# Network Effects Regarding Open Source Software Adoption

Lars Erik Schonander

The George Washington University

10/12/2019

**Abstract:**

Using a standard OLS regression, I create a model for the popularity of open source software projects. I use a basket of repositories GitHub with data from GitHub to determine what determine the popularity of a package. The baseline model has a $R^2$ of .27 with the model that includes an additional variable, Forks having a $R^2$ of .70. However, large amounts of endogeneity is present in all the regression models raises further questions how these variables are interlinked with each other.

## I Introduction:

Network effects since the rise of the internet have influenced what tools and platforms we have adopted. For example, in the past, there was a conflict between the browser Netscape and Internet Explorer during the dotcom boom. While Netscape was the more innovative browser because the company behind it Mosaic invented client-side scripting, they still lost to Microsoft's Internet Explorer. To put it briefly, because Microsoft controlled what software was distributed with Windows computers, they could automatically make everyone who bought a Windows computer a user of Internet Explorer. They then made Internet Explorer free, unlike Mosaic, who to make money, had to charge money for people to use Netscape.

For reference, a network effect is a phenomenon described in economics about goods or services that gain additional value when more people use it. A classic example of this has to with telephones, where the more telephone users exist, the greater utility each individual telephone owner has from owning a telephone. A real time of this phenomena appears with social networks, where the more people use a social platform, the more utility each user of the service gets. Software because of how it's produced, large upfront cost with zero marginal cost to copy, have large network effects specifically what is known as two-sided network effect. In this case, there are two sides, developers and users, where people adopt software based not only on the software, but what is possible to install and/or develop. An example of this type of network effect would appear with operating systems such as Microsoft & Linux for computers, or Android & iOS on mobile phones.

If one is not involved the field of computer science or a practitioner, it is likely that they have never heard of the idea of open source software[1]. That is, software that is produced for free, and is free to distribute and modify by anyone who has the technical skill to work with the software. However, the world in practice runs on open source software. From open source databases like SQLite, which are present on every iPhone in the world, frameworks like React.js, which many websites such as Facebook and Twitter are built with, to software that powers artificial intelligence such as TensorFlow or Keras, open source software is present everywhere. Network effects play a large part in how open source software is adopted, because the more people use it, developers gain a larger utility as the software will likely receive updates from new developers that expand the software capabilities. An example of this being the many component libraries from React.js an open source software framework for creating websites. As more people use React.js, developers decided to develop on the platform, creating new forms of open source software that require React.js, such components that make maps or calendars. This means that the capabilities of React.js increase as the ecosystem of React.js related tools grows larger.

Lack of awareness of this field is understandable, because of its technical nature. However, open source software represents a microcosm in that many of the concerns that people have about major organizations is mirrored at the level of individual repositories. This ranges from worries how corporate sponsorship of open source software project is stifling projects not sponsored by corporations, to how important projects, despite being critical for the internet to run, are run by people who are paid very little. These developers are also overwhelmed with requests from developers who use the open source software. The inventor of the open source

[1] Will be referred to interchangeably as OSS for the sake of not writing things out.

key-value database Redis, Sanfilippo (2019)[2], writes about many stresses he faces from maintaining the database. These stresses range from the overwhelming number of requests he gets, to the above hours of work he has to do because his full-time job consists of him being paid to maintain Redis.

The paper is organized into several subsections. First, a literature of different papers on network effects and open source software, split between pre and post 2000. Next, a methodology section that explains how I collected the data for the regression, along with an explanation of my regression framework, along with summary statistics and graphics for the variables I am looking at. The section after that presents the model and explains the output. The final section concludes the paper, with an appendix that has some caveats I had when I was writing the paper. Using several OLS regression models, this paper examines the impact that individual contributors, updates, corporate sponsorship, and the type of software correlate to how popular an individual open source software project is. My question is, what are the factors that make an open source software project popular?

## II Literature Review

While open source software has only been studied in later years, the study of networks has been a part of economics in the 1970's. In fact, the study of network effects is important to learn the history of open source technology because network effects strongly influence what technologies get adopted. In the 1990's and particularly after, economist begin to study the production of open source software, such as how it is distributed, and how does it compare to

[2] Redis was initially published in 2009. The 2019 refers to in the bibliography a piece he wrote on his experiences maintaining open source software.

software produced by corporations. Unfortunately, papers that look at actual datasets on the production of open source software are rare, requiring me to create my own dataset.

I split the literature review into two sections, pre 2000 and post 2000. The reasoning for this is twofold. First, the pre 2000 period is where much of the foundational literature on network effects was written, so I wanted to ensure before the paper goes into network effects regarding software, the literature review has explained what a network effect exactly is. Secondly, the split reflects the predominance of software within our lives. With the pre 2000 portion being more on economics and theorizing on how software interacts in the world. This is combined with a brief glance of the dotcom boom, to help explain how network effects may come into conflict with each other. The post 2000 period represents an evolution in the literature, considering that after this period many more people have access to the internet, and are as such able to interact with the open source community more easily.

**Pre 2000**

The introductory paper to the field sets the tone of how the field is studied. Rolfs (1974), sets the foundational theory of the field in his paper by stating that: *"The utility that a subscriber derives from a communications service increases as others join the system"*. While a simple assumption, this idea has been cited and used for other papers in the economic study of network effects. The subject of this paper is talking about network effects regarding telecommunication systems, along with different methods to bootstrap a network. Telecommunication systems are seen as a classic example of network effects working mainly as the more people own telephones, the more benefit an individual earns by owning a telephone. One method he states to build up a network is to find user group that requires a network to do what the group is interested on. This

paper influences the variables I use in my paper, mainly by looking to see how many people have contributed to an open source software project.

Economides (1992) then goes into more detail about the economics of networks by specifically noting the fact that networks can be competing against each other. This is because even though two separate products might do the same thing, they can compete over the same network effect. This leads to companies making their goods or software incompatible with each other, so rival companies don't benefit from network effect. A contemporary example of this would the rivalry between different social media networks, such as Snapchat vs Instagram. While each gains more utility the more users are on the platform, the other platform doesn't gain any extra utility for its rival gaining. This leads to competition between the platforms to try to make sure users are not on the other platform, ranging from copying features, or making users who use both platforms have a worse experience then those who stick to one .This can be applied to the production of OSS tools, as while they are meant to do similar tasks, in practice they are incompatible, and users tend to use one standard over another standard. This can be seen in practice in conversations on technology where the discussion of what software is better can lead to vicious arguments.

Varian (1998) presents a practical example the idea of a conflict between different standards competing for the same market with the idea of a standard war. A standard was is a conflict between two different goods that are competing over the same network. This can range from browsers competing over user share, or companies producing OSS deep learning software to convince developers to use their framework, and not the other company's framework. Varian (1998) finds that standard wars tend to be violent because having control over a network effect can be very profitable. He finds that the end result of a standard war tends to be a compromise,

or complete market domination by one of the participants in the standard war. The example that Varian (1998) uses was the contemporary at the time conflict between Microsoft's browser Internet Explorer and Netscape's browser Netscape Navigator. This ended with Netscape was driven out of business because Microsoft decided to bundle Internet Explorer with Microsoft operating system, making Internet Explorer free, along with providing a large population of users for Internet Explorer automatically. This ended up with Internet Explorer achieving peak market share from browsing software, with a peak of 96% of internet users using Internet Explorer as their browser in 2002.

**Post 2000**

More recent papers after the 2000's go more into details about the practically of OSS production, and provide a better base for explaining the day to day details about how OSS is produced. Benkler (2006) writes about the idea of commons-based peer production, using OSS as a key example of how this could possibly be done. He talks about some of the components of this economy, that is will come about because of tools that promote collaboration, along with suggesting reasons why people create goods other than because of financial incentives. It does feel like Benkler (2006), tries to tackle to much in his book by trying to understand the internet in general, instead of the production of open source software which is the first part.

