Question 1. *Justification of the chosen DBMS model.*

Given our interest in relational databases and our desire to learn a more established approach on the market of data management systems, we decided to pursue the project with PostgeSQL. At first, after opening the *gitIssues* CSV file and familiarising with the columns it was unclear how to import data with different datatypes into the SQL database. We did not find any automated solution on the internet either. It became clear that we needed to first change the values of various datatypes (lists, dates) making them readable by a SQL table prior to inserting values into it, but we were uncertain how to modify a large CSV file outside Pandas. Our solution consisted of iterating over each JSON object using Python library Psycopg2, using custom functions to perform the conversion and finally inserting values, ignoring missing keys by appending null values. In terms of server selection, we chose the local host because it appeared to be 100 times faster than Amazon RDS free Tier.

Before creating any architecture, we created raw tables *gitissues*, *added_deleted_lines* and *gitdata*. Then we inserted values into it from the JSON file using the for loop. In order to understand the nature of the data present we ran numerous SQL scripts checking unique values and relationships between columns.

GitIssues

We've come into conclusion, that there are two entitles in this table: the issues and the corresponding comments. We created the column *issue_id* containing the sequence of integers to assign it as a Primary Key (PK) for *issue* table and as a Foreign Key (FK) for each comment. The size of *issue* and *comment* combined dropped from 73 to 15 mb comparing to raw *gitissues* table in PostgreSQL (further – PG).

GitData

After interacting over some JSON *gitdata* objects, we discovered that the datatypes are very similar to those in gitissues, with the exception of one column – *diff_parsed,* which contains two more nested dictionaries. To unpack it, we decided to separate it into a table called *added_deleted_lines.* Later, we'll use *id* as a FK to connect it to another table, *update_info*. After learning how git commits function and why, for instance, the column hash contains many repeated values, we splitted the *gitdata* table into 2 tables: *commit_info* containing distinct values from columns describing each commit having hash as a PK and *update_info,* which describes the changes made to each file in the GitHub repo (name, path, number of lines of code, etc.). It is linked to *commit_info* via hash as an FK.

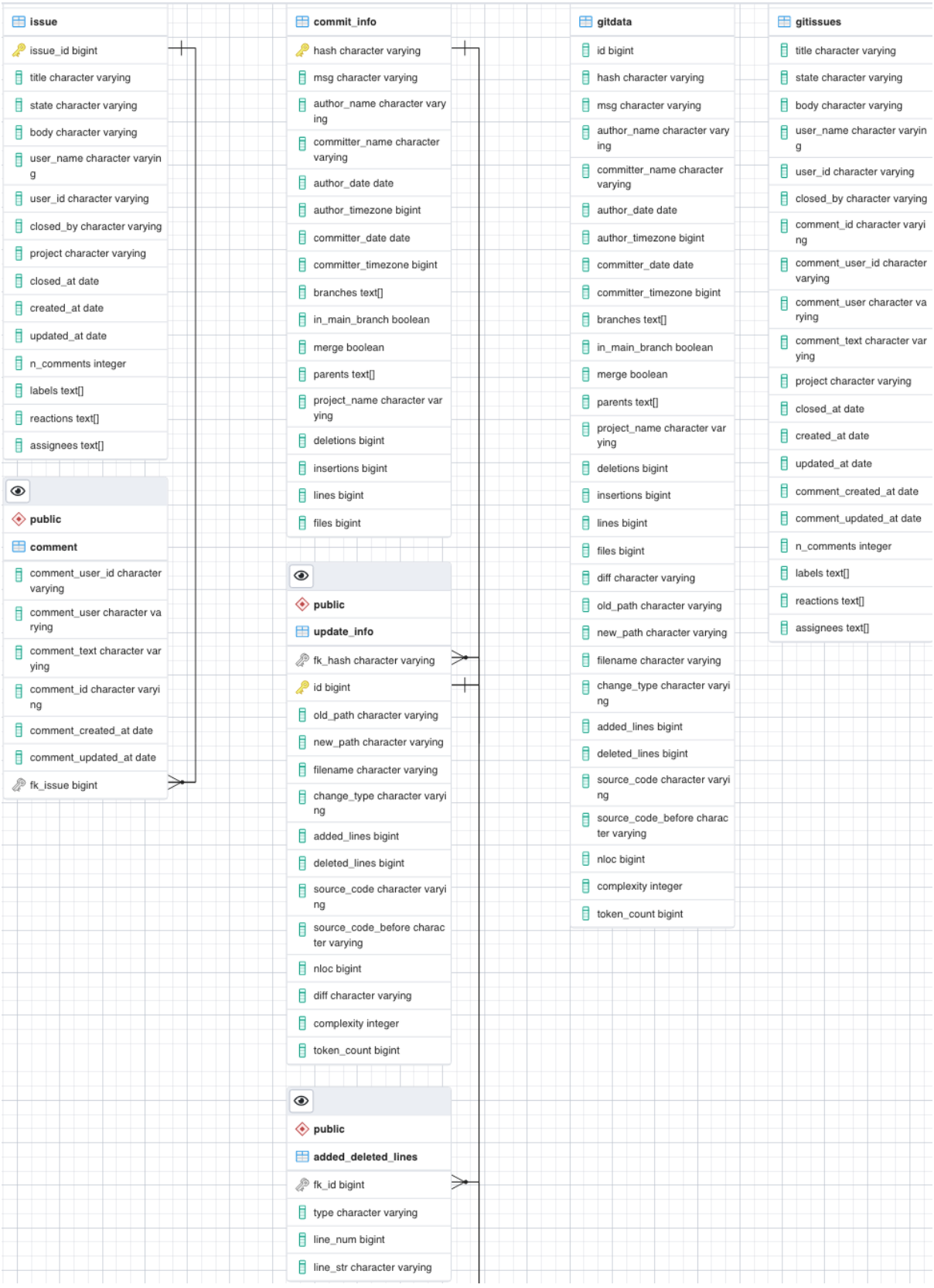
The resulting ERD Diagram is presented on the Figure 1. To increase the speed of the queries we created indexes for all Primary Keys. We did not drop raw PG tables from the schema so that we could compare raw and split tables. However, they can be deleted without losing any information.

