab Task #2:** Sample the natality dataset

[43]:
```
# TODO 2
# TODO -- Your code here.
# every_n allows us to subsample from each of the hash values
train_df = dataframe_from_query(data_query,100)
eval_df = dataframe_from_query(data_query,100)
test_df = dataframe_from_query(data_query,100)
# This helps us get approximately the record counts we want
```

```
print("There are {} examples in the train dataset.".format(len(train_df)))
print("There are {} examples in the validation dataset.".format(len(eval_df)))
print("There are {} examples in the test dataset.".format(len(test_df)))
```

```
There are 100 examples in the train dataset.
There are 100 examples in the validation dataset.
There are 100 examples in the test dataset.
```

## 1.4 Preprocess data using Pandas

We'll perform a few preprocessing steps to the data in our dataset. Let's add extra rows to simulate the lack of ultrasound. That is we'll duplicate some rows and make the `is_male` field be `Unknown`. Also, if there is more than child we'll change the `plurality` to `Multiple(2+)`. While we're at it, we'll also change the plurality column to be a string. We'll perform these operations below.

Let's start by examining the training dataset as is.

[44]:
```
train_df.head()
```

[44]:
```
   weight_pounds  is_male  mother_age  plurality  gestation_weeks  \
0       4.695846    False          31          1               32
1       5.687926     True          30          1               36
2       7.936641    False          24          1               39
3       8.198992     True          29          1               39
4       7.061406    False          40          1               37

           hash_values
0 -8036809554133325397
1 -6132026233917866995
2  8439164539444335271
3   410994157116516864
4 -4947124986429933431
```

Also, notice that there are some very important numeric fields that are missing in some rows (the count in Pandas doesn't count missing data)

[45]:
```
train_df.describe()
```

[45]:
```
       weight_pounds  mother_age  plurality  gestation_weeks   hash_values
count     100.000000  100.000000 100.000000       100.000000  1.000000e+02
mean        7.431849   26.710000   1.030000        38.880000  5.249029e+17
std         1.200063    6.141916   0.171447         1.950291  5.345227e+18
min         4.499635   16.000000   1.000000        32.000000 -9.192824e+18
25%         6.686620   21.000000   1.000000        38.000000 -3.573048e+18
50%         7.593823   27.000000   1.000000        39.000000  7.224660e+17
75%         8.190724   31.000000   1.000000        40.000000  5.152596e+18
max        10.500618   42.000000   2.000000        43.000000  9.221737e+18
```

It is always crucial to clean raw data before using in machine learning, so we have a preprocessing step. We'll define a `preprocess` function below. Note that the mother's age is an input to our model so users will have to provide the mother's age; otherwise, our service won't work. The features we use for our model were chosen because they are such good predictors and because they are easy enough to collect.

**Lab Task #3:** Preprocess the data in Pandas dataframe

[46]:
```
    # TODO 3
    # TODO -- Your code here.
def preprocess (df):
    # Modify plurality field to be a string
    twins_etc = dict(zip([1,2,3,4,5],
                  ["Single(1)",
                   "Twins(2)",
                   "Triplets(3)",
                   "Quadruplets(4)",
                   "Quintuplets(5)"]))
    df["plurality"].replace(twins_etc, inplace=True)

    # Clone data and mask certain columns to simulate lack of ultrasound
    no_ultrasound = df.copy(deep=True)

    # Modify is_male
    no_ultrasound["is_male"] = "Unknown"

    # Modify plurality
    condition = no_ultrasound["plurality"] != "Single(1)"
    no_ultrasound.loc[condition, "plurality"] = "Multiple(2+)"

    # Concatenate both datasets together and shuffle
    return pd.concat(
        [df, no_ultrasound]).sample(frac=1).reset_index(drop=True)
```

Let's process the train, eval, test set and see a small sample of the training data after our preprocessing:

[47]:
```
train_df = preprocess(train_df)
eval_df = preprocess(eval_df)
test_df = preprocess(test_df)
```

[48]:
```
train_df.head()
```

[48]:
```
   weight_pounds  is_male  mother_age  plurality  gestation_weeks  \
0       8.688418     True          31  Single(1)               41
1       6.393406  Unknown          30  Single(1)               38
2       5.136771  Unknown          21  Single(1)               37
3       6.188376  Unknown          42  Single(1)               37
```

```
4       7.625790  Unknown          22  Single(1)              38

           hash_values
0 -1569657028734518022
1 -1052589534453650062
2 -7363173917873728029
3  7782266297148452291
4  7579041105174423352
```

[49]:
```
train_df.tail()
```

[49]:
```
     weight_pounds  is_male  mother_age  plurality  gestation_weeks  \
195       8.198992  Unknown          29  Single(1)               39
196       7.125340    False          25  Single(1)               40
197       7.625790     True          26  Single(1)               42
198       6.492614  Unknown          28  Single(1)               41
199       8.785421  Unknown          24  Single(1)               41

             hash_values
195   410994157116516864
196  3190725018108452514
197  5655500751972499290
198  8297008456970080747
199  5617905498255901254
```

Let's look again at a summary of the dataset. Note that we only see numeric columns, so `plurality` does not show up.

