Project 2: Exploring the GitHub Dataset with Colaboratory

In this project, you will explore one of BigQuery's public datasets on GitHub and learn to make visualizations in order to answer your questions. This project is due on **Monday, November 4th at 11:59 PM**. It is worth 50 points, for 10% of your overall grade. After completing this project, make sure to follow the submission instructions in the handout to submit on Gradescope.

Notes (read carefully!):

- Be sure you read the instructions on each cell and understand what it is doing before running it.
- Don't forget that if you can always re-download the starter notebook from the course website if you need to.
- You may create new cells to use for testing, debugging, exploring, etc., and this is in fact encouraged! Just make sure that the final answer for each question is **in its own cell** and **clearly indicated**.
- Colab will not warn you about how many bytes your SQL query will consume. **Be sure to check on the BigQuery UI first before running queries here!**
- See the assignment handout for submission instructions.
- Have fun!

Collaborators:

Please list the names and SUNet IDs of your collaborators below:

- *Name, SUNet ID*

Overview

BigQuery has a massive dataset of GitHub files and statistics, including information about repositories, commits, and file contents. In this project, we will be working with this dataset. Don't worry if you are not too familiar with Git and GitHub -- we will explain everything you need to know to complete this part of the assignment.

Notes

The GitHub dataset available on BigQuery is actually quite massive. ***A single query on the "contents" table alone (it is 2.16TB!) can eat up your 1TB allowance for the month AND cut into about 10% of your GCloud credit for the class.***

To make this part of the project more manageable, we have subset the original data. We have preserved almost all information in the original tables, but we kept only the information on the top 500,000 most "watched" GitHub repos between January 2016 and October 2018.

You can see the tables we will be working with here. **Read through the schemas to get familiar with the data**. Note that some of the tables are still quite large (the contents table is about 500GB), so you should exercise the usual caution when working with them. Before running queries on this notebook, it's good practice to first set up query limits on your BigQuery account or see how many bytes will be billed on the web UI.

Make sure to use our subsetted dataset, not the original BigQuery dataset!

A Super Quick Primer on Git

If you are not very familiar with Git and GitHub, here are some high-level explanations that will give you enough context to get you through this part of the problem:

- *GitHub*: GitHub is a source-control service provider. GitHub allows you to collaborate on and keep track of source code in a fairly efficient way.
- *commit*: A commit can be thought of as a change that is applied to some set of files. i.e., if some set of files is in some state A, you can make changes to A and *commit* your changes to the set of files so that it is now in state B. A commit is identified by a *hash* of the

information in your change (the author of the commit, who actually committed [i.e. applied] the changes to the set of files, the changes themselves, etc.)

- *parent commit*: The commit that came before your current commit.
- *repo*: A repo (short for repository) is GitHub's abstraction for a collection of files along with a history of commits on those files. If you have GitHub username "foo" and you make a repository called "data-rocks", your repo name will be "foo/data-rocks". You can think of a repo's history in terms of its commits. E.g., "foo/data-rocks" can have the set of "states" A->B->C->D, where each state change (A->B, B->C, C->D) was due to a commit.
- *branch*: To keep track of different commit histories, GitHub repos can have branches. The 'main' branch (i.e. commit history) of the repo is called the 'master' branch. Say on "foo/data-rocks" we have the commit history A->B->C->D on the master branch. If someone else comes along and decides to add a cool new feature to "foo/data-rocks", they can create a branch called "cool-new-feature" that branches away from the master branch. All the code from the main branch will be there, but new code added to "cool-new-feature" will not be on the main branch.
- *ref*: For the purpose of this assignment, you can think of the 'ref' field on the "files" table as referring to the branch in which a file lives in a repository at some point in time.

For the purposes of this question, you don't need to know about the following things in detail:

- Commit trees
- The encoding attribute on the commits table

If you want more clarifications about Git and GitHub in order to answer this question, be sure to post on Piazza or come to Office Hours. In

Section 1 | Understanding the Dataset (4 points)

Question 1: Schema Comprehension (4 points)

Each of the following parts is worth 1 point.

a) What is the primary key of `github_repo_files`? (1 point)

Things to note:

- A file ID changes based on a file's contents; it is not assigned at a file's creation.
- Different repos can have files with the same paths.
- It is possible to have separate files with identical contents.
- A repo may have one file across multiple branches.

---

lrepo_name, path, and id.

---

b) What is the primary key in `github_repo_licenses`? What is the foreign key? (1 point)

Primary key and foreign key is lrepo_name.

---

c) If we were given an author and we wanted to know what language repos they like to contribute to, which tables should we use? (1 point)

---

github_repo_languages and github_repo_commits

---

d) If we wanted to know whether using different licenses had an effect on a repo's watch count, which tables would we use? (1 point)

---

github_repos and github_repo_licenses

Section 2 | Query Performance (8 points)

In this section, we'll look at some inefficient queries and think about how we can make them more efficient. For this section, we'll consider efficiency in terms of how many bytes are processed.

Question 2: Optimizing Queries (8 points)

For the next three subquestions, consider the following query:

```
SELECT author.name
FROM `cs145-fa19.project2.github_repo_commits` commits_1
WHERE (SELECT COUNT(*)
       FROM `cs145-fa19.project2.github_repo_commits` commits_2
       WHERE commits_1.author.name = commits_2.author.name) > 20
```

NOTE: We do **NOT** recommend running this unoptimized query in BigQuery, as it will run for a very long time (over 15 minutes if not longer). However, feel free to run an optimized version of this query after finishing part (c), which takes about 5 seconds to run.

a) In one to two sentences, explain what this query does. (1 point)

This query outputs author names who have more than 20 commits.