This optimistic take on open source software production is critiqued however in a more in-depth economic analysis about a problem OSS has. Economides (2007) in a recent paper writes about the idea of psychic costs associated with OSS. The example he provides is all the learning required to install and use software on a Linux computer, versus a Windows computer. While Linux is free, in practice there are other costs associated with using the software, that being the installation and learning of software. The Windows user doesn't suffer these costs, as

the software and platform is designed to minimize psychic costs, by for example, not needing to install important software, or use a command line to do basic work on the computer. It will be interesting to apply Economides (2007) theory that corporate backed projects will be more popular by looking at corporate sponsorship as a variable in the paper's regression analysis.

## III Methodology:

To collect the data for the regression model, I spoke to a researcher at GitHub to help pick a basket of N = 54 major open source software projects that are hosted on the website. GitHub is a website that makes it easy for programmers to store their code and update it. It became a popular place to run open source software projects because of the community and the tooling made it easier for people to contribute these projects. The features of GitHub make the production of OSS more social as GitHub has features that make the website have the features of a social network, such as user feeds and the ability to "like" various repositories. These packages were chosen in particular because most software developers have either heard of them or used them in their work. There was not defined threshold for the variable used in the regression because preselecting projects that are popular defeats the purpose of discovering projects that may be commonly used, but not necessarily rated popularly on GitHub. This is mainly because there are projects that may be commonly used, such as SQLite[3] .To then collect the statistics required to do the analysis, I used GitHub's API to query the data, which is then stored in an Excel file. I extracted the *Stars*, *Commits*, *Contributors* and *Forks* variables from the GitHub

[3] A SQL database which the main utility is that SQLite databases can be created as .db files, and one does not need a server to run them. This allows them to be embedded everywhere. For example, iPhone's use SQLite databases to store some types of data.

API for each individual repository. I  then created two variables with an input of a research, a dummy variable *corporate*, and a discrete variable *category*. These variables will be explained below in the regression framework section. One constraint in the data is that I had to collect it myself, because there is no definitive database that has this information collected, hence why I talked to a research to help create curate a selection of open source software projects hosted on GitHub.

**Regression Framework**

To understand how a package becomes popular, this paper uses an OLS regression model with the following variables

$$Stars_r = \alpha + \beta_1 Commits_r + \beta_2 Contributors_r + \beta_3 Corporate?_r + \beta_4 Category_r + \beta_5 Forks_r + \varepsilon_r$$

Where $Stars_r$ refers to how many stars of an individual repository *r* has as described above. An individual star represents someone liking the repository to follow it later, meaning that there is a 1 to 1 ratio between an individual and a star, an individual project cannot give out multiple stars to the same project. For a better mental model, think of a star simply like a like on a social media platform. $Commits_r$ refers to how many commits a repository received, representing how many times the codebase was updated. Every time someone writes new code that gets added to the repository, a commit is created that lists all the changes they made to the repository. $Contributors_r$, refers to how many individuals contributed code to a repository *r*. An individual that commits code to a repository is called a contributor. $Corporate?_r$ is a dummy variable that for a given repository *r* that us 0 if the project is not run by a corporation, and 1 if the given repository *r* is an initiative by a corporation. Not all, but quite a few open source software projects are projects spun out from corporations that they are willing to give to people

for free to develop on. Members of the open source community are worried about these types of projects due to concerns that corporation sponsorship will corporatize the ecosystem and dominate non corporate sponsored projects even if the project is not of the same quality. $Forks_r$ represents how many times a given repository was forked. To fork a repository means merely to make a copy of the master repository. Most forks are used to be the starting point of a new project, or to a place to propose changes to the original repository Finally, $\alpha$ is the constant term and $\varepsilon$ is the error term. The key terms I am looking at are *Commits*, *Contributors*, and *Forks* as they represent how many times a repository has been updated, how many unique individuals have contributed to a repository, and how many times a repository has been copied respectively.

Stars is being used as the left-hand variable as it is being used as a stand in for popularity. Given that there is not direct indicator of how widely used a software package is, the model used stars as a heuristic. This paper decided on doing this because many users on GitHub user stars to show that they like a project. As this measure would be the equivalent on a social media platform of a 'like' and limited measures of popularity exist, *stars* works well enough for measuring popularity.

**Data Analysis:**

To begin with data analysis of the dataset of repositories (N = 54), below is a summary table of the dataset (Figure One). The main outlier being contributors with a min of one. This represents the code repository for Linux, because in practice the creator, Linus Torvalds, has to approve every update to Linux. This also causes the right-skew for Mean and Standard Deviations for the commits and contributors' column. The median highlights this rightward skew, considering for *Stars* its 20303, *Contributors* it's 536, and for *Commits* it's 25113.5. This is an important reminder to look at both the mean and the median, as often times the mean can be

skewed. The range for commits is so large because some projects have a policy of automating commits, leading to large amounts of commits daily, versus' other projects which may only do a few commits a day, leading to projects with equal star counts in practice having different amounts of commits. *Stars* is the largest variable simply because it is the easiest action to perform on a repository, as it merely requires the clicking of a button. All other actions require somewhat more effort to be counted.

Below are several tables (Figure Two)(Figure Three) of grouping statistics for the categorical variables *corporate* and *category* respectively, along with charts to visualize the data. These tables and charts are at the very end of the paper.

I create a dummy variable for the dataset *corporate*, that takes the value of 1 if the OSS project is backed primarily by a corporation, and 0 if the project is independently run. The range between mean star counts for corporate versus non-corporate project is 8221. This may seem like a lot, but in practice during the regression analysis, this variable proves to be not significant.

I used a categorical variable to measure what type of project is. If the project is primarily for a tool on the server-side, it is 1 and backend[4], 2 if the project is a data science tool, 3 if the project is a database, 4 if the project is mainly on the frontend/client side, and 5 if the project lacks easy categorization but is important in a developers daily work[5]. This is based on a personal hypothesis that some projects might be less popular because their technical complexity makes it harder for new users to contribute in some way to that particular OSS project.

[4] Front End refers to everything that is available to a user when looking at a webpage on a browser. Backend refers to everything that is on the server side, for a web application that being a web framework tool such as Flask in Python or Ruby on Rails in Ruby.
[5] This ranges from utilities such as CURL, which lets one make HTTP requests in bash, to projects such as Bitcoin, which lack easy classification.

For the continuous numeric variables (*Stars*, *Commits*, *Contributors*, *Forks*) to visualize their distribution, I made a histogram for each, mainly to help visualize the fact that except for a few outliers, each of these continuous variables is left-skewed in practice, in spite of the skew in the Commits and Contributor variables caused by Linux being in the dataset.

First, I visulized the distribution of the *stars* variable. As seen above, the distribution follows a leftward skew where most packages have less then 50,000 stars, with only a few having more then 100,000 stars. Next, I visulized the *commits* and *contributors* variables. Like stars, both of these variables have a leftskew, with the majority of values being distributied on the left.

To test multicolonarity in my dataset, I regressed contibuters against commits. There is no multicolinnarity because of the way that commits are contributed. As only a few people in practice make a majority of the commits, the amount of contributers doesn't mean as much. Additionly, open source software projects have varying standards for commits. With Linux for example, the majority of commits come from a single person Linus Torvalds. While with other projects, such as TensorFlow, a robot actually makes many of the commits, as it used a way to aggergate commits coming in from Google.

## IV Results and Regression Analysis:

The results of the regression framework for the default framework as presented as the table below. I run an OLS regression to analyze the data. I test a total of five variables, with stars being my dependent variable, and *commits*, *contributors*, *corporate*, *category* and *forks* being the independent variables in this paper's regression analysis. The output of the regression is shown

below in (Table Four). The first five regressions do not include *forks*, as the regressions which run *forks* prove to change the analysis of the dataset quite a bit.

