

A Study on Empirical Evaluation of Software Maintainability in Open Source Projects

Ueyama Kanto

Under the supervision of
Professor Hiroyuki Okamura

Dependable Systems Laboratory,
Informatics and Data Science Program,
Graduate School of Advanced Science and Engineering,
Hiroshima University, Higashi-Hiroshima, Japan

Feb 2023

Abstract

Open Source Software (OSS) is developed under an open license, allowing for the modification and redistribution of its source code. The advantages of OSS can include, but are not limited to, high stability and quality, depending on the size and activity of its community. OSS also promotes transparency and often serves as a component in many systems. However, while OSS is freely available, it may come without guarantees, and users might need to rely on community support or seek commercial assistance for complex issues. When introducing OSS into a system, evaluating maintainability is crucial, as multiple options often exist for a single functionality. This requires considering factors such as the history of bug fixes, community engagement, documentation quality, and code complexity.

This research aims to quantify maintainability in OSS projects based on the history of bug fixes and explore its relationship with various measurable indicators, such as code complexity, community activity, and frequency of updates, to better understand the factors influencing the long-term sustainability of OSS projects.

Acknowledgements

In this research, I would like to express my heartfelt gratitude to Professor Hiroyuki Okamura, my academic mentor, for his unwavering guidance, warm advice, and continuous encouragement throughout my academic journey. I would also like to extend my thanks to Professor Tadashi Dohi, Associate Professor Sayaka Kamei, and Assistant Professor Junjun Zheng from Osaka University for their valuable insights, constant support, helpful guidance, and encouraging words. Finally, I want to acknowledge the hospitality and encouragement extended to me by the past and present members of the Dependable Systems Laboratory at the Graduate School of Advanced Science and Engineering, Hiroshima University.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	3
2 The classification of OSS	5
2.1 Classification regarding bug fixing capability	5
2.1.1 Persistence diagram	6
2.1.2 PWGK	7
2.1.3 Classification procedure	7
2.2 Classification regarding programming languages	9
2.2.1 Classification procedure	9
2.3 Classification regarding description texts of OSS	10
2.3.1 Classification procedure	10
2.3.2 BERT	10
2.4 Experiment	12
2.4.1 Overview of the experiment	12
2.4.2 Experiment	12
2.4.3 Experimental results	14
2.4.4 Characteristics of each class	21
2.4.5 Discussion	24
3 Evaluation of maintainability in OSS	25
3.1 Related research	26
3.2 The maintainability assessment model	27
3.3 Experiment	28

3.3.1 Experiment using ASF OSS projects.	28
3.3.2 Experiment using Mozilla OSS projects.	38
3.3.3 Discussion	42
4 Conclusion	45
4.1 Summary	45
4.2 Future work	48
A Persistent diagram	49
B Programming language	223
C Description texts of OSS	225
Bibliography	231

Chapter 1

Introduction

Open Source Software (OSS) refers to software developed under open licenses that allow for modification and redistribution of source code, such as GPL, BSD, and MIT. It is generally developed by open communities. Notably, OSS developed by large communities like Mozilla.org and the Apache Software Foundation is known for its high quality and stability. As a result, it is often used as foundational software for building systems.

However, when using OSS, it is essential to pay attention to its quality, particularly its reliability and maintainability. Reliability refers to the characteristic of the OSS not causing disruptions due to bugs. Low reliability in the chosen OSS can decrease the overall system's reliability. Maintainability is the characteristic related to the ability to fix bugs and respond to specification changes. While OSS is available for free use, the responsibility for its operational guarantee always falls on the user. Typically, in OSS maintenance, users report bugs they find, and based on these reports, the community fixes them. Therefore, the maintainability of OSS strongly depends on the activity of the development community. In other words, an active OSS community may quickly address bugs, while using OSS from a less active community could lead to unaddressed bugs and ultimately reduce the system's reliability. Hence, evaluating the maintainability of OSS projects is a critical issue.

This study conducts a maintainability assessment of OSS projects. In particular, we aim to evaluate maintainability using several measurable indicators that serve as activity metrics for OSS projects. The structure of this paper is as follows. First, in Chapter 2, we classify OSS based on the geometric properties

of bug discovery and fixing history and unstructured data such as the development language and project overview. In Chapter 3, we analyze the relationship between quantitative maintainability indicators for OSS proposed in literature [1] and measurable activity metrics in OSS projects. Finally, in Chapter 4, we summarize the findings from Chapters 2 and 3 and discuss future challenges.

Chapter 2

The classification of OSS

In this chapter, we classify and characterize open-source software based on three perspectives: bug-fixing capability, programming languages, and description texts of open-source software. We provide detailed explanations of each classification method.

2.1 Classification regarding bug fixing capability

In this section, we describe the classification method based on the perspective of bug-fixing capability. In this paper, we evaluate the bug-fixing capability of open-source software based on the history of bug fixes and discoveries. Specifically, we create a diagram called a "persistence diagram" from the bug fix and discovery history, and classify software by calculating the similarity of persistence diagrams.