---

b) Briefly explain why this query is inefficient (in terms of bytes that need to be processed) and how it can be improved to be more efficient. (1 point)

Because this query uses inner join of which the cross product takes lots of time. It can be optimized by using GROUP BY and HAVING clauses.

c) Following from part (b), write a more efficient version of the query. (2 points)

```
SELECT author.name
FROM `cs145-fa19.project2.github_repo_commits`
GROUP BY author.name
HAVING COUNT(*) > 20
```

For the next three subquestions, consider the following query:

```
SELECT id
FROM (
  SELECT files.id, files.mode, contents.size
  FROM
    `cs145-fa19.project2.github_repo_files` files,
    `cs145-fa19.project2.github_repo_readme_contents` contents
  WHERE files.id = contents.id
  )
```

```
WHERE mode = 33188 AND size > 1000
LIMIT 10
```

d) Briefly explain why this query is inefficient (in terms of bytes that need to be processed) without the query optimization and how it can be improved to be more efficient. (1 point)

This query contains nested query, which introduces extra projection cost to output mode and size. It can be optimized without the nested query, and then, apply the conditions directly after joining the two tables.

---

e) Following from part (d), write a more efficient version of the query. (2 points)

Hint: Think about the rows the number of bytes processed by the query. Are all the operators required for the final result?

```
SELECT files.id
FROM `cs145-fa19.project2.github_repo_files` files,
    `cs145-fa19.project2.github_repo_readme_contents` contents
WHERE files.id = contents.id
AND files.mode = 33188
AND contents.size > 1000
LIMIT 10
```

---

f) Run both the original query and your optimized query on BigQuery and pay attention to the number of bytes processed. How do they compare, and is it what you expect? Explain why this is happening in a few sentences. (1 point)

Hint: Look at the query plan under "Execution details" in the bottom panel of BigQuery. It may be especially helpful to look at stage "S00: Input".

Both queries consume the same bytes, which are 3.2 GB. They have nearly the same runtime. BigQuery will automatically optimize the query for you, no matter how many nested queries you write, as long as the output is the same.

To learn more about writing efficient SQL queries and how BigQuery optimizes queries, check out Optimizing query computation and Query plan and timeline.

Section 3: Visualizing the Dataset (38 points)

In this section, you'll be answering questions about the dataset, similar to the first project. The difference is that instead of answering with a query, you will be answering with a visualization. Part of this assignment is for you to think about which data (specifically, which indicators) you should be using in order to answer a particular question, and about what type of chart/picture/visualization you should use to clearly convey your answer.

General Instructions

- For each question, you will have at least two cells - a SQL cell where you run your query (and save the output to a data frame), and a visualization cell, where you construct your chart. For this project, make sure that **all data manipulation is to be done in SQL**. Please do not modify your data output using `pandas` or some other data library.
- Please make all charts clear and readable - this includes adding axes labels, clear tick marks, clear point markers/lines/color schemes (i.e. don't repeat colors across categories), legends, and so on.

Setting up BigQuery and Dependencies

Run the cell below (shift + enter) to authenticate your project.

Note that you need to fill in the `project_id` variable with the Google Cloud project id you are using for this course. You can see your project ID by going to https://console.cloud.google.com/cloud-resource-manager

```
# Run this cell to authenticate yourself to BigQuery.
from google.colab import auth
auth.authenticate_user()
project_id = "cs145-fa19-254923"
```

Visualization

For this project, we will be officially supporting the use of matplotlib (https://matplotlib.org/3.0.0/tutorials/index.html), but feel free to use another graphing library if you are more comfortable with it.

```
# Add imports for any visualization libraries you may need
import matplotlib.pyplot as plt
%matplotlib inline
```

How to Use BigQuery and visualize in Colab

Jupyter notebooks (what Colab notebooks are based on) have a concept called "magics". If you write the following line at the top of a `Code` cell:

```
%%bigquery --project $project_id variable # this is the key line
SELECT ....
FROM ...
```

That "%%" converts the cell into a SQL cell. The resulting table that is generated is saved into `variable`.

Then in a second cell, use the library of your choice to plot the variable. Here is an example using matplotlib:

```
plt.figure()
plt.scatter(variable["x"], variable["y"])
plt.title("Plot Title")
plt.xlabel("X-axis label")
plt.ylabel("Y-axis label")
```

Question 3: A First Look at Repo Features (6 points)

Let's get our feet wet with this data by creating the following plots:

1. Language distribution across repos
2. File size distribution across repos
3. The distribution of the length of commit messages across repos

Note that you will not receive full credit if your charts are poorly made (i.e. very unclear or unreadable).

Hints

- Some of these plots will need at least one of their axes to be log-scaled in order to be readable
- For more readable plots, you can use pandas.DataFrame.sample. A sample size between 1,000 and 10,000 should give you more readable plots.

Reminders

- Be careful with your queries! Don't run `SELECT *` blindly on a table in this Colab notebook since you will not get a warning of how much data the query will consume. Always how much data a query will consume on the BigQuery UI first -- you are also better off setting a query limit as we described earlier.
- Don't forget to use the subsetted GitHub tables we provide here, not the original ones on BigQuery.

a) Language distribution (2 points)

(x-axis: programming language, y-axis: # repos containing at least one file in that language)

To keep the chart readable, only keep the top 20 languages.