I originally just ran the model doing no transformations, other than turning *corporate* into a factor, but I ran a Breusch-Pagan and NCV[6] test to check for heteroscedasticity. As the NCV test returned a p-value of $p = 0.00001$, thus proved $p < 0.05$, meaning that heteroscedasticity was present in the dataset. To solve this problem, a modified model was required. To create a non-heteroscedastic model, I did a Box Cox Transformation on my dependent variable, which converted the distribution of the dependent variable *stars* into a normal distribution. A third model was created bases of the Box Cox Transformation model which is merely the modified *stars* variable along with *contributors*. Also, to test an assumption, the paper did a s similar transformation of *stars* by taking the log. In practice this had the same effect as the Box Cox Transformation.

Both regression analysis shows that in practice, the only significant variable is contributors, which has a positive correlation with the number of stars a repository has. As Table 3 Regression results shows, in the OLS regression, only contributors has a $p < 0.05$. While in the original regression commits being $p < 0.1$ can be argued to be marginally significant, as it is not below 0.05, in practice the influence of commits on star cart is marginal. For every additional contributor to an open source software project, that project has 12.893 more stars. This seems to tie back into Rolfs (1974), that people derive more utility from a service the more people use it. In this case, the more contributors to a project, the more features the project can have, and the project has more people on hand so solve problems users might be facing. In the regression

[6] A Non-Constant Error Variance Test. It is a test very similar Breusch-Pagan test that for testing for the same types of problems. Additionally, a link to the documentation of the function to prove this: https://www.rdocumentation.org/packages/car/versions/3.0-5/topics/ncvTest

where *stars* was transformed via the usage of a Box Cox Transformation, contributors is the only significant variable, with a p-value of 0.005.

Surprisingly, *corporate* is not a statistically significant variable. While at first one might think a project backed by a corporation would be more popular, given that they have more resources, in practice this is not the case. This seems to go against Economides (2007) research where he claimed corporate projects would be more popular because they lack the psychic costs that open source software projects have in exchange for not costing any money to use. *Category* is also not a statistically significant variable.

Figure 5, 6, and 7 shows the difference between the regular regression model and BoxCox transformation model, and base plus fork model respectively by comparing the regression to what a full correlation would look like. If the regression line was fully correlated with the dataset, the $R^2$ of the regression would be 1.00. But as seen as the $R^2$ for the original model is only 0.27, the variables in this regression only explain more than a quarter of the variance within the actual dataset. The $R^2$ of the BoxCox Model is 0.31, a .04 improvement over the original model, but that still means only a little less then a third of the variance is explained by this model. As for the smaller third model, the $R^2$ for that model is .22, meaning that 0.09 of the variances can be explained just by contributors. For the last model, the $R^2$ is .70, making it the most predictive model. This model will be explained in the next paragraph, as its employment was due to flaws in the previous model.

To test for endogeneity, I decided to use the Box Cox Transformation, and make *contributors* the dependent variable and the transformed *stars* variable an independent variable. This turned to show there was an issue with endogeneity, as the transformed *stars* variable was statistically significant with a P value of .005. To explore this issue further, I decided to add an

additional variable, *forks*. *Forks* not only significant with a p-value less than .05, the models which use Forks have much higher $R^2$, with the default model plus Forks have a $R^2$ of .70. Additionally, in this model, contributors turns out not to be significant. To then test for endogeneity again, the paper used the Box Cox Transformation Model, but made Forks the dependent variable like before. Like before, endogeneity is present in the model as *Forks* proves to be a significant variable. However, this can be explained in the actions that the average user of GitHub does. When looking at the statistics for *stars*, *contributors*, and *forks* there are far more *stars* then there are *contributors* or *forks*. To star a repository requires one simply to have a GitHub account, and click the star button on a given repository on GitHub. To contribute, one has to take the time to download the given repository, make changes, and then wait and see if the owner of the repository will accept one's changes into the given OSS code base. Additionally, in a user's workflow, one would usually like the repository before doing any additional actions, as the workflow of forking or contributing to a repository before liking it would be considered strange.

**Caveats**

One of the caveats I had was my regression model. Originally, my regression model was going to look like this.

$$Google\ Trends\ Score_r = \alpha + \beta_1 Stars_r + \beta_2 Commits_r + \beta_3 Contributors_r + \beta_5\ Corporate?_r + \beta_6 Category_r + \varepsilon_r$$

The left-hand variable, *Google Trends Score*, is a number gotten from the Google Trends dataset, which looks at the popularity of a given search term. I decided not to go with this model because while a time series regression model would be good to track popularity of a singular

repository (*r)* overtime, it was not a good model for looking at the popularity of a selection of repositories, as the way I collected this data was overtime, with a row for each week of the year, along with an additional row representing the year the data was collected.

Another caveat I have is  wish I had more data. The trouble with GitHub and GitHub's API is that the design of the website and API makes it difficult to actually collect summary statistics from individual repositories. I had to use stars as a heuristic for popularity, which is the closest variable I could find to approximate a popularity variable. The trouble is that outside of GitHub and its competitors, there are very few statistics available for open source software projects. The API works well for collecting data for individual projects, not so much for collecting data from many different projects, as the API requires the repository to be specified when querying for information.

I also realized at the last moment that I should add *Forks* as a data point. While adding the forks value for each repository did not take long, by adding *Forks* to the regression analysis, the outcome changed. For one, the $R^2$ of models that included *Forks* increased to .70. On the other hand, *contributors* when *Forks* was included ceased to be a significant variable.

**Multicollinearity**

Navodhya suggested that to test for multicollinearity, that I should run a regression for contributors against commits. I did exactly that to test for multicollinearity and confirmed that commits and contributors not only lack multicollinearity, but also don't even correlate with each other.

## V Conclusion

In conclusion, my OLS regression model for a selection of open source repositories shows that only contributors has a significant correlation at the $P < 0.05$ level. In contradiction to Economides (2007) findings, corporate sponsorship is not a statistically significant variable. However, following Rolfs (1974), analysis of network effects, open source software does follow the basic role that the more people that contribute to the software, the more people use it. However, as this model has an issue with endogeneity, as the last model shows that when *stars* and *contributors* are reversed, *stars* is a significant variable. This also applies when *Forks* is applied to the model, even if the predictive power of the baseline model increases as the $R^2$ of the new model is .70 . The endogeneity problem raises further questions to be explore that is it contributors that lead to a package getting more stars, or the other reverse, where the more stars a package has, the more contributors it has. However, looking at user workflow on GitHub, this issue is not important as much considering the case of someone forking the repository and linking it versus the reverse is a lot less common then the liking of a repository and then forking of it.

In the introduction, I ask what the factors that influence the spread of open source software. My research tells me that despite previous findings in the field of open source software, that corporate sponsorship is not a statistically significant variable. This seems contradictory to qualitative research I conducted on the field previously where concerns over the corporate takeover of the field seemed to be something on people's minds. If this paper does nothing else, at minimum I hope that this paper reveals a field that is under looked because of its technical complexity.

