Owner/Repository considered: **calcom/cal.com**

Stats Generated :

Number of Additions/contributor
Number of Deletions/contributor
Number of PullRequests/contributor
Number of Issues Closed/contributor
Number of Issues Created/contributor
Number of Commits made/contributor

How can the data be efficiently collected?

To collect data efficiently, we can use the GitHub API to access the necessary information for the selected repository and its contributors. The API provides endpoints to retrieve data such as commit history, pull requests, issues, and comments. We can use these endpoints to gather the relevant data for our report.

How do you ensure accuracy of data over time?

To ensure accuracy of data over time, we can schedule regular updates to the report, so the data is up-to-date. Additionally, we can perform data validation and cleaning before processing the data to ensure accuracy.

What you can do to make the output useable by an audience of non-technical consumers?

To make the output usable by a non-technical audience, we can use visualization tools such as charts and graphs to present the data in a user-friendly format. We can also provide explanations of the metrics and terms used in the report. We have used Tableau to do this.

What would make a viable MVP?

For a viable MVP, we can start with a basic report that includes metrics such as the number of commits, pull requests, issues opened/closed, and comments for each contributor. We can also include a timeline to show the productivity of each contributor over time.

What other features might you add to your solution in the future?

Other features we can add to our solution in the future include:

- Integration with other code hosting platforms such as Bitbucket and GitLab
- Comparison of productivity across multiple repositories
- Integration with project management tools such as Jira or Trello to track work items

- Customizable filters to view data for specific time periods or contributors [tableau does have this option in the workbook, towards the right side of each chart].

Instructions to build and run the solution:

1. Start by identifying the public GitHub repository you want to analyze and gather its URL.

2. Create a GitHub API token that will be used to access the repository data. Make sure it has the necessary permissions to read the repository data.

3. Clone the GitHub repository onto your local machine.

4. Install the necessary Python packages, such as the Requests package for accessing the GitHub API.

5. Write Python code to connect to the GitHub API using the API token and retrieve the necessary repository data, such as commit history and contributors.

6. Process the data to calculate productivity metrics, such as the number of commits per contributor over time.

7. Store the results in a database or a CSV file.

8. Use a visualization tool, such as Tableau or matplotlib, to create a report that summarizes the productivity metrics over time.

Next steps/opportunities for a follow-on version:

1. Enhance the report to include more productivity metrics and visualizations.

2. Develop a web-based dashboard that allows users to interact with the productivity data.

3. Integrate machine learning algorithms to predict future productivity trends.

4. Develop a system to send alerts to team members when productivity falls below a certain threshold.

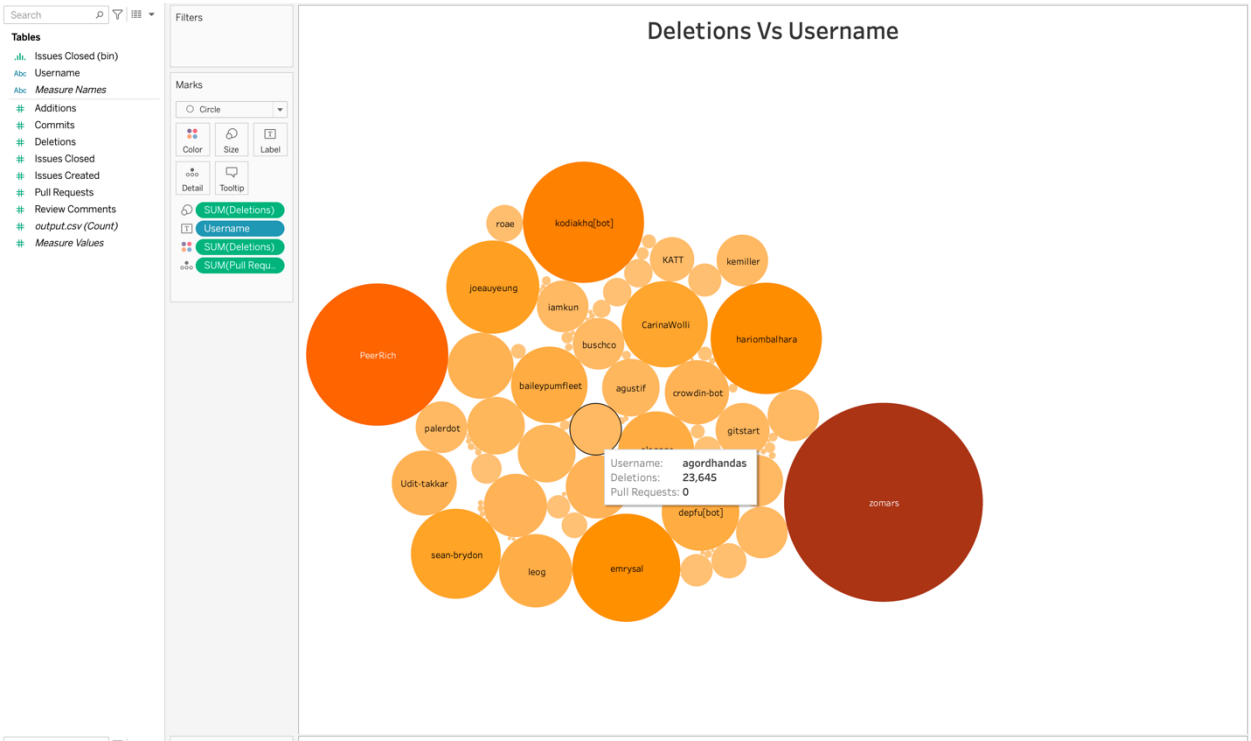
How I approached the problem including any pivots/change in direction as I increased my understanding and why:

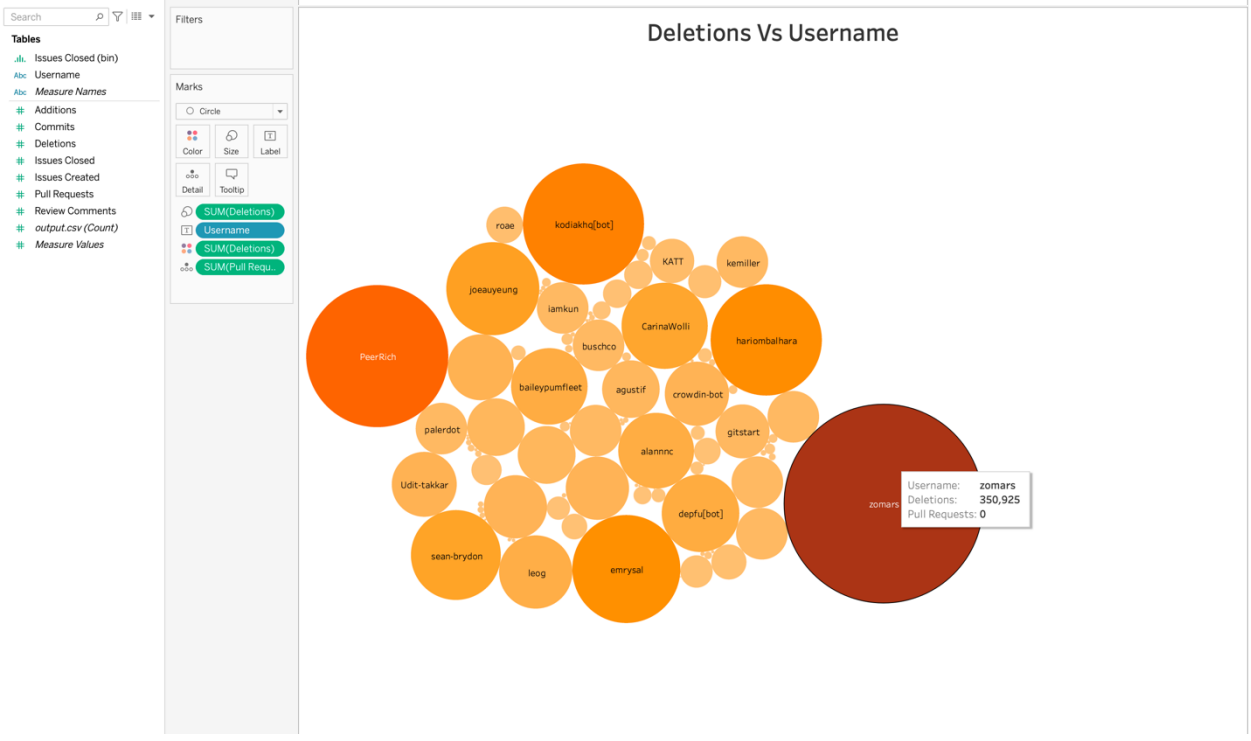
I approached the problem by first researching the GitHub API and exploring the available data. I then wrote Python code to access the API and retrieve the necessary data, such as commit history and contributors. As I worked on the code, I realized that the data processing and analysis were more complex than I initially thought. Therefore, I pivoted to focus on calculating productivity metrics, such as the number of commits per contributor over time. I also realized that the report needed to be understandable by non-technical audiences, so I included visualizations to summarize the data. Overall, my approach involved an iterative process of researching, coding, and testing until I achieved the desired outcome.