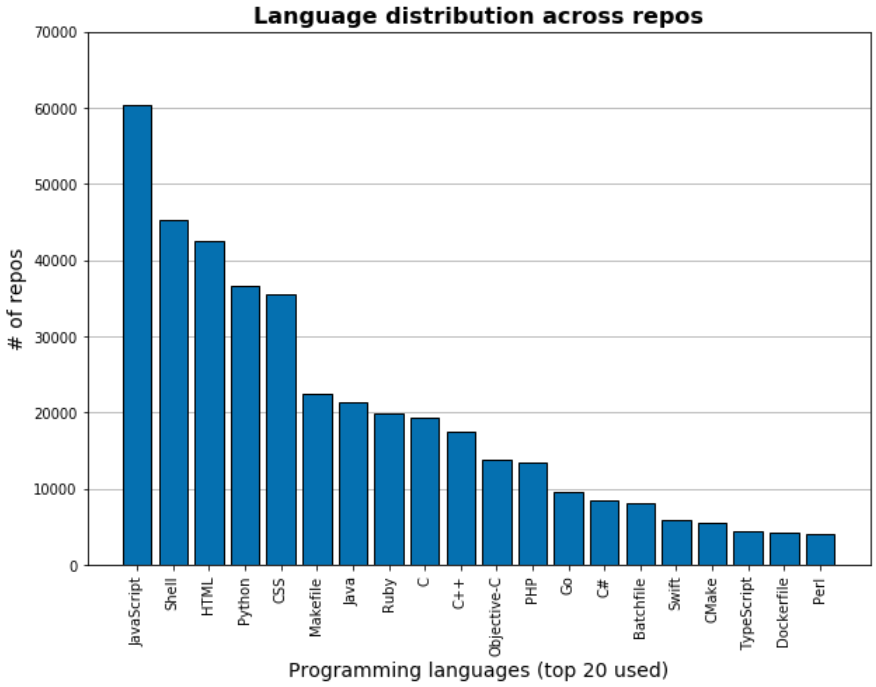
Hint: https://cloud.google.com/bigquery/docs/reference/standard-sql/arrays

```
%%bigquery --project $project_id q3a

SELECT name, COUNT(lrepo_name) AS cnt
FROM `cs145-fa19.project2.github_repo_languages`
CROSS JOIN UNNEST(language)
GROUP BY name
ORDER BY COUNT(lrepo_name) DESC
LIMIT 20
```

```
plt.figure(figsize=(10, 7))
plt.grid(axis='y', zorder=0)
x_pos = list(range(1, 21))
plt.bar(x_pos, q3a["cnt"], align='center', edgecolor='black',color='#0570B0', zorder = 3)
plt.title("Language distribution across repos",fontsize=16,fontweight='bold')
plt.xlabel("Programming languages (top 20 used)", fontsize=14)
plt.xticks(x_pos, q3a["name"], rotation='vertical')
plt.ylabel("# of repos", fontsize=14)
plt.ylim([0, 70000])
```

```
(0, 70000)
```

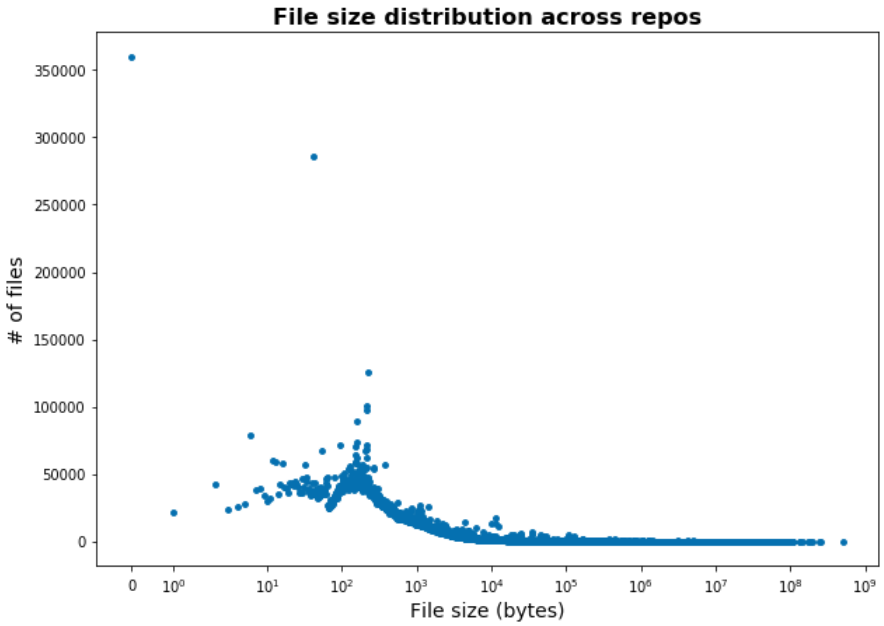


b) File size distribution (2 points)

(x-axis: file size, y-axis: # files of that size)

```
%%bigquery --project $project_id q3b
SELECT size, COUNT(id) AS cnt
FROM `cs145-fa19.project2.github_repo_contents`
GROUP BY size
ORDER BY size
```

```
plt.figure(figsize=(10, 7))
plt.scatter(q3b["size"], q3b["cnt"], color='#0570B0', s=15)
plt.title("File size distribution across repos",fontsize=16,fontweight='bold')
plt.xlabel("File size (bytes)", fontsize=14)
plt.ylabel("# of files", fontsize=14)
plt.xscale('symlog')
# plt.yscale('log')
```



c) The distribution of the length of commit messages (2 points)

(x-axis: length of the commit message, y-axis: # commits with that length)