## VI Bibliography:

Benkler, Yochai, “The Wealth of Networks”, Yale University Press (2006)

Economides, Nicholas “Competition and Integration Among Complements, And Network Market Sturcture” The Journal of Industrial Economic, Vol XL, No.1 (March 1992) pp. 105-123

Economides, Nicholas “Two-Sided Competition of Proprietary vs. Open Source Technology Platforms and the Implications for the Software Industry” Management Science. Vol 52, No.7, (June 2007), pp. 1057-1071

Rolfs, Jeffery J. “A Theory of Interdependent Demand for a Communications Service” The Bell Journal of Economics and Management Science, Vol. 5, No. 1 (Spring, 1974), pp. 16-37 RAND Corporation

Sanfilippo, Salvatore “The Struggles of an Open Source Maintainer” antirez, 2019

Varian, Hal “Standard Wars” California Management Review Vol 41, No2 Winter 1998 University of California Press

## VII Figures:

Figure 1: Histogram of Stars

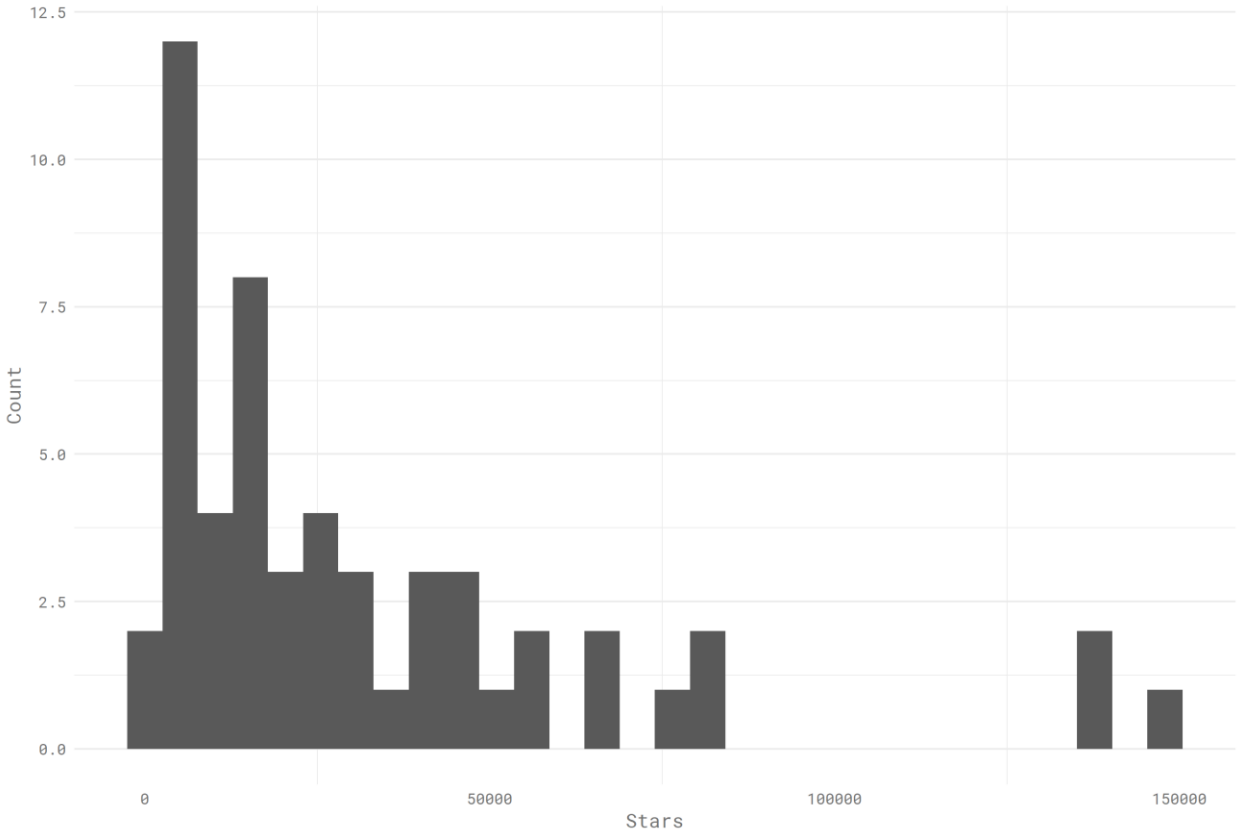