Appendix for Question 1

Table 1. The sizes of the files and tables

File / Table name	Type	Size (mb)
gitdata	Raw JSON file	> 8.000
gitissues	Raw JSON file	134
gitissues	PG raw table	73
gitdata	PG raw table	2270
added_deleted_lines	PG table containing unpacked diff_parsed keys and values	300
commit_info	PG table derived from gitdata raw table	0.87
update_info	PG table derived from gitdata raw table	2272
issue	PG table derived from gitissues raw table	0.99
comment	PG table derived from gitissues raw table	14

Figure 1. Relational diagram of the resulting database



Question 2. *Valuable descriptive insights.*

COMMITS EXPLORATION

Out database contains 15 124 unique commits for 9 projects (Tensorflow, PyTorch and 7 additional PyTorch sub-projects); we have 1 102 authors and 182 committers from all time zones. We have similar number of parent commits (14 562/15 124 = 96.3%), which indicates that the structure of the commit-parents relation is likely to be straight rather than tree-shaped.

[1]

	count_hash bigint	count_msg bigint	count_author_na bigint	count_committer_name bigint	count_author_timezone bigint	count_commiter_timezone bigint	count_branches bigint	count_parents bigint	count_project_name bigint	count_files bigint
1	15124	14979	1102	182	16	16	2	14562	9	102

In terms of distribution among the projects, we saw that most commits are for Tensorflow (~60%), second one is main pyrotch project (~30%) and other 10% are for torch-related other projects.

[6]

	project_name character varying	count_proj bigint
1	tensorflow	9152
2	pytorch	4514
3	vision	475
4	serve	382
5	audio	330
6	xla	101
7	torchrec	94
8	text	66
9	elastic	10

As for committers, who contributes mostly with respect to the project, we clearly see that contribution for both projects [7]:

	committer_name character varying	project_name character varying	counter bigint
1	TensorFlower Gardener	tensorflow	8081
2	Facebook GitHub Bot	pytorch	4514
3	GitHub	vision	475
4	GitHub	audio	255
5	Samuel Marks	tensorflow	175
6	GitHub	tensorflow	103
7	GitHub	xla	99
8	Facebook GitHub Bot	torchrec	91
9	jagadeesh	serve	86
10	Facebook GitHub Bot	audio	75
11	lxning	serve	71
12	GitHub	text	66
13	Frederic Bastien	tensorflow	53

	committer_name character varying	project_name character varying	counter bigint
1	Facebook GitHub Bot	torchrec	91
2	GitHub	torchrec	2
3	facebook-github-bot	torchrec	1

COMMITTERS EXPLORATION

Top committers are TensorFlower Gardener (TensorFlower), Facebook GitHub Bot (Facebook), GitHub and Samuel Marks. TensorFlower works solely on tensorflow project and responsible for 88% of tensorlow's commit, Facebook works on 4 PyTorch projects and solely responsible for main pytorch and elastic project commits (it means that no one but this Facebook committed on those 2 projects). GitHub works on both Tensorflow (1%) and some of the pytorch prohects. Share of torchrec commits by Facebook is close to 100% (97%). As for audio, this is the only exception, where another committer GitHub is responsible for 77% of commits (together with Facebook they cover all audio commits). GitHub is solely responsible for vision, test and almost solely (98%) for xla. So, we can see that for 6 out of 8 pytorch projects all commits are made by Facebook or GitHub. As for xla and serve, there are 16 other committers:

[8]

	committer_name character varying	project_name character varying	counter bigint
1	jagadeesh	serve	86
2	lxning	serve	71
3	Mark Saroufim	serve	50
4	Nikhil Kulkarni	serve	38
5	Shrinath Suresh	serve	28
6	Hamid Shojanazeri	serve	23
7	Ubuntu	serve	19
8	min-jean-cho	serve	15
9	Jonathan Sum	serve	7
10	nskool	serve	5
11	유용환	serve	2
12	dbish	serve	1
13	Brian Johnson	serve	1
14	Aaqib Ansari	serve	1
15	Shauheen	xla	1
16	Alex Suhan	xla	1

As for TensorFlow, number of committers who are not TensorFlower or GitHub is sufficiently higher than for all PyTorch projects: 162, even though 36% of them committed only one commit [9].

COMIT-PARENTS RELATION

We have analyzed a structure of commit-parent relation and guessed that the relationship is not tree-based. There is one outlier parent with 108 commits to it, but majority (96%) are one-to-one relations [3]:

	parents text[]	count_parents bigint
1	{'71e3a6d3a7cd6a0bfd924dad53a6bf24e17c884b'}	108
2	{'3ac38d3011e83410c8572ae3b411c1b9e3206ec0'}	8
3	{'68a8f6cb5408ecf49af42ae83eb41f2fd7bc9fb7'}	7
4	{'8903ca1fb059eab3c1e8eccdee1376d4ff52fb67'}	7
5	{'89ccbf66c32c7aede6d184d85ecbf2e1882d648d'}	6
6	{'b56e6db5e7cccfffa824bbc1b5e018c6cc413c21'}	6
7	{'f7fa693f1bb8e96b71239e28d99eb9722bb104b1'}	5
8	{'cdca64c821d4106b752a5e9421e205b685afbfda'}	4
9	{'89c2727b9de0ab84af383c6699eafde7c2ba3977'}	4
10	{'9a2eb827481ca5a2fa3fff9635886301e741ade3'}	4
11	{'e5132a1a1c2a47b2496189dd0e0880d53816dea3'}	4
12	{'2b93a58c7164fa9933cf47b20abe760dad25ba61'}	4

If we go deeper into this, we can see that such non-direct commit-parent relation happens only for ‘Serve’ and “Tensorflow’ projects

	project_name character varying	count_project_name bigint
1	serve	65
2	tensorflow	868

CODE FEATURES WITH RESPECT TO DATE_DIFF

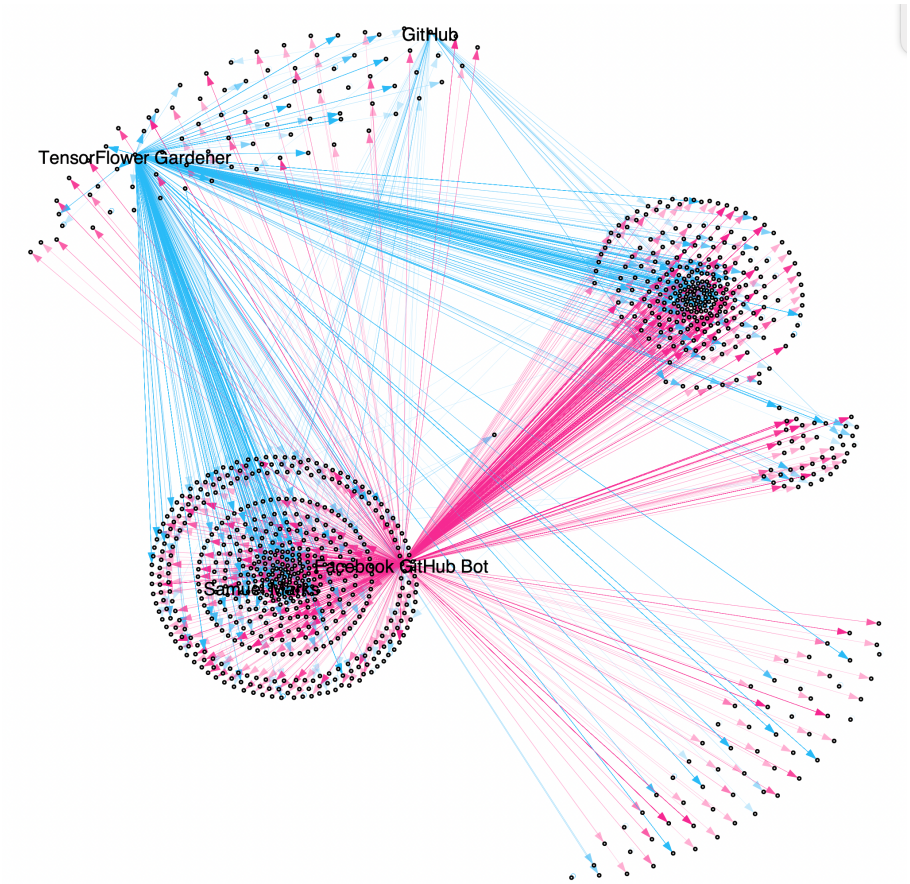
Majority of the commits have been approved on the same day (when author_date is the same as commit_day) [5]:

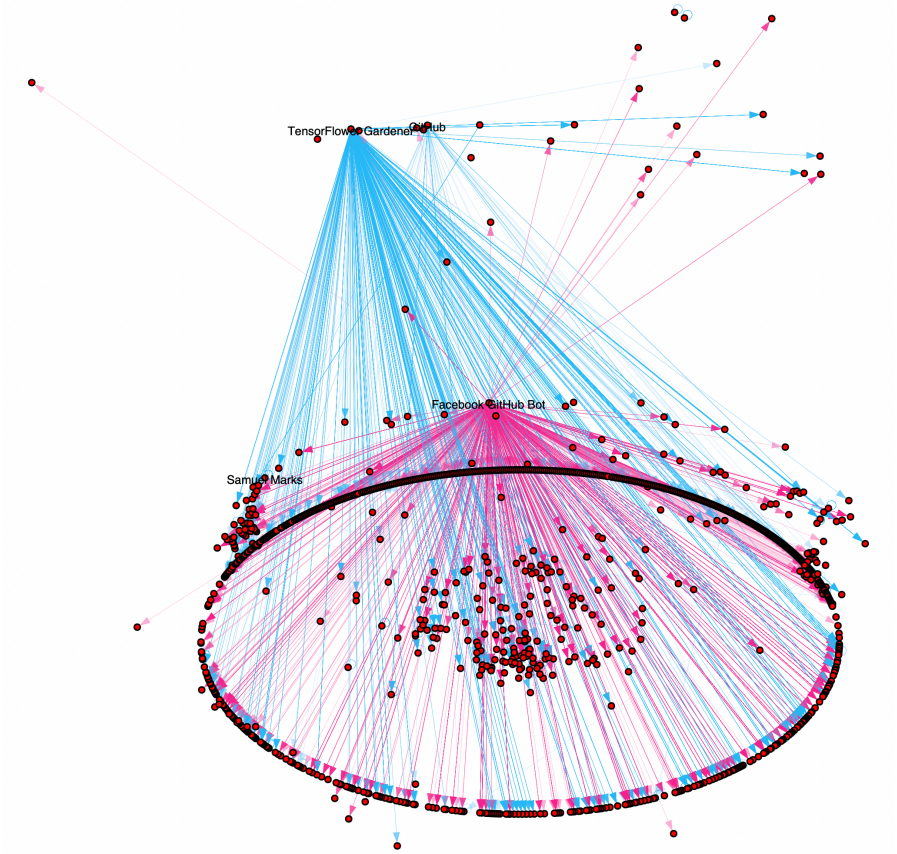
	date_diff integer	counter bigint	deletions numeric	insertions numeric	lines numeric	files numeric	added_lines numeric	deleted_lines numeric
1	0	14657	85.3	128.1	213.4	4.1	[null]	84.1638125127925223
2	1	154	22.4	39.1	61.5	3.1	[null]	22.4025974025974026
3	2	30	106.6	73.9	180.4	3.6	[null]	105.7333333333333333
4	3	8	26.3	16.8	43.0	2.3	[null]	26.2500000000000000
5	4	10	69.3	92.4	161.7	5.0	[null]	69.3000000000000000
6	5	8	5.3	159.0	164.3	3.4	[null]	5.2500000000000000
7	6	7	7.0	63.3	70.3	1.9	[null]	7.0000000000000000
8	7	7	73.0	92.9	165.9	3.3	[null]	73.0000000000000000
9	8	8	35.5	35.9	71.4	1.6	[null]	35.5000000000000000
10	9	2	366.5	385.5	752.0	3.5	[null]	366.5000000000000000
11	10	3	66.7	81.3	148.0	2.3	[null]	66.6666666666666667

Other parameters (deletions, insertions etc.) are less distinct.