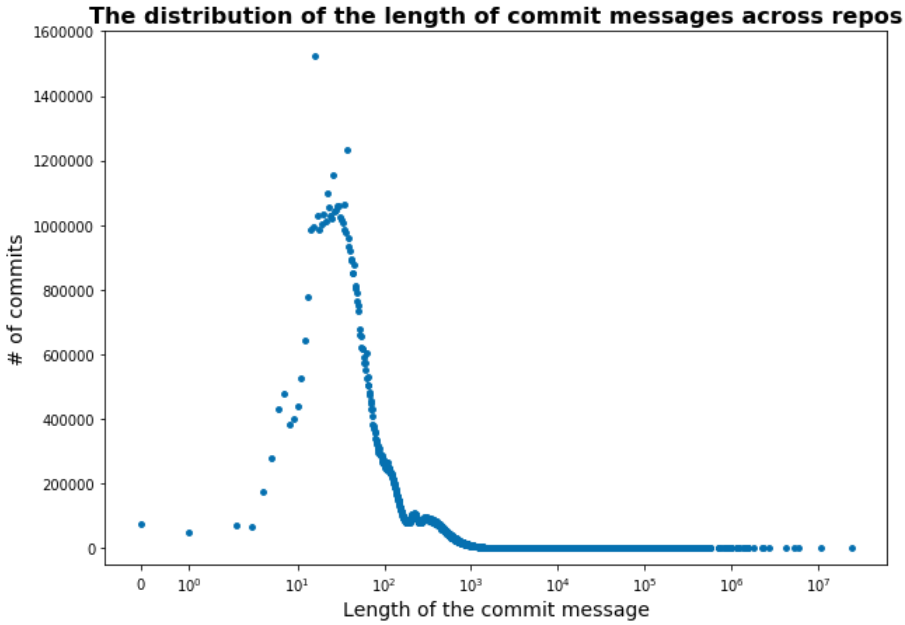
Note: The query for this plot may use ~30GB of data.

```
%%bigquery --project $project_id q3c

SELECT CHAR_LENGTH(message) AS message_length, COUNT(commit) AS cnt
FROM `cs145-fa19.project2.github_repo_commits`
GROUP BY CHAR_LENGTH(message)
```

```
plt.figure(figsize=(10, 7))
plt.scatter(q3c["message_length"], q3c["cnt"], color='#0570B0', s=15)
plt.title("The distribution of the length of commit messages across repos",fontsize=16,fontweight='bold')
plt.xlabel("Length of the commit message", fontsize=14)
plt.ylabel("# of commits", fontsize=14)
plt.xscale('symlog')
plt.ylim([-50000, 1600000])
# plt.yscale('log')
```

```
(-50000, 1600000)
```



What Makes a Good Repo?

Given that we have some interesting data at our disposal, let's try to answer the question: what makes a good GitHub repo? For our purposes, a "good" repo is simply a repo with a high watch count; this refers to how many people are following the repo for updates.

To begin, let's see if any of the features we've already explored give us any good answers.

Question 4: Using What We've Worked With (17 points)

Create plots for the following features in a repo and how they relate to that repo's watch count:

1. Languages used
2. Average file size in a repo
3. Average message length of commits in a repo

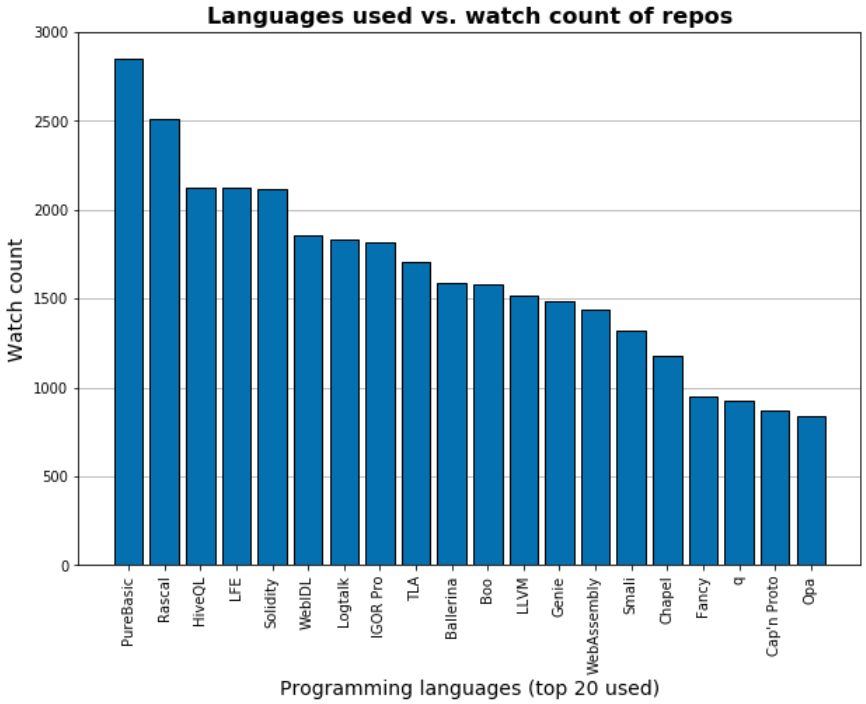
a) Languages used (4 points)

```
%%bigquery --project $project_id q4a

SELECT name, ROUND(AVG(b.watch_count)) AS watch_count
FROM `cs145-fa19.project2.github_repo_languages` a
CROSS JOIN UNNEST(language)
JOIN `cs145-fa19.project2.github_repos` b
ON a.lrepo_name = b.lrepo_name
GROUP BY name
ORDER BY AVG(b.watch_count) DESC
LIMIT 20
```

```
plt.figure(figsize=(10, 7))
x_pos = list(range(1, 21))
plt.bar(x_pos, q4a["watch_count"], align='center', edgecolor='black',color='#0570B0', zorder=3)
plt.title("Languages used vs. watch count of repos",fontsize=16,fontweight='bold')
plt.xlabel("Programming languages (top 20 used)", fontsize=14)
plt.xticks(x_pos, q4a["name"], rotation='vertical')
plt.ylabel("Watch count", fontsize=14)
plt.grid(axis='y', zorder=0)
plt.ylim([0.0, 3000])
```

```
(0.0, 3000)
```