Figure 2: Histogram of Commits

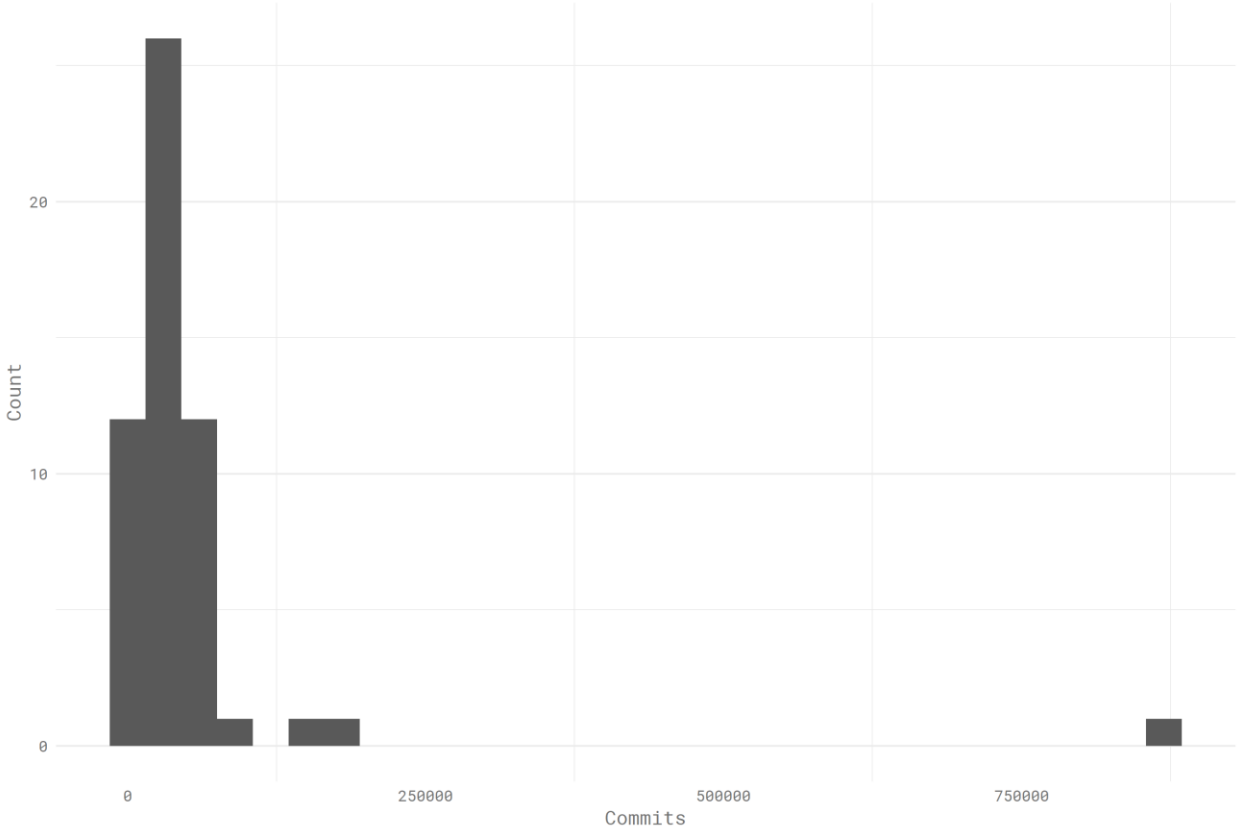



Figure 4: Histogram of Forks

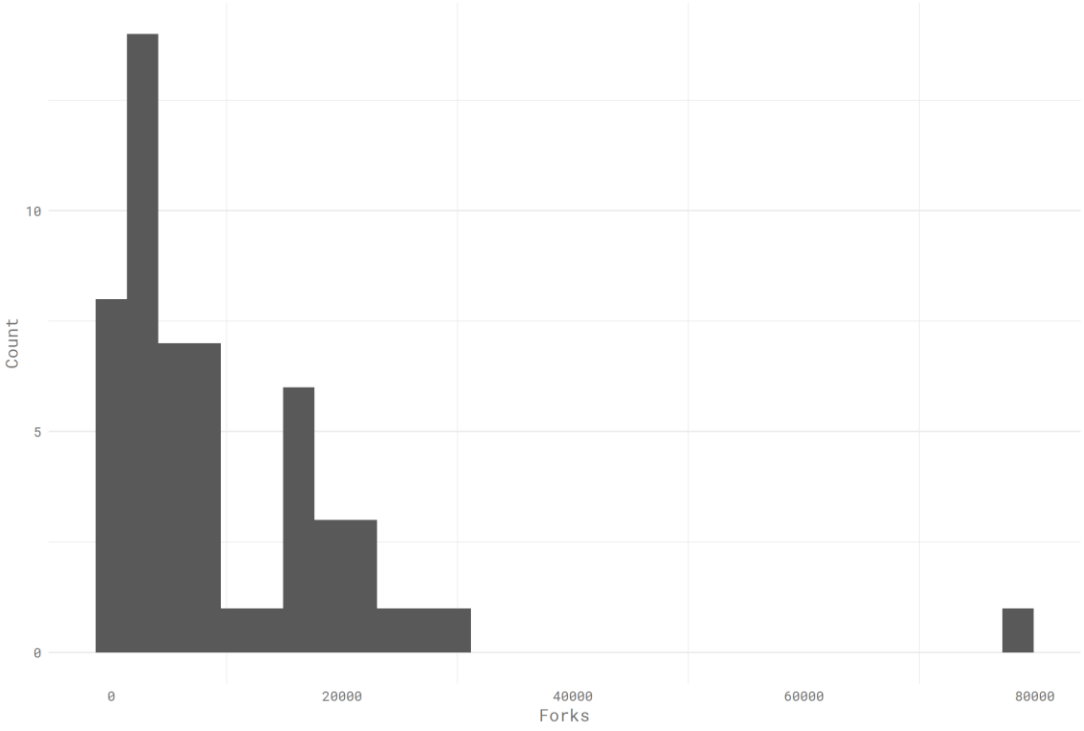


Figure 3: Histogram of Contributers

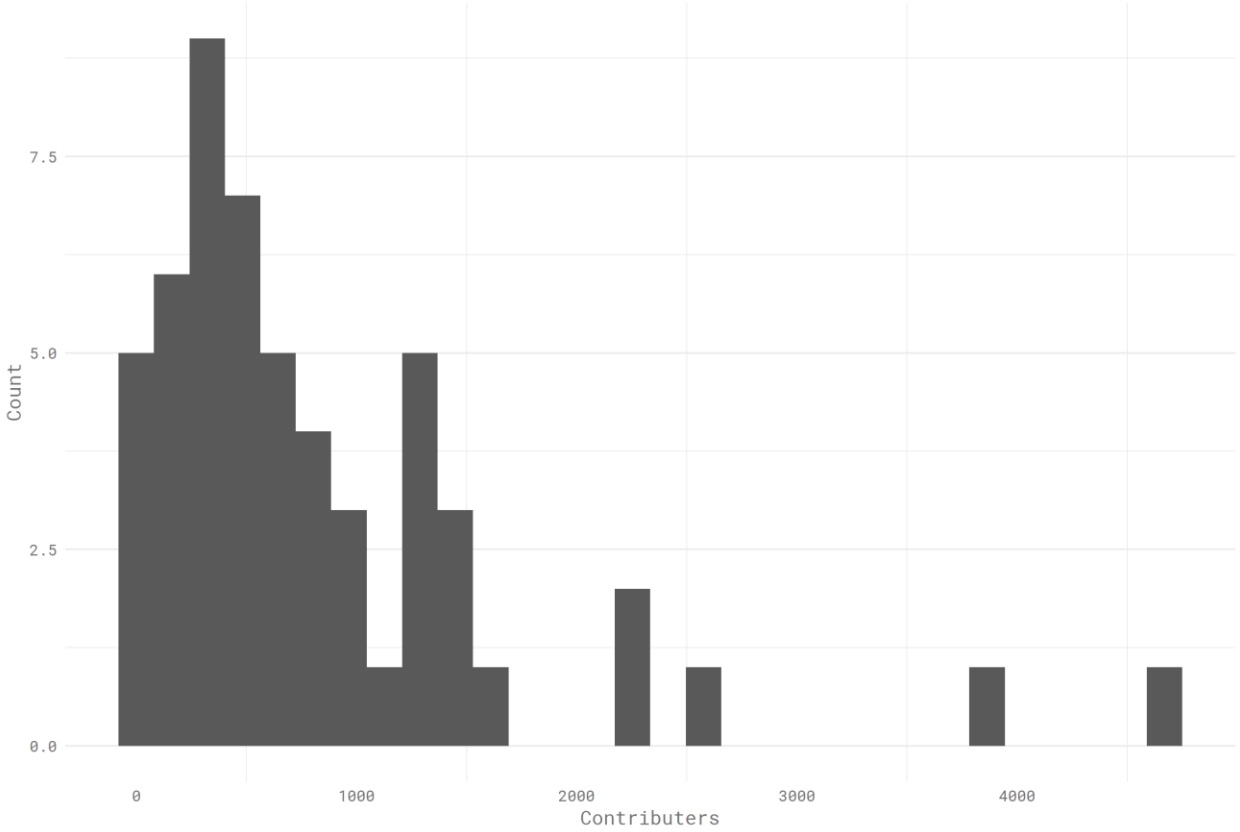



Figure 4: Histogram of Forks

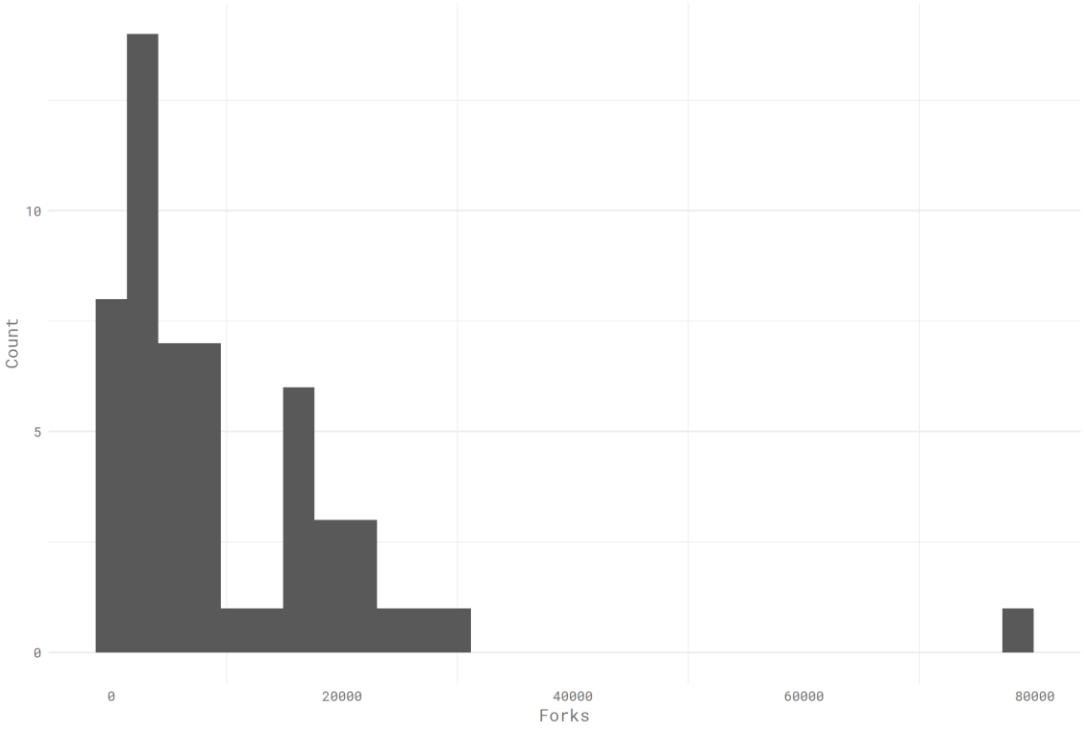


Figure 5: Regresion vs Full Correlation

Baseline Model

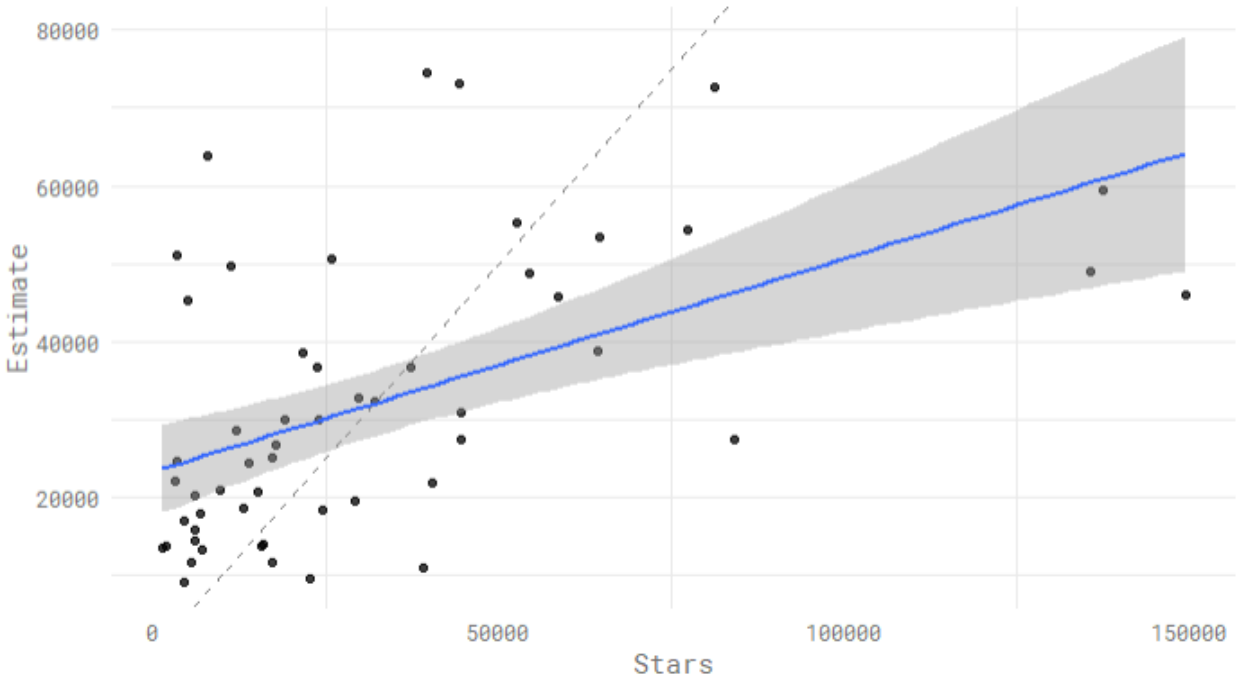
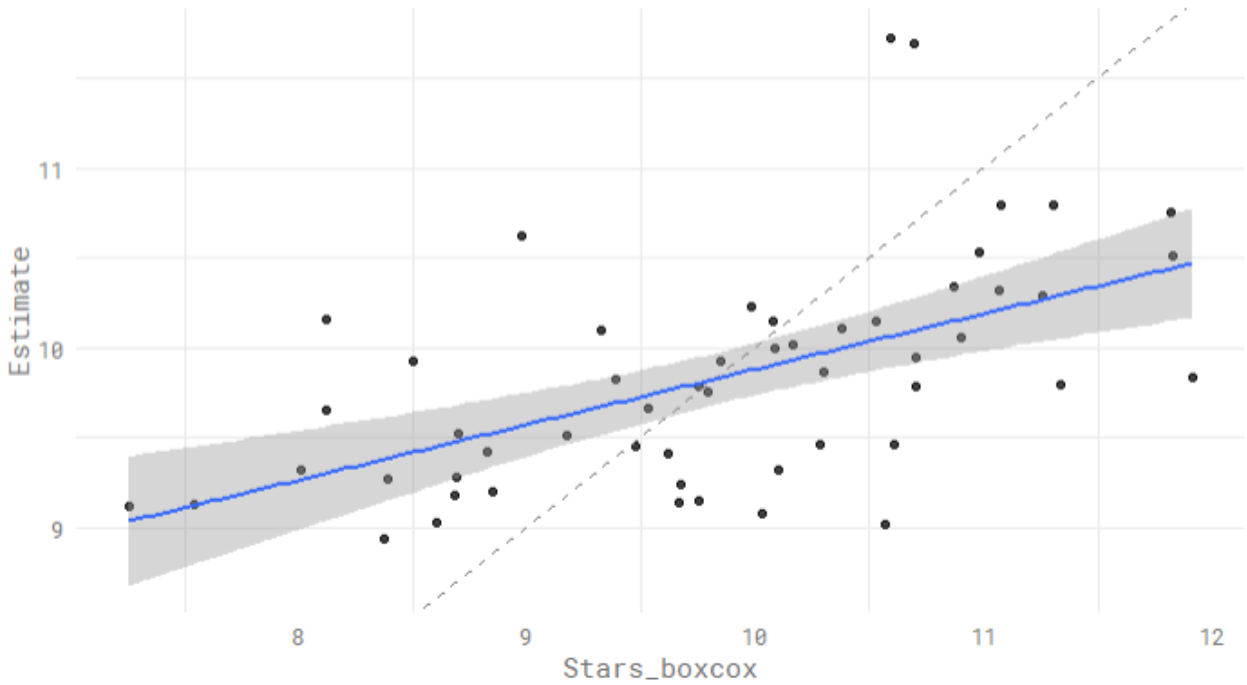
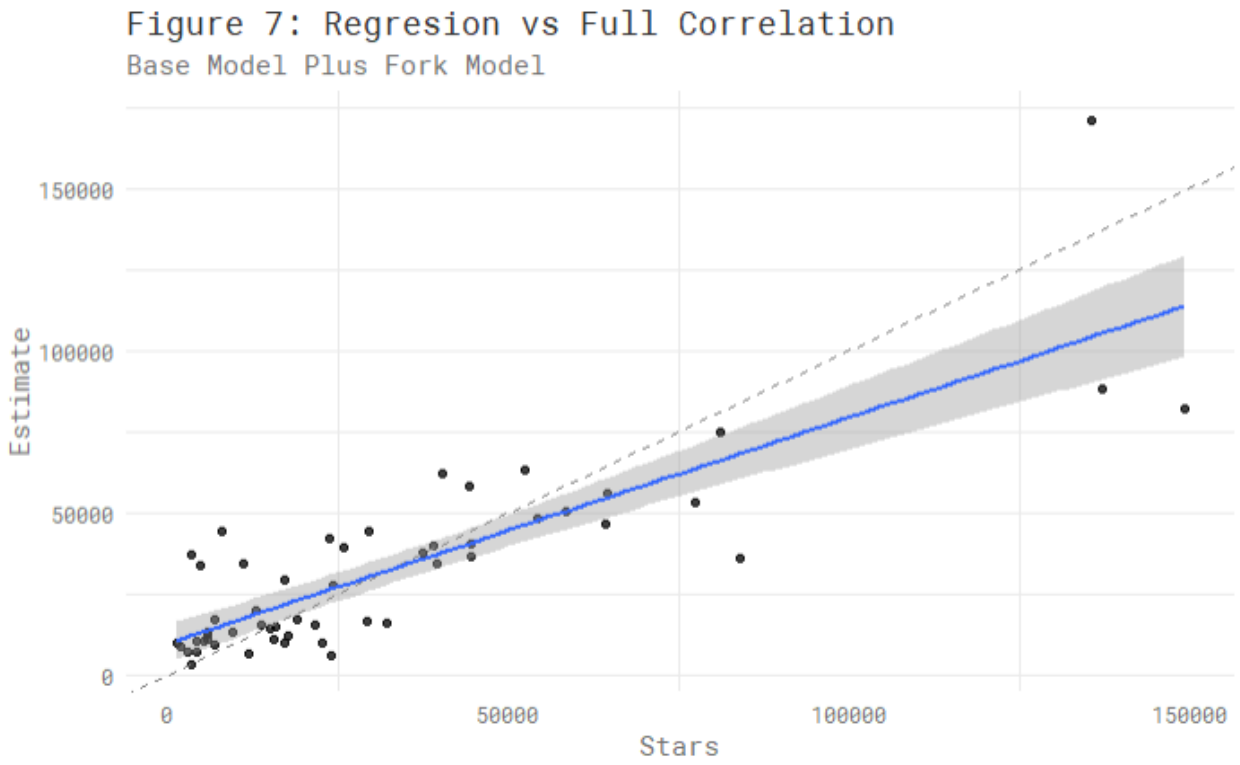