COMMITTERS-AUTHORS NETWORK

We decided to explore in more details the network between committers and authors, so we have created a directed network using fruchterman reingold and kamada-hawai layouts in package igraph. The nodes are unique developers and edges represent a commit where direction points from committers to authors. Pink is for PyTorch related projects and blue is for Tensorflow. The edges have opacity and more vivid ones mean that this edge (commit from committer) happened several times. We can clearly see how top performers commit:


GitHub
TensorFlow Gardener
Facebook GitHub Bot
Samuel Marks

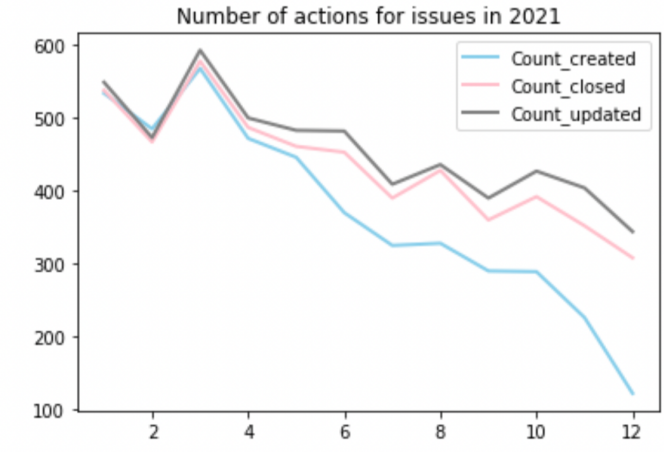

TensorFlow Gardener
GitHub
Facebook GitHub Bot
Samuel Marks

ISSUES EXPLORATION

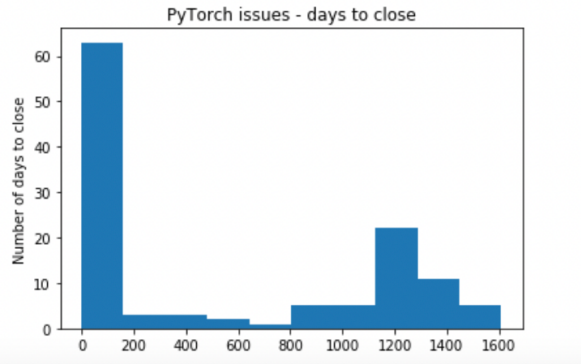
In our database we have 5478 issues, where 97.8% of issues are for TensorFlow:

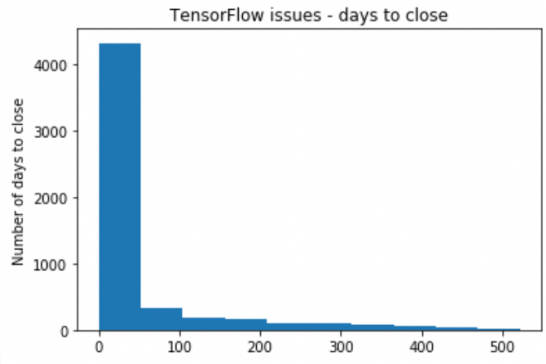
	count bigint	project character varying
1	5358	tensorflow
2	120	pytorch

We have extracted number of issues per month for 2021that have been created/closed or updated:



We have noticed different distribution about how fast the issues are closed for PyTorch and TensorFlow issues [14]





We clearly see that issues with tensorflow are resolved way faster with average number of 46 days where the average number of days to resolve a torch issue is 530.

We decided to go deeper here and check to whom torch's issues are being assign [16] and we see that 85% of the issues are not assigned while for tensorflow this is 3.5%:

	assignees text[]	count_assignees bigint
1	{}	102
2	{'apaszke'}	2
3	{'zou3519'}	2
4	{'kurtamohler'}	2
5	{'pjh5'}	2
6	{'soumith'}	2

	assignees text[]	count_assignees bigint
1	{'gbaned'}	1235
2	{'Saduf2019'}	494
3	{'tilakrayal'}	379
4	{'amahendrakar'}	310
5	{'ymodak'}	291
6	{'jvishnuvardhan'}	261
7	{'sushreebarsa'}	207
8	{}	189

LABELS EXPLORATION

Even though all issues have been closed (column state), 44% of the issues still have the label "awaiting response" and 29% are stalled.