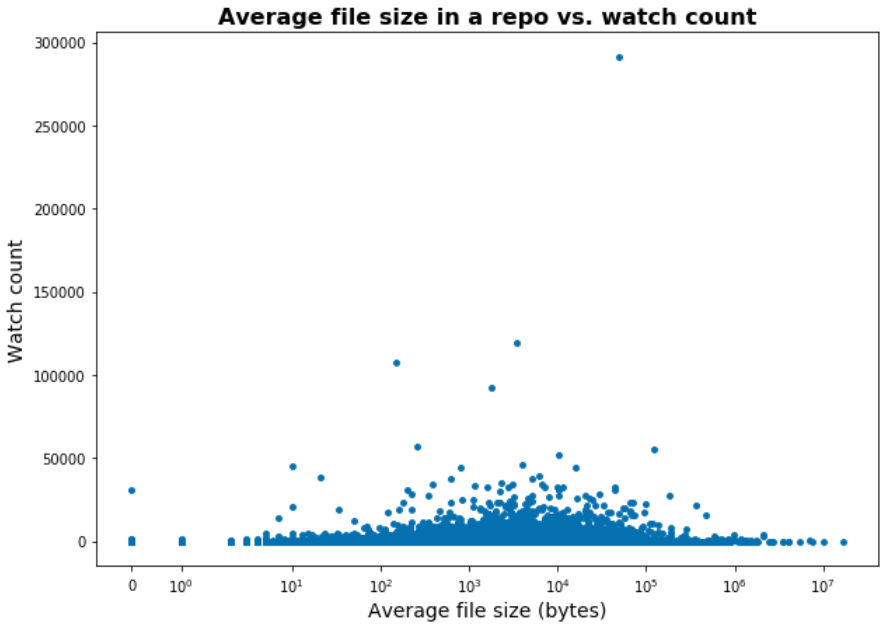
b) Average file size in a repo (4 points)

Note: For this question, you may use the `github_repo_readme_contents` table instead of the full contents table.

```
%%bigquery --project $project_id q4b

SELECT average_size, c.watch_count AS watch_count
FROM `cs145-fa19.project2.github_repos` c, (SELECT a.lrepo_name, ROUND(AVG(b.size)) AS average_size
  FROM `cs145-fa19.project2.github_repo_files` a
  JOIN `cs145-fa19.project2.github_repo_readme_contents` b
  ON a.id = b.id
  GROUP BY a.lrepo_name) AS table
WHERE table.lrepo_name = c.lrepo_name


plt.figure(figsize=(10, 7))
plt.scatter(q4b["average_size"], q4b["watch_count"], color='#0570B0', s=15)
plt.title("Average file size in a repo vs. watch count",fontsize=16,fontweight='bold')
plt.xlabel("Average file size (bytes)", fontsize=14)
plt.ylabel("Watch count", fontsize=14)
plt.xscale('symlog')
# plt.yscale('symlog')
```



c) Average message length of commits on a repo. (6 points)

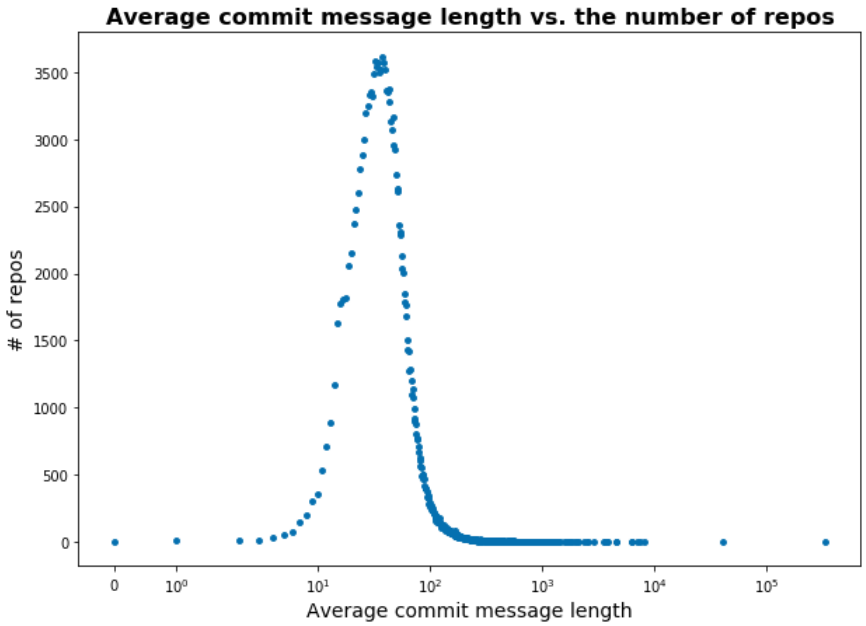
First, make a plot of the average commit message length of repositories against the number of repositories with that average commit message length.

Then, make a plot of how average commit message length of a repository correlates to its watch count. Round the average commit message length to the nearest integer.

```
%%bigquery --project $project_id q4c_avg_commit_length_count

SELECT table.average_len AS average_length, COUNT(table.lrepo_name) AS cnt
FROM (SELECT lrepo_name, ROUND(AVG(CHAR_LENGTH(message))) AS average_len
  FROM `cs145-fa19.project2.github_repo_commits`
  GROUP by lrepo_name) table
GROUP BY table.average_len
ORDER BY table.average_len
```