Figure 6: Regresion vs Full Correlation

BoxCox Transformed Model






## VIII Tables

**Table 1: Summary Table**

| Statistic | N | Mean | St. Dev. | Min | Pctl(25) | Pctl(75) | Max |
|---|---|---|---|---|---|---|---|
| Stars | 54 | 32,042.37 | 34,311.38 | 1,420 | 7,172.2 | 43,264 | 149,317 |
| Contributers | 54 | 843.19 | 906.67 | 1 | 275.2 | 1,227.2 | 4,670 |
| Commits | 54 | 51,405.35 | 118,485.30 | 1,715 | 16,725.5 | 47,826.8 | 871,239 |
| Forks | 54 | 9,915.69 | 12,369.46 | 246 | 2,050 | 15,575 | 78,800 |
| Stars_boxcox | 54 | 9.83 | 1.13 | 7.26 | 8.88 | 10.67 | 11.91 |

**Table 3: Category Variable Analysis**

| | Category | Mean Star Count | Median Star Count | Min Star Count | Max Star Count | Standard Deviation | Count |
|---|---|---|---|---|---|---|---|
| 1 | Back-End | 47564.25 | 54207.50 | 17177 | 64665 | 22378.18 | 4 |
| 2 | Data Science | 31271.50 | 22685 | 3371 | 135490 | 34886.19 | 12 |
| 3 | Database | 13189.88 | 6935 | 3013 | 38995 | 11923.35 | 9 |
| 4 | Front-End | 52408.20 | 39183.50 | 3362 | 149317 | 54024.56 | 10 |
| 5 | Infrastructure | 27472.73 | 18937 | 1420 | 84007 | 25150.93 | 19 |

**Table 2: Corporate Variable Analysis**

| | Corporate | Mean Star Count | Min Star Count | Max Star Count | Standard Deviation (Stars) | Count |
|---|---|---|---|---|---|---|
| 1 | No | 27627.08 | 1420 | 149317 | 32267.22 | 25 |
| 2 | Yes | 35848.65 | 3362 | 137283 | 36104.83 | 29 |

**Table 4: Regression Models**

| | *Dependent variable:* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Stars | Stars_boxcox | | log(Stars) | Contributers | Stars | Stars_boxcox | Contributers |
| | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) |
| Stars_boxcox | | | | | 294.90*** | | | 108.69 |
| | | | | | (101.27) | | | (127.10) |
| Commits | 0.07* | 0.0000 | | 0.0000 | -0.001 | 0.0003 | 0.0000 | -0.001 |
| | (0.04) | (0.0000) | | (0.0000) | (0.001) | (0.03) | (0.0000) | (0.001) |
| Contributers | 12.89** | 0.001*** | 0.001*** | 0.001*** | | -3.12 | 0.0001 | |
| | (5.66) | (0.0002) | (0.0002) | (0.0002) | | (4.21) | (0.0002) | |
| CorporateNo | 912.81 | -0.10 | | -0.10 | -359.72 | -655.61 | -0.14 | -356.73 |
| | (9,622.43) | (0.31) | | (0.31) | (224.59) | (6,282.88) | (0.26) | (215.12) |
| CategoryData Science | -2,306.26 | -0.06 | | -0.06 | -676.62 | -29,073.61** | -0.69 | -924.92** |
| | (19,288.35) | (0.62) | | (0.62) | (451.18) | (13,031.82) | (0.53) | (445.81) |
| CategoryDatabase | -12,382.84 | -0.56 | | -0.56 | -1,317.52*** | -23,351.74 | -0.82 | -1,396.93*** |
| | (21,410.25) | (0.69) | | (0.69) | (478.69) | (14,040.89) | (0.57) | (459.84) |
| CategoryFront-End | 23,413.84 | 0.28 | | 0.28 | -1,263.76*** | 2,516.65 | -0.22 | -1,331.99*** |
| | (20,195.68) | (0.65) | | (0.65) | (447.50) | (13,440.44) | (0.55) | (429.68) |
| CategoryInfrasturcture | -6,769.48 | -0.33 | | -0.33 | -877.37** | -20,340.66 | -0.65 | -995.14** |
| | (18,623.01) | (0.60) | | (0.60) | (428.45) | (12,273.39) | (0.50) | (413.65) |
| Forks | | | | | | 2.25*** | 0.0001*** | 0.03** |
| | | | | | | (0.28) | (0.0000) | (0.01) |
| Constant | 17,786.41 | 9.50*** | 9.34*** | 9.50*** | -933.05 | 29,644.65** | 9.78*** | 780.98 |
| | (20,241.92) | (0.65) | (0.19) | (0.65) | (1,145.38) | (13,294.46) | (0.54) | (1,332.33) |
| Observations | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 |
| $R^2$ | 0.27 | 0.31 | 0.22 | 0.31 | 0.40 | 0.70 | 0.53 | 0.46 |
| Adjusted $R^2$ | 0.16 | 0.20 | 0.20 | 0.20 | 0.31 | 0.64 | 0.45 | 0.37 |
| Residual Std. Error | 31,389.38 (df = 46) | 1.01 (df = 46) | 1.01 (df = 52) | 1.01 (df = 46) | 751.90 (df = 46) | 20,485.28 (df = 45) | 0.84 (df = 45) | 720.19 (df = 45) |
| F Statistic | 2.48** (df = 7; 46) | 2.94** (df = 7; 46) | 14.56*** (df = 1; 52) | 2.94** (df = 7; 46) | 4.44*** (df = 7; 46) | 12.96*** (df = 8; 45) | 6.36*** (df = 8; 45) | 4.88*** (df = 8; 45) |
| *Note:* | | | | | | | | *p<0.1; **p<0.05; ***p<0.01 |

# IX Appendix

## Data

https://github.com/Leschonander/ThesisResearch

Linked to my GitHub account is the dataset I used to write my paper, along with the code used to run my regressions. This is simply so that an outsider can verify the work I did with the same dataset that I used.

**Projects**

Below is an index of all the projects I that where in my dataset by category. Each will have a link to the repository along with a brief explanation on what it actually is.

Back-End

- Node

  https://github.com/nodejs/node

  A JavaScript runtime built on Chrome's JavaScript V8 Engine. Let's one do server side programming in JavaScript.

- Go

  https://github.com/golang/go

  The Go programming language. Programming language invented by Google to make it easier to build software. Often used for distributed systems applications.

- Ruby on Rails

  https://github.com/rails/rails

  Web framework created by Basecamp to make it easy to build web applications in the programming language Ruby. A famous example of the usage of Model-View-Control when it comes to creating web applications.

- NPM

  https://github.com/npm/npm

A command line tool created by NPM to help manage JavaScript packages. Meant to make it easy to install different packages for JavaScript projects.

- Kotlin

  https://github.com/JetBrains/kotlin

  A programming language on the JVM (Java Virtual Machine) by JetBrains. Meant to reduce boilerplate when doing any Java programming along with making it easier to create Android applications.