[10]

	counter bigint	tag text
1	2424	stat:awaiting response
2	1606	stalled
3	1436	cla: yes
4	1419	type:bug
5	977	comp:keras
6	799	comp:lite
7	798	TF 2.4
8	723	ready to pull
9	714	type:build/install
10	669	type:support

We have explored information from 'label' and found out that most common labels that issues contain are 'comp', 'type', 'stat' and 'cla' [15]:

	count bigint	tag_type text
1	5172	[null]
2	3876	comp
3	3718	type
4	2920	stat
5	1542	cla
6	1516	size
7	535	subtype
8	133	prtype
9	57	module
10	23	kokoro
11	2	oncall

We have calculated most popular types of issues (from the ‘type’ information in ‘label’) for all projects and ‘bugs’ are the most common ones [13]:

	tag text	count_tag bigint
1	bug	1419
2	build/install	714
3	support	669
4	others	345
5	feature	216
6	performance	167
7	docs-bug	154
8	docs-feature	34

Question 3. *Perform an insightful data analysis.*

We are considering analysing the relationship between project (Tensorflow and Pytorch) with number of comments (*n_comments*) and *lifespan* of this issue, to see how could *n_comments* and *lifespan* determine which project is handling. All of the features are inside *issue* table.
We start by building connection between Pyspark and PostgreSQL, for some reason (might because of M1 chip), it is not able to connect. However, after applying a line of code found on the internet, the connection is built successfully under the terminal, so all the following code is run under terminal.
We connect all tables from PostgreSQL to Pyspark first, and we created a new column "*lifespan_day*" in *issue* table which is the difference of days between the creation date and closure date of an issue. We generate some descriptive tables to have a general insight. We display the tables of *issue* table with *n_comments* and *lifespan* column before and after NA deletions as following.

```
>>> df_3.describe(['n_comments','project','lifespan_days']).show()
+-------+------------------+----------+------------------+
|summary|        n_comments|   project|     lifespan_days|
+-------+------------------+----------+------------------+
|  count|              5478|      5478|              5478|
|   mean|5.8921139101861995|      null| 56.43556042351223|
| stddev| 5.809671274823725|      null|140.90654012369717|
|    min|                 1|   pytorch|                 0|
|    max|                85|tensorflow|              1609|
+-------+------------------+----------+------------------+

>>> df_3 = df_3.na.drop('any')
>>> df_3.describe(['n_comments','project','lifespan_days']).show()
+-------+------------------+----------+------------------+
|summary|        n_comments|   project|     lifespan_days|
+-------+------------------+----------+------------------+
|  count|              2117|      2117|              2117|
|   mean|6.1884742560226735|      null| 67.83089277279169|
| stddev|   6.6883712096811|      null|188.40885018938107|
|    min|                 1|   pytorch|                 0|
|    max|                85|tensorflow|              1609|
+-------+------------------+----------+------------------+
```

The number of Tensorflow projects decreases more than a half, but we still have 2117 entries, so we decide to go with the none NA dataset. Worth noticing that, the standard deviation of the project lifespan is very big, which in Question 2, we see that the lifespan for Pytorch is way longer than Tensorflow in general.
After preprocessing the dataset, now we have 2005 Tensorflow projects and 112 Pytorch projects, and we turn text binary column "*project*" into categorial data by using first "StringIndexer" and second "One-hot-encoder". We then assemble vectors for logistic regression, and we get:

```
>>> df_3.select(['features_0']).show()
+-----------+
| features_0|
+-----------+
| [33.0,1.0]|
|  [6.0,1.0]|
|[114.0,1.0]|
|  [7.0,1.0]|
| [69.0,1.0]|
| [30.0,1.0]|
|  [0.0,1.0]|
| [34.0,1.0]|
|  [2.0,1.0]|
| [86.0,1.0]|
|  [0.0,1.0]|
|  [1.0,1.0]|
|  [0.0,1.0]|
|  [0.0,1.0]|
| [15.0,1.0]|
| [42.0,1.0]|
| [16.0,1.0]|
|  [0.0,1.0]|
| [69.0,1.0]|
|  [1.0,1.0]|
+-----------+
only showing top 20 rows
```