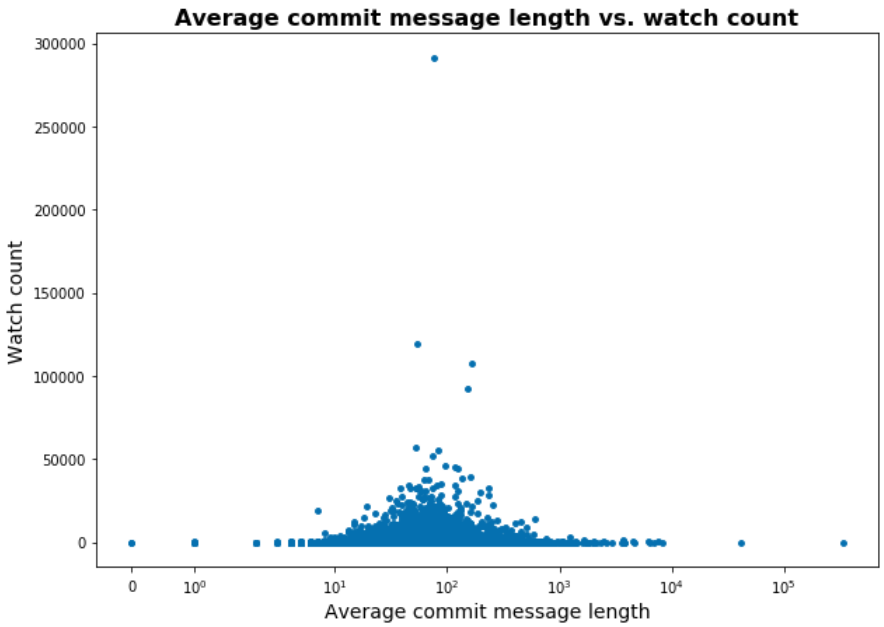
```
plt.figure(figsize=(10, 7))
plt.scatter(q4c_avg_commit_length_count["average_length"], q4c_avg_commit_length_count["cnt"], color='#0570B0', s=15)
plt.title("Average commit message length vs. the number of repos",fontsize=16,fontweight='bold')
plt.xlabel("Average commit message length", fontsize=14)
plt.ylabel("# of repos", fontsize=14)
plt.xscale('symlog')
# plt.ylim([-250, 4000])
# plt.yscale('log')
```




```
%%bigquery --project $project_id q4c_msg_length_watch_count

SELECT average_len, b.watch_count AS cnt
FROM (SELECT a.lrepo_name, ROUND(AVG(CHAR_LENGTH(a.message))) AS average_len
  FROM `cs145-fa19.project2.github_repo_commits` a
  GROUP by a.lrepo_name) table
JOIN `cs145-fa19.project2.github_repos` b
ON table.lrepo_name = b.lrepo_name
```

```
plt.figure(figsize=(10, 7))
plt.scatter(q4c_msg_length_watch_count["average_len"], q4c_msg_length_watch_count["cnt"], color='#0570B0', s=15)
plt.title("Average commit message length vs. watch count",fontsize=16,fontweight='bold')
plt.xlabel("Average commit message length", fontsize=14)
plt.ylabel("Watch count", fontsize=14)
plt.xscale('symlog')
# plt.yscale('symlog')
```



d) Which, if any, of the features we inspected above have a high correlation to a repo having a high watch count? Does the answer make sense, or does it seem counterintuitive? Explain your answer in a small paragraph, no more

than 200 words. Be sure to cite the charts you generated. (3 points)

There is correlation between programming languages and watch count to certain degree. For instance, PureBasic is the most popular language. There are notable differences in watch count among these languages. In other words, watch count is not nearly the same, which indicates a higher correlation. However, the plot of average file size per repo aganist watch is quite noisy, which suggests a weaker correlation between these two variables. Some repos having greater watch count have various file sizes from low to high. As for average commit message length, repos with greater popularity are densely distributed around average message length of 50. Too short or too long for the message length both results in lower watch count.

What Do Others Have to Say?

At this point we have learned a couple of things about how certain features may or may not impact the popularity of a GitHub repo. However, we really only looked at features of GitHub repos that we had initially explored when we were getting a feel for the dataset! There has got to be more things we can inspect than that.

If you do a web search for "how to make my git repo popular," you will find that more than a couple of people suggest investing time in your `README` file. The `README` usually gives an overview to a GitHub project and may include other information about the codebase such as whether its most recent build passed or how to begin contributing to that repo. Here is an example `README` file for the popular web development framework Vue.js.

IMPORTANT: Note about Contents Table

Note that the original `github_repo_contents` table is about half a TB! In order to save you the pain of using up 500GB of your credits to subset this table into a workable size for this problem, we have done it for you.

*For the rest of this question, be sure that you use the `github_repo_readme_contents` table *

Question 5: Analyzing README Features (15 points)

Analyze the following features of a repo's `README` file and how they relate to the popularity of a repository, generating an informative plot for each feature:

1. Having or not having a `README` file
2. The length of the `README` file

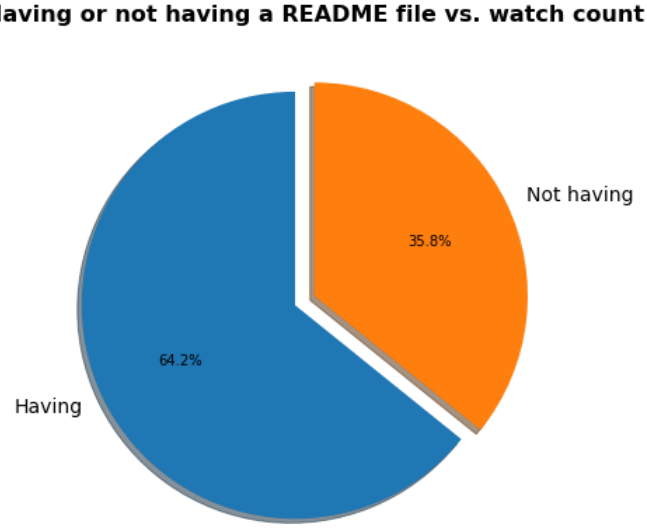
Consider a `README` file to be any file with the path beginning with "README", not case-sensitive.