Data-Science

- Tensorflow

  https://github.com/tensorflow/tensorflow

An open source framework for Google to make it easier to write machine learning code. The specialty however is in writing deep learning code. (Deeping learning is a subset of the field of artificial intelligence).

- Keras

  https://github.com/keras-team/keras

  Deep learning package in Python created by researcher François Chollet. Known for a module interface that makes it easy to create different types of neural networks.

- PyTorch

  https://github.com/pytorch/pytorch

  Deep learning framework created by Facebook. Specialized in having a lower level interface that makes it easier to create more complicated neural networks vs's ones designed for more generic means.

- Apache Spark

https://github.com/apache/spark

Big data analytics engine. Makes It easy to split up terabyte and larger sized datasets into smaller portions that can then be analyzed in programming languages such as Python and Scala.

- Julia

  https://github.com/JuliaLang/julia

  Programming language designed to be a high performance when it comes to technical computing.

- Pandas

  https://github.com/pandas-dev/pandas

  Package in Python that makes it easy to work with table/relational data within data. Meant to be a high powered data manipulation tool that is easy to use.

- Scikit-Learn

  https://github.com/scikit-learn/scikit-learn

  Python package that implements various out-of-the-box machine learning algirthims ranging from linear regressions to random forests. Also provides utilities such as tools to make it easy to do the traditional train-test-split.

- Apache Kafka

  https://github.com/apache/kafka

  A distributed streaming and real time event platform. Designed to do things suh as messaging or tracking live time activites.

- Apache MXNet

  https://github.com/apache/incubator-mxnet

  A deep learning framework designed for both symbolic and imperative programming that has automatic optimization within it.

- Apache Beam

https://github.com/apache/beam

Model for creating and defining ETL (Extract, Transform, Load) pipelines

- Apache Hadoop

  https://github.com/apache/hadoop

  Open source version of Google MapReduce. Framework to make it easy to do distributed tasks such as counting words across a massive number of documents.

- Numpy

  https://github.com/numpy/numpy

  Package in Python to implemented in C/C++ to do fast numeric programming. Has algorithms implemented in C/C++ that can be called in Python to quickly do complicated equations such as linear algebra or Fourier transformations.

Infrastructure

- Kubernetes

  https://github.com/kubernetes/kubernetes

  Open source system created by Google to help manage containerized applications from scaling, maintenance, to deployment.

- Vscode

  https://github.com/microsoft/vscode

  Visual Code Studio is a code editor created by Microsoft that is specialized in creating web and cloud applications. Provides  a language server to make it the equivalent of a IDE (Integrated Developer Environment) for many different languages.

- FaaS

https://github.com/openfaas/faas

Functions as a Service. An open source implementation of the concept of lambdas, but not tied to a major cloud computing service like AWS, Google Cloud, or Microsoft Azure.

- Red Hat Openshift

  https://github.com/openshift/origin

  A service provided by Red Hat to make it easier to run Kubernetes distributions.

- Ansible

  https://github.com/ansible/ansible

  An IT automation platform to make it easy to configure various networks, along with IT task automation.

- Elasticsearch

  A distributed RESTful search engine. Specialized in making it easy to search text and documents.

- Elastic Kibana

  https://github.com/elastic/kibana

  A browser method to search and analyze data collected from Elasticsearch.

- Kebyase

  https://github.com/keybase/client

  Client for Keybase by Keybase. Keybase is a method via encryption to connect social media identities to encryption keys. It also provides encrypted chat and filesystem services.

- Bitcoin

  https://github.com/bitcoin/bitcoin

Digital Currency invented by Satoshi nakomoto back in 2009. Uses peer-to-peer technology to operate without the need of a centralized athourity.

- Ethereum

  https://github.com/ethereum/go-ethereum

  Cryptocurency invented by Vitalk Buterin back in 2014. Unlike Bitcoin Ethereum possess a concept known as a smart contract which makes it possible to do transactions that involve programmable money.

- Terraform

  https://github.com/hashicorp/terraform

  Infrastructure building and versioning tool created by hashicorp to make it easier to update code.

- Git

  https://github.com/git/git

  Distributed Version Control software originally invented by Linus Torvalds in 2005.

- Linux

  https://github.com/torvalds/linux

  Operating system invented by Linus Torvalds. Used on the majority of the worlds super computers along with on user computers and via android on every single android device in the world.

- Curl

  https://github.com/curl/curl

A command line tool for delivering data that is a URL.

- Chef
- https://github.com/chef/chef

A automation platform created by Chef to make it easier to manage IT processes.

- Varnish
- https://github.com/varnishcache/varnish-cache
- A HTTP accelerator. Allows one to cache HTTP responses to make loading the page faster after someone has been there previously.

Front-End

- React

  https://github.com/facebook/react

  Web framework created by Facebook. Based on the ideas of declarative and functional programming as one creates functions that transform themselves into elements on the page.
- Angular

  https://github.com/angular/angular

  Web framework created by Google that is also based on a component system to design web applications.
- Vue

  Web framework created by an independent developer. Also based on a component framework.

  https://github.com/vuejs/vue

- Automattic WordPress Calypso

  https://github.com/Automattic/wp-calypso

The frontend of Wordpress managed by Automattic, the company that is behind WordPress, one of the most widly used content management systems on the planet.

- TypeScript

  https://github.com/microsoft/TypeScript

Subset of JavaScript created by Microsoft. JavaScript with a type system to make larger JavaScript systems easier to manage.

- Joomla

  https://github.com/joomla/joomla-cms

  An open-source content management system (CMS) system.

- Magento

  https://github.com/magento/magento2

  Web framework by Adobe to make doing e-commerce transactions on the internet easier by providing components to do these transactions on webpages.

- Electron

  https://github.com/electron/electron

  JavaScript framework to make it possible with web technologies (HTML, CSSS, JavaScript), to build

- Mapbox

  https://github.com/mapbox/mapbox-gl-js

  Mapping library and SDK (Standard Developer Kit) specialized in mapping along with software for maps within cars.

- Leaflet

https://github.com/Leaflet/Leaflet

Open source mapping library.

Databases

- CockroachDB

  https://github.com/cockroachdb/cockroach

  Open source database by Cockroach labs focused on scale and consistent transactions along with the reliability to survive a variety of failures from disk to datacenter.

- Ne04j

  https://github.com/neo4j/neo4j

  Graph database with unique query language to make it easy to do queries that require looking at highly nested or interconnected data.

- MongoDB

  https://github.com/mongodb/mongo

  Modern document-based database desgned for scabaliyt. A document database for reference is based on the document model versus the relation model that a traditional SQL database would have.

- RethinkDB

  https://github.com/rethinkdb/rethinkdb

  Distributed database focused for real-time web applications for things such as video games that require real-time data feeds that are reliable.

- Redis

https://github.com/antirez/redis

In memory database, cache, and message broker. However it is primary used as a key-value database.

- Cassandra

  https://github.com/apache/cassandra

Scalable database that is compatible with a varity of hardware and cloud infrastructure to make it easy to scale.

- CouchDB

  https://github.com/apache/couchdb

  Open source document database by the Apache foundation.

- MySQL

  https://github.com/mysql/mysql-server

  One of the first SQL databases to be ever made. MySQL was created by oracle and was only in recent years it became open source.

- MariaDB

  https://github.com/MariaDB/server

  Fork of MySQL back when MySQL was not open source. Drop in as to make it very easy to export database data from MySQL to MariaDB.

- SQLite

  https://github.com/mackyle/sqlite

A small database that is the most used database in the world. Unlike most databases it is embedded and read and writes to ordinary disk files, so one does not need to run a server to run SQLite.

- PostgreSQL

  https://github.com/postgres/postgres

  Open source relational database that has existed for the past thirty years. Originally the POSTGRES project because they pivoted to making it a SQL database. Famous for strong add-ons such as PostGIS for geospatial data.