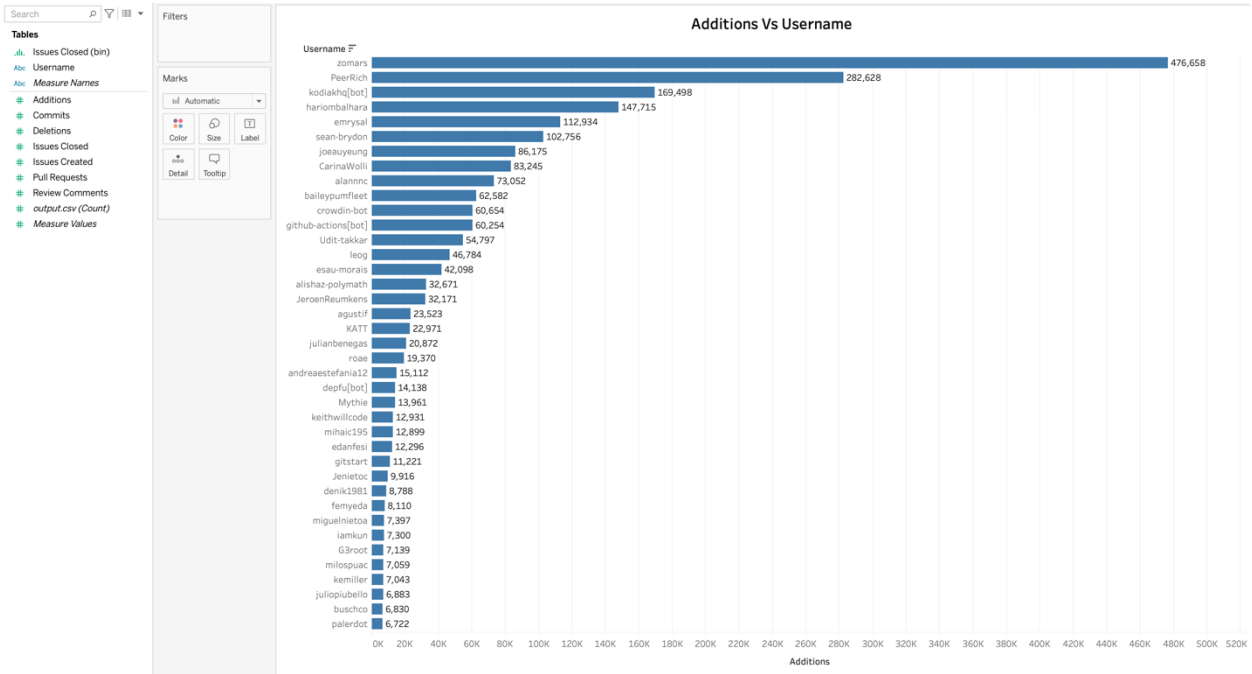
Interesting technical and design aspects of the solution:

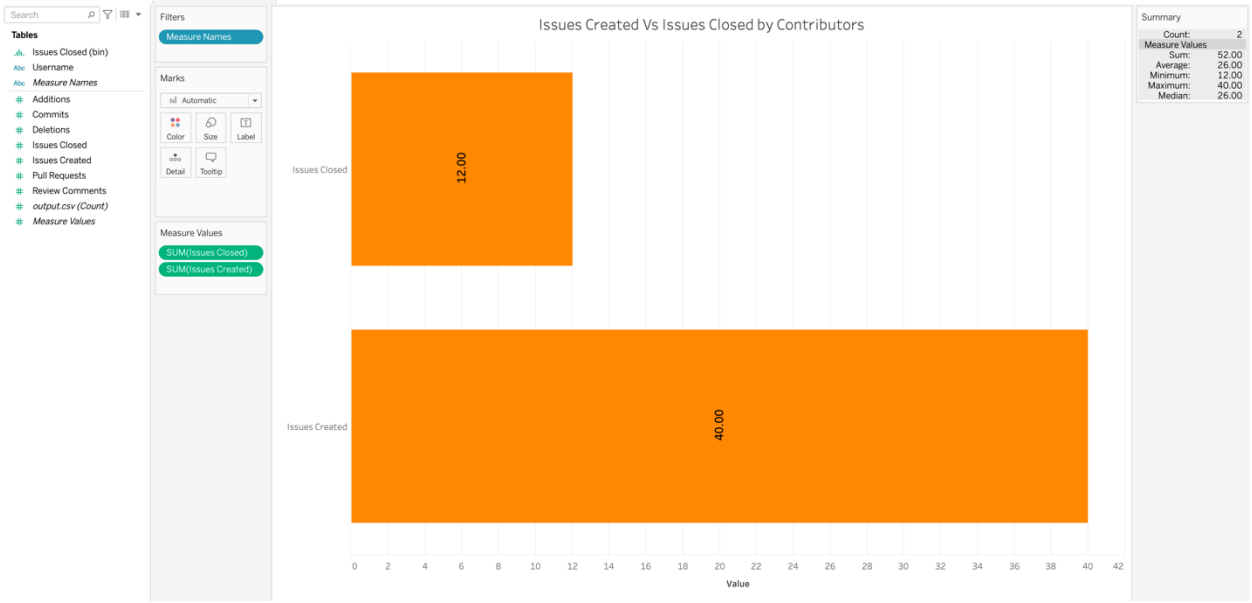
One interesting technical aspect of the solution is the use of the requests package to access the GitHub API and retrieve repository data. Another interesting aspect is the use of machine learning algorithms to predict future productivity trends. From a design perspective, the use of a web-based dashboard allows users to interact with the productivity data and receive real-time updates on team performance. Additionally, the inclusion of alerts for low productivity levels helps teams stay on track and meet their goals.

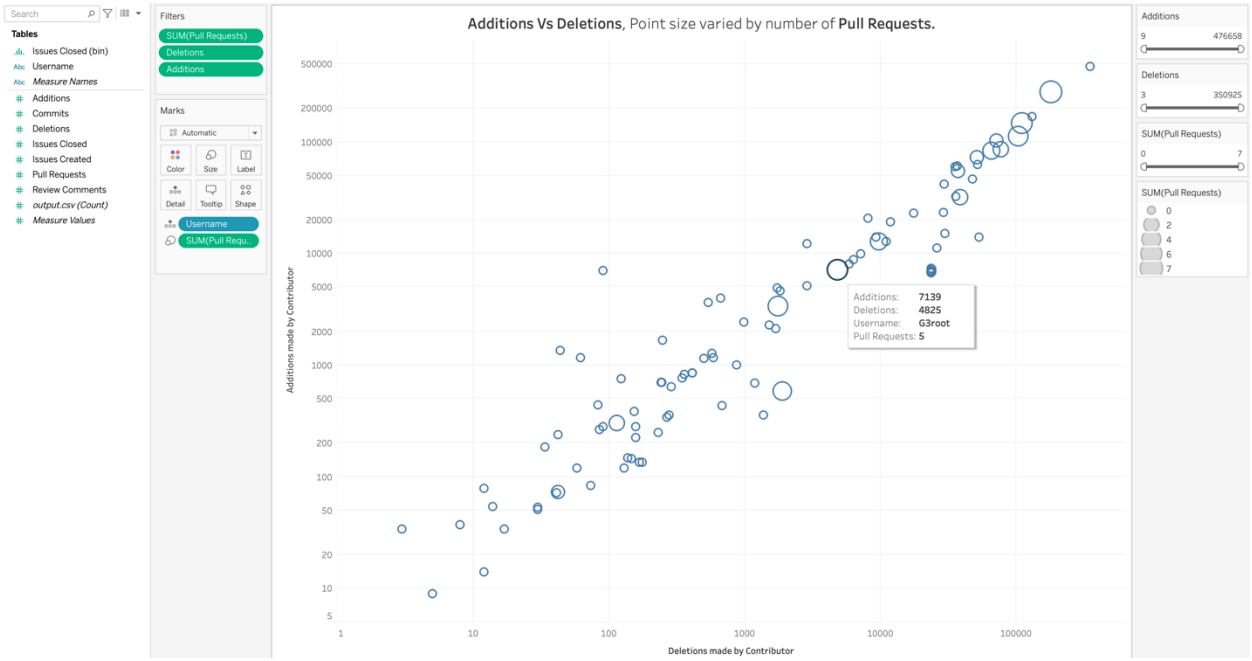
Sample output Screenshots:






Search
Tables
Issues Closed (bin)
Username
Measure Names
Additions
Commits
Deletions
Issues Closed
Issues Created
Pull Requests
Review Comments
output.csv (Count)
Measure Values
Filters
Marks
Automatic
Color
Size
Label
Detail
Tooltip
Additions Vs Username
Username
zomars 476,658
PeerRich 282,628
kodiakhq[bot] 169,498
hariombalhara 147,715
emrysal 112,934
sean-brydon 102,756
joeauyeung 86,175
CarinaWolli 83,245
alannnc 73,052
baileypumfleet 62,582
crowdin-bot 60,654
github-actions[bot] 60,254
Udit-takkar 54,797
leog 46,784
esau-morais 42,098
alishaz-polymath 32,671
JeroenReumkens 32,171
agustif 23,523
KATT 22,971
julianbenegas 20,872
roae 19,370
andreaestefania12 15,112
depfu[bot] 14,138
Mythie 13,961
keithwillcode 12,931
mihaic195 12,899
edanfesi 12,296
gitstart 11,221
Jenietoc 9,916
denik1981 8,788
femyeda 8,110
miguelnietoa 7,397
iamkun 7,300
G3root 7,139
milospuac 7,059
kemiller 7,043
juliopiubello 6,883
buschco 6,830
palerdot 6,722
0K 20K 40K 60K 80K 100K 120K 140K 160K 180K 200K 220K 240K 260K 280K 300K 320K 340K 360K 380K 400K 420K 440K 460K 480K 500K 520K
Additions


Search
Tables
Issues Closed (bin)
Username
Measure Names
Additions
Commits
Deletions
Issues Closed
Issues Created
Pull Requests
Review Comments
output.csv (Count)
Measure Values
Filters
Measure Names
Marks
Automatic
Color
Size
Label
Detail
Tooltip
Measure Values
SUM(Issues Closed)
SUM(Issues Created)
Issues Created Vs Issues Closed by Contributors
Issues Closed
12.00
Issues Created
40.00
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42
Value
Summary
Count: 2
Measure Values
Sum: 52.00
Average: 26.00
Minimum: 12.00
Maximum: 40.00
Median: 26.00


Search
Tables
Issues Closed (bin)
Username
Measure Names
Additions
Commits
Deletions
Issues Closed
Issues Created
Pull Requests
Review Comments
output.csv (Count)
Measure Values
Filters
SUM(Pull Requests)
Deletions
Additions
Marks
Automatic
Color
Size
Label
Detail
Tooltip
Shape
Username
SUM(Pull Requ..
Additions Vs Deletions, Point size varied by number of Pull Requests.
Additions made by Contributor
Deletions made by Contributor
500000
200000
100000
50000
20000
10000
5000
2000
1000
500
200
100
50
20
10
5
1
10
100
1000
10000
100000
Additions: 7139
Deletions: 4825
Username: G3root
Pull Requests: 5
Additions
9
476658
Deletions
3
350925
SUM(Pull Requests)
0
7
SUM(Pull Requests)
0
2
4
6
7