a) Having or not having a `README` file (6 points)

```
%%bigquery --project $project_id q5a
SELECT readme_cnt, noreadme_cnt
FROM (SELECT AVG(b1.watch_count) AS readme_cnt
  FROM `cs145-fa19.project2.github_repos` b1
  WHERE b1.lrepo_name IN (SELECT DISTINCT a1.lrepo_name
  FROM `cs145-fa19.project2.github_repo_files` a1
  WHERE LOWER(a1.path) LIKE '%readme%')),
  (SELECT AVG(b2.watch_count) AS noreadme_cnt
  FROM `cs145-fa19.project2.github_repos` b2
  WHERE b2.lrepo_name NOT IN (SELECT DISTINCT a2.lrepo_name
  FROM `cs145-fa19.project2.github_repo_files` a2
  WHERE LOWER(a2.path) LIKE '%readme%'))
```

```
plt.figure(figsize=(10, 7))
explode = (0.1, 0)
labels = ['Having', 'Not having']
patches, texts, autotexts = plt.pie(q5a, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
texts[0].set_fontsize(14)
texts[1].set_fontsize(14)
plt.title("Having or not having a README file vs. watch count", fontsize=16,fontweight='bold')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: MatplotlibDeprecationWarning: Non-1D inputs to pie() ar
  """
Text(0.5, 1.0, 'Having or not having a README file vs. watch count')
```



b) The length of the `README` file (6 points)

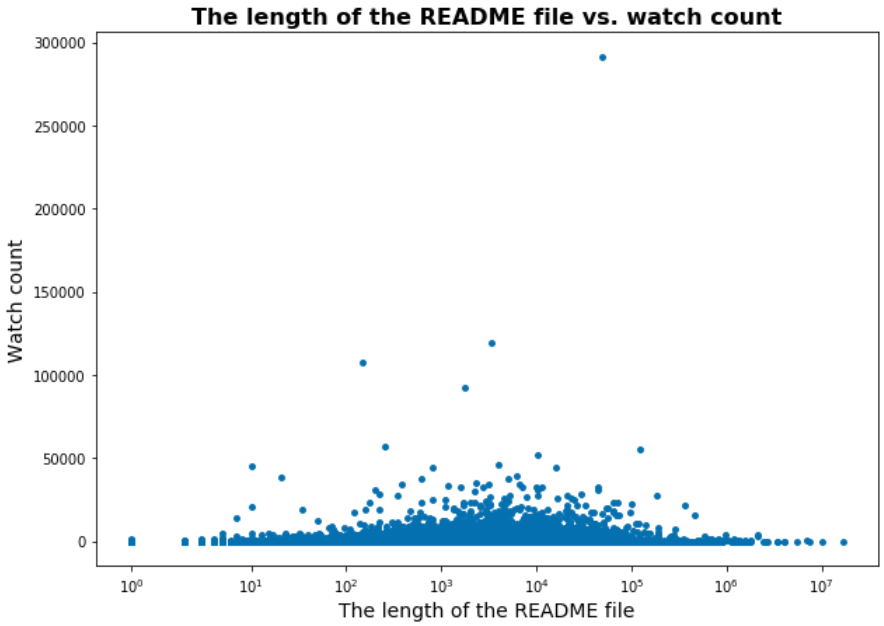
You may ignore README files with length 0.

Note: If a project has multiple `README` files, you can just take the average size of those files.

```
%%bigquery --project $project_id q5b

SELECT table.average_size AS readme_size, watch_count AS cnt
FROM `cs145-fa19.project2.github_repos` r
JOIN (SELECT f.lrepo_name AS selected_reponame, ROUND(AVG(c.size)) AS average_size
  FROM `cs145-fa19.project2.github_repo_files` f
  JOIN `cs145-fa19.project2.github_repo_readme_contents` c
  ON f.id = c.id
  GROUP BY f.lrepo_name
  HAVING ROUND(AVG(c.size)) != 0.0) table
ON r.lrepo_name = table.selected_reponame
```

```
plt.figure(figsize=(10, 7))
plt.scatter(q5b["readme_size"], q5b["cnt"], color='#0570B0', s=15)
plt.title("The length of the README file vs. watch count",fontsize=16,fontweight='bold')
plt.xlabel("The length of the README file", fontsize=14)
plt.ylabel("Watch count", fontsize=14)
plt.xscale('symlog')
# plt.yscale('symlog')
```



c) Would you say that a "good" `README` is correlated with a popular repository, based on the features you studied? Why or why not? If you were to analyze more in-depth features on the `README` file for correlation with repo popularity what would they be? (3 points)

No, a "good" README is not strongly correlated with greater watch count. However, illustrated by Question 5(a), average watch count differs slightly between the repos with README file and without a one. Having a README file is likely to be watched by more people to some extent. On ther other hand, Question 5(b)'s plot is chaotic, not producing a typical pattern for positive or negative correlation. If I were to analyze more in-depth features on README, I would consider the content of README. Maybe I should classify some general categories for the content and find any correlation between the topics of contents and average watch count.

Question 6 (Extra Credit): What other features might correlate with a highly watched repo? (3 possible points)

We studied only a handful of features that could correlate with a highly watched repo. Can you find a few more that seem especially promising? Back your proposed features with data and charts.