The last step for preparing our dataset is to split the dataset into train and test set. We get 70% of the dataset for train set and 30% for test set.
Since the dataset is imbalanced (2005 vs 112), we first create a training set which has balanced proportion of two projects and fit a logistic model. We use original test set to test the model, surprisingly, the accuracy is 1! After looking into coefficient and prediction details, we can see for different projects, there's significant difference at "rawProdiction", and "probabilty" between Pytorch (1) and Tensorflow (0). So, we can say, some features really determine the project type. With this in mind, we will use the original imbalanced training set to train a logistic model.

```
>>> predictions.select('project_idx', 'rawPrediction', 'probability', 'prediction').toPandas().head(30)
    project_idx                                      rawPrediction                                        probability  prediction
0           0.0  [18.901605976848654, -18.901605976848654]   [0.9999999938178887, 6.18211126734991e-09]         0.0
1           0.0  [18.928985353063748, -18.928985353063748]   [0.9999999939848547, 6.015145270765743e-09]         0.0
2           0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
3           0.0  [18.900552923917306, -18.900552923917306]    [0.999999993811375, 6.188624945835386e-09]         0.0
4           0.0  [18.934250617720497, -18.934250617720497]   [0.9999999940016443, 5.983556983224503e-09]         0.0
5           0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
6           0.0   [18.93530367065185, -18.93530367065185]   [0.9999999940227406, 5.977259354139619e-09]         0.0
7           0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
8           0.0   [18.89212850046651, -18.89212850046651]   [0.9999999937590194, 6.240980621186054e-09]         0.0
9           0.0   [18.93530367065185, -18.93530367065185]   [0.9999999940227406, 5.977259354139619e-09]         0.0
10          0.0  [18.932144511857796, -18.932144511857796]  [0.9999999940038276, 5.9961724474533185e-09]         0.0
11          0.0   [18.93319756478915, -18.93319756478915]   [0.9999999940101385, 5.989861495692139e-09]         0.0
12          0.0  [18.877385759427607, -18.877385759427607]   [0.9999999936663286, 6.333671365155169e-09]         0.0
13          0.0   [18.93530367065185, -18.93530367065185]   [0.9999999940227406, 5.977259354139619e-09]         0.0
14          0.0   [18.93109145892645, -18.93109145892645]  [0.9999999939975099, 6.0024900605526454e-09]         0.0
15          0.0   [18.53303745087622, -18.53303745087622]   [0.9999999910627461, 8.937253936913692e-09]         0.0
16          0.0   [18.93530367065185, -18.93530367065185]   [0.9999999940227406, 5.977259354139619e-09]         0.0
17          1.0   [-19.79148956430032, 19.79148956430032]  [2.5390151512345384e-09, 0.9999999974609849]         1.0
18          0.0  [18.934250617720497, -18.934250617720497]   [0.9999999940016443, 5.983556983224503e-09]         0.0
19          0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
20          0.0      [18.923720088407, -18.923720088407]   [0.9999999939530999, 6.046900091760676e-09]         0.0
21          0.0   [18.74680719594023, -18.74680719594023]    [0.9999999927828609, 7.21713910856181e-09]         0.0
22          0.0   [18.93109145892645, -18.93109145892645]  [0.9999999939975099, 6.0024900605526454e-09]         0.0
23          0.0  [18.921613982544304, -18.921613982544304]   [0.9999999939403512, 6.059648782752447e-09]         0.0
24          0.0  [18.932144511857796, -18.932144511857796]  [0.9999999940038276, 5.9961724474533185e-09]         0.0
25          0.0   [18.93530367065185, -18.93530367065185]   [0.9999999940227406, 5.977259354139619e-09]         0.0
26          0.0  [18.934250617720497, -18.934250617720497]   [0.9999999940016443, 5.983556983224503e-09]         0.0
27          0.0  [18.902659029780008, -18.902659029780008]   [0.9999999938243953, 6.175604694291792e-09]         0.0
28          0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
29          0.0     [18.9363567235832, -18.9363567235832]   [0.9999999940290316, 5.970968386392883e-09]         0.0
>>> evaluator.evaluate(predictions)
```

The result is

```
- Accuracy: 1.0
- Area Under ROC : 1.0
- False Positive Rate by Label: [0.0, 0.0]
- Precision by Label: [1.0, 1.0]
- Tot. Iterations: 24
```

with

```
Coefficients:
=============
[0.004394261525645274,-36.43630252237874]


Intercept:
==========
17.697400671056492
```

Thus, our logistic model predicts 100 accurately, because of distinct lifespan difference between Pytorch and Tensorflow issue.

APPENDIX FOR QUESTION 2: codes

[1] To calculate unique values

```
SELECT COUNT(DISTINCT hash) as count_hash, COUNT(DISTINCT msg) as count_msg,
COUNT(DISTINCT author_name) as count_author_name, COUNT(DISTINCT
committer_name) as count_committer_name,
COUNT(DISTINCT author_timezone) as count_author_timezone, COUNT(DISTINCT
committer_timezone) as count_commiter_timezone,
COUNT(DISTINCT branches) as count_branches, COUNT(DISTINCT parents) as
count_parents,
COUNT(DISTINCT project_name) as count_project_name, COUNT(DISTINCT files) as
count_files
FROM commit_info;
```

[3]

```
SELECT parents, count(*) as count_parents from commit_info group by parents order by
count_parents desc limit 15;
```

[4]

```
-- To find out what projects have non-direct relation between commits and parents
SELECT project_name, COUNT(*) AS count_project_name FROM commit_info WHERE
parents IN
  (SELECT parents FROM commit_info GROUP BY parents HAVING COUNT(*) > 1)
GROUP BY project_name;
```