[50]:
```
train_df.describe()
```

[50]:
```
       weight_pounds  mother_age  gestation_weeks   hash_values
count     200.000000  200.000000       200.000000  2.000000e+02
mean        7.431849   26.710000        38.880000  5.249029e+17
std         1.197044    6.126465         1.945385  5.331780e+18
min         4.499635   16.000000        32.000000 -9.192824e+18
25%         6.686620   21.000000        38.000000 -3.573048e+18
50%         7.593823   27.000000        39.000000  7.224660e+17
75%         8.190724   31.000000        40.000000  5.152596e+18
max        10.500618   42.000000        43.000000  9.221737e+18
```

### 1.5 Write to .csv files

In the final versions, we want to read from files, not Pandas dataframes. So, we write the Pandas dataframes out as csv files. Using csv files gives us the advantage of shuffling during read. This is important for distributed training because some workers might be slower than others, and shuffling the data helps prevent the same data from being assigned to the slow workers.

[51]:
```python
# Define columns
columns = ["weight_pounds",
           "is_male",
           "mother_age",
           "plurality",
           "gestation_weeks"]

# Write out CSV files
train_df.to_csv(
    path_or_buf="train.csv", columns=columns, header=False, index=False)
eval_df.to_csv(
    path_or_buf="eval.csv", columns=columns, header=False, index=False)
test_df.to_csv(
    path_or_buf="test.csv", columns=columns, header=False, index=False)
```

[52]:
```
%%bash
wc -l *.csv
```

```
 200 eval.csv
 200 test.csv
 200 train.csv
 600 total
```

[53]:
```
%%bash
head *.csv
```

```
==> eval.csv <==
7.5618555866,False,20,Single(1),43
4.7509617461,False,36,Single(1),39
6.75055446244,True,21,Single(1),38
7.5618555866,Unknown,20,Single(1),43
7.91239058318,Unknown,30,Single(1),39
5.4454178714,Unknown,30,Single(1),35
7.06140625186,Unknown,40,Single(1),37
6.87401332916,True,29,Single(1),38
8.377565956,Unknown,28,Single(1),42
7.5618555866,Unknown,20,Single(1),40

==> test.csv <==
7.12534030784,True,18,Single(1),42
7.0988848364,Unknown,28,Single(1),40
9.56365292556,Unknown,30,Single(1),40
9.06320359082,Unknown,29,Single(1),41
6.1883756943399995,True,17,Single(1),36
4.7509617461,Unknown,36,Single(1),39
6.41324720158,Unknown,29,Single(1),38
6.87621795178,Unknown,19,Single(1),38
5.1367707046,True,21,Single(1),37
```

```
9.16902547658,Unknown,19,Single(1),40

==> train.csv <==
8.68841774542,True,31,Single(1),41
6.393405598,Unknown,30,Single(1),38
5.1367707046,Unknown,21,Single(1),37
6.1883756943399995,Unknown,42,Single(1),37
7.62578964258,Unknown,22,Single(1),38
8.344496616699999,False,22,Single(1),39
7.81318256528,Unknown,17,Single(1),37
7.936641432,True,36,Single(1),38
8.93754010148,Unknown,31,Single(1),40
8.3665428429,True,25,Single(1),40
```

[54]:
```
%%bash
tail *.csv
```

```
==> eval.csv <==
6.393405598,Unknown,30,Single(1),38
6.944561253,Unknown,32,Single(1),37
6.4992274837599995,True,31,Twins(2),37
7.5618555866,True,20,Single(1),40
7.25100379718,Unknown,28,Single(1),39
5.3241636273,True,38,Twins(2),34
6.0406659788,Unknown,33,Single(1),37
7.12534030784,True,18,Single(1),42
6.4926136159,Unknown,28,Single(1),41
7.12534030784,Unknown,25,Single(1),39

==> test.csv <==
5.8753192823,True,21,Single(1),36
7.13856804356,Unknown,22,Single(1),38
7.12534030784,Unknown,18,Single(1),42
5.4454178714,Unknown,30,Single(1),35
7.06140625186,Unknown,40,Single(1),37
7.31273323054,Unknown,23,Single(1),39
5.8135898489399995,False,37,Single(1),37
9.93843877096,Unknown,28,Single(1),41
7.936641432,False,19,Single(1),40
7.0988848364,False,28,Single(1),40

==> train.csv <==
8.375361333379999,True,30,Single(1),40
7.3744626639,Unknown,26,Single(1),39
4.7509617461,False,36,Single(1),39
6.75055446244,Unknown,29,Single(1),38
7.91239058318,False,30,Single(1),39
8.19899152378,Unknown,29,Single(1),39
```

```
7.12534030784,False,25,Single(1),40
7.62578964258,True,26,Single(1),42
6.4926136159,Unknown,28,Single(1),41
8.7854211407,Unknown,24,Single(1),41
```

### 1.6 Lab Summary:

In this lab, we set up the environment, sampled the natality dataset to create train, eval, test splits, and preprocessed the data in a Pandas dataframe.