---

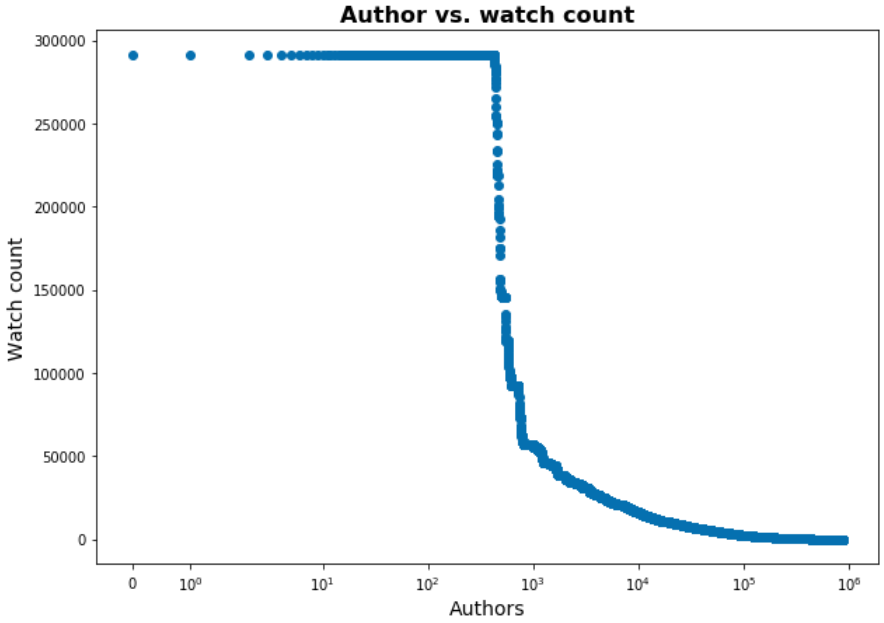
*Basically, I explored two additional features that might have correlation with a highly watched repo. The first one is author's name. Some authors might be very reputable and influential in certain fields. So, this group of authors probably have lots of followers, which introduces greater popularity of their repos. I plotted the author against watch count of their repos by running query q6a. However, for the most around 120 popular authors, their average watch count is nearly the same. Then, watch count differentiates drastically across different authors. This can indicate that if one is the very famous author, his or her share of watch count is stable. For less popular authors, their average watch count declines notably.

The second feature I explored is the number of files per repo. However, the scatter plot is quite noisy, which does not demonstrate any positive correlation or negative correlation.*

```
%%bigquery --project $project_id q6a

SELECT author.name AS author_name, AVG(watch_count) AS cnt
FROM `cs145-fa19.project2.github_repo_commits` c
JOIN `cs145-fa19.project2.github_repos` r
ON c.lrepo_name = r.lrepo_name
GROUP BY author.name
ORDER BY cnt DESC
```

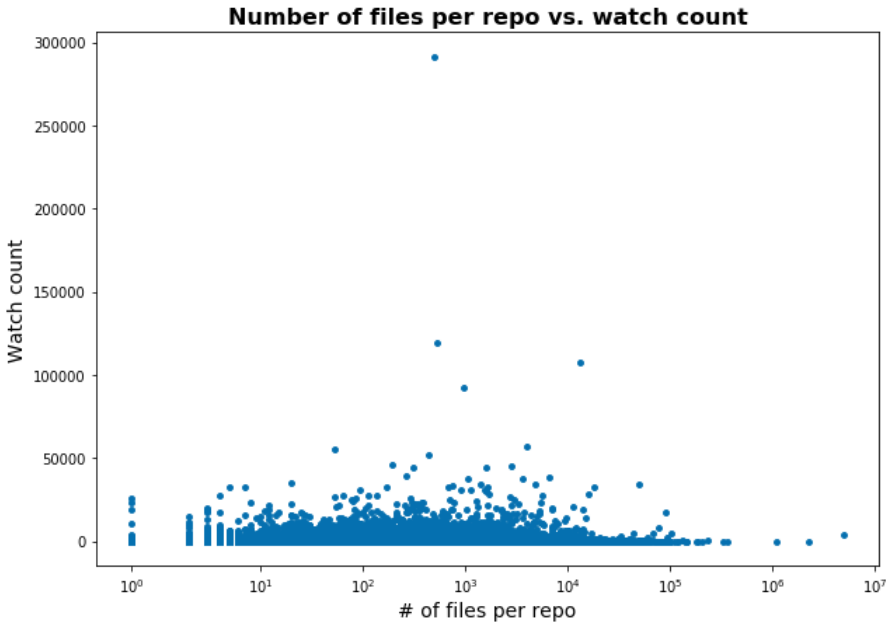
```
plt.figure(figsize=(10, 7))
plt.scatter(q6a["author_name"], q6a["cnt"], color='#0570B0')
plt.title("Author vs. watch count",fontsize=16,fontweight='bold')
plt.xlabel("Authors", fontsize=14)
plt.ylabel("Watch count", fontsize=14)
plt.xscale('symlog')
# plt.yscale('log')
```




```
%%bigquery --project $project_id q6b

SELECT table.file_count AS file_cnt, watch_count AS watch_cnt
FROM (SELECT f.lrepo_name, COUNT(f.id) AS file_count
  FROM `cs145-fa19.project2.github_repo_files` f
  GROUP BY f.lrepo_name) table
JOIN `cs145-fa19.project2.github_repos` r
ON table.lrepo_name = r.lrepo_name
```

```
plt.figure(figsize=(10, 7))
plt.scatter(q6b["file_cnt"], q6b["watch_cnt"], color='#0570B0', s=15)
plt.title("Number of files per repo vs. watch count",fontsize=16,fontweight='bold')
plt.xlabel("# of files per repo", fontsize=14)
plt.ylabel("Watch count", fontsize=14)
plt.xscale('symlog')
# plt.yscale('symlog')
```


Number of files per repo vs. watch count
Watch count
# of files per repo
300000
250000
200000
150000
100000
50000
0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7