[5]

```
--distribution for number of commits for dare_diff and average values per commit for
deleted/added lines and changed files
SELECT (committer_date - author_date) as date_diff, count (hash) as counter,
ROUND(AVG(deletions),1) as deletions,
ROUND(AVG(insertions),1) as insertions,
ROUND(AVG(lines),1) as lines,
ROUND(AVG(files),1) as files,
AVG(added_lines) as added_lines,
AVG(deleted_lines) as deleted_lines
```

```
-- add added_lines and deleted_lines
FROM commit_info LEFT JOIN (SELECT fk_hash, ROUND(SUM(added_lines),1) as
added_lines,
ROUND(SUM(deleted_lines),1) as deleted_lines
 FROM update_info GROUP BY fk_hash) as ui
ON commit_info.hash = ui.fk_hash
GROUP BY (committer_date - author_date)
ORDER BY (committer_date - author_date)

[6]
-- Commits distribution with respect to project
SELECT project_name, COUNT(*) AS count_proj FROM commit_info
GROUP BY project_name ORDER BY count_proj DESC;

[7] -- Distribution of committers with respect to the projects
SELECT committer_name, project_name, COUNT (*) AS counter FROM commit_info
GROUP BY committer_name, project_name ORDER BY counter DESC;
-- Distribution of committers with respect to the projects + order by project name
SELECT committer_name, project_name, COUNT (*) AS counter FROM commit_info
GROUP BY committer_name, project_name ORDER BY project_name, counter DESC;

[8]
-- Find non Facebook or GirHub torch contributers
SELECT committer_name, project_name, COUNT (*) AS counter FROM commit_info
WHERE (project_name = 'xla' OR project_name = 'serve') AND
(committer_name!='GitHub') GROUP BY committer_name, project_name ORDER BY
project_name, counter DESC;

[9] – Contributors for TensorFlow without
SELECT committer_name, project_name, COUNT (*) AS counter FROM commit_info
WHERE (project_name = 'tensorflow') AND (committer_name!='TensorFlower Gardener'
AND committer_name!= 'GitHub') GROUP BY committer_name, project_name ORDER BY
project_name, counter DESC;
```

[10]--To check most popular labels with information inside

```
DROP TABLE IF EXISTS tags;
CREATE TABLE tags AS
SELECT issue_id, REPLACE(tag, "", '') as tag
FROM (SELECT issue_id, unnest(labels) AS tag FROM issue) as t;
SELECT issue_id, tag from tags;
SELECT COUNT(*) as counter, tag FROM tags GROUP BY tag ORDER BY counter
DESC;
```

[11] -- Count issues per project

```
SELECT  COUNT(*), project from issue GROUP BY project ORDER BY count DESC;
```

[12]

```
select to_char(created_at,'mm') as month_created,
      extract(year from created_at) as year_created,
      count("issue_id") as "Count_created"
from issue
where created_at > '2020-12-31'
group by 1,2
order by 2,1

select to_char(closed_at,'mm') as month_closed,
      extract(year from closed_at) as year_closed,
      count("issue_id") as "Count_closed"
from issue
where closed_at > '2020-12-31'
group by 1,2
order by 2,1

select to_char(updated_at,'mm') as month_updated,
      extract(year from updated_at) as year_updated,
      count("issue_id") as "Count_updated"
from issue
where updated_at > '2020-12-31'
```

```
group by 1,2
order by 2,1

[13]
--To count most popular types of issues that contain 'type' information
SELECT tag, COUNT(*) AS count_tag FROM issue_tag WHERE tag_type='type' GROUP
BY tag ORDER BY count_tag DESC;

[14] – tensorflow issues
SELECT created_at, closed_at, closed_at-created_at AS difference from issue where project
= 'tensorflow' order by difference DESC;
– torch issues
SELECT created_at, closed_at, closed_at-created_at AS difference from issue where project
!= 'tensorflow' order by difference DESC;

--create distributions for tensorflow
select width_bucket(closed_at-created_at, 0, 600, 6) as bucket,
      count(*) as cnt from issue where project = 'tensorflow'
group by bucket
order by bucket;

--create distributions for pytorch
select width_bucket(closed_at-created_at, 0, 1700, 10) as bucket,
      count(*) as cnt from issue where project != 'tensorflow'
group by bucket
order by bucket;


[15]
--To count how many issues contain a specific label
SELECT * FROM issue_tag;
SELECT count(*), tag_type FROM issue_tag GROUP BY tag_type ORDER BY count
DESC;
[16] to check on whom torch's issues are being assign
```

SELECT assignees, count(assignees) as count_assignees from issue where project = 'pytorch' group by assignees order by count_assignees DESC;

SELECT assignees, count(assignees) as count_assignees from issue where project != 'pytorch' group by assignees order by count_assignees DESC;