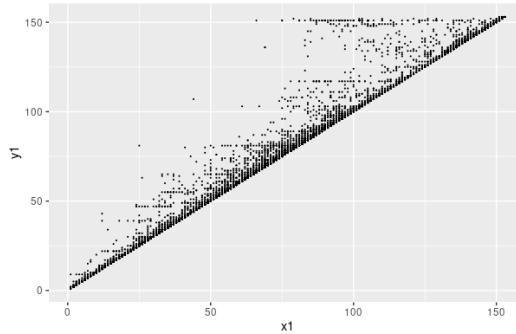
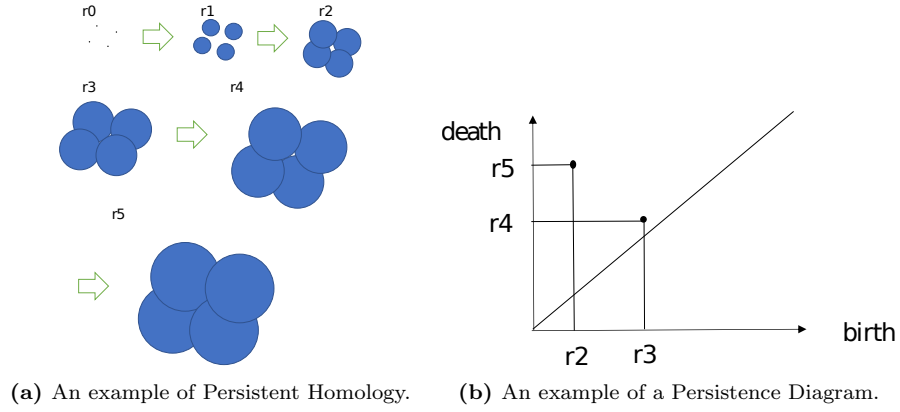


Figure 2.1: Persistence Diagram

2.1.1 Persistence diagram



"Persistent Homology" is a mathematical framework for extracting the topological structure of data. In the context of point set data, circles (or in higher dimensions, spheres) centered around points are considered, where the radius increases over time (Figure 2.2a). When a certain radius is reached, "holes" emerge from the connectivity of these circles. Further increasing the radius causes these "holes" to be filled by overlapping circles, leading to their disappearance. In topology, the number of "holes" serves as a characteristic indicator of a shape. By recording the occurrence and disappearance history of these "holes" over time, valuable information about the point set data can be extracted.

Specifically, a two-dimensional graph is used, with the time of "hole" emergence (the radius of the circle at that time) represented on the horizontal axis and the time of disappearance (the radius of the circle at that time) on the vertical axis. The resulting plot is known as a "Persistence Diagram," depicted in Figure 2.2b, which serves as an indicator expressing the characteristics of the point set. [2]

Similarly, for the bug detection and correction history in software, a Persistence Diagram can be created by plotting the detection time on the horizontal axis and the correction time on the vertical axis.

2.1.2 PWGK

In reference [2], the authors define the Persistence Weighted Gaussian Kernel (**PWGK**) to quantify features in persistent diagrams and discuss the similarity and classification of persistent diagrams. The idea behind PWGK is to emphasize the noisy nature of points corresponding to 'holes' near the diagonal of the persistence diagram, which quickly disappear after being generated. It utilizes weights based on the distance from the diagonal. Specifically, for any positive constants C and p , the weight $w(x)$ is defined as $\arctan(CPers(x)^p)$, where $Pers(x)$ represents the difference between the birth and death times of point x . The PWGK for two points x and y in the persistence diagram is calculated accordingly.

$$k(x, y) = w(x)w(y)\exp(-\frac{\|y - x\|^2}{2\sigma^2}). \quad (2.1)$$

Furthermore, if we denote the point sets in the two persistent diagrams a, b as $P_a = (x_i^a; i = 1, \dots, N_a), P_b = (x_i^b; i = 1, \dots, N_b)$, we can define the following kernel for the persistent diagrams.

$$K(P_a, P_b) = \exp(-\frac{\|\epsilon_k(P_a) - \epsilon_k(P_b)\|^2}{2\tau^2}). \quad (2.2)$$

Here, τ is an arbitrary constant, and

$$\begin{aligned} & \|\epsilon_k(P_a) - \epsilon_k(P_b)\|^2 \\ &= \sum_{i=1}^{N_a} \sum_{j=1}^{N_a} k(x_i^a, x_j^a) \\ & \quad + \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} k(x_i^b, x_j^b) \\ & \quad - 2 \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} k(x_i^a, x_j^b) \end{aligned} \quad (2.3)$$

is defined.

2.1.3 Classification procedure

The classification procedure is as follows.

1. Aggregate the monthly bug fix and discovery counts of open-source software recorded in the Apache Software Foundation (ASF) bug tracking system.

2. Performing task 1 for all types of open-source software managed by ASF allows the creation of a dataset containing the complete history of bug discovery and fixes for all software types.
3. The data created in tasks 1 and 2 consists of count data containing the monthly history of bug fixes and discoveries for each slot. Directly plotting this data will not result in a persistent diagram. To address this, random numbers are assigned as point data corresponding to the number of bugs contained in each slot of this count data. This process allows the creation of persistent diagrams for all open-source software. (Figure [2.1](#))
4. To calculate the similarity of persistent diagrams, a kernel is computed using the formula [\(2.2\)](#) for all combinations of persistent diagrams (for all types of open-source software \times all types of open-source software).
5. From the results of step 4, create a similarity matrix of (all types of open-source software \times all types of open-source software). Subsequently, perform principal component analysis on the similarity matrix.
6. Apply k-means clustering to the matrix obtained from step 5 after performing principal component analysis.

By employing the above method, it is possible to classify open-source software based on the perspective of bug-fixing capability.

2.2 Classification regarding programming languages

Here, we describe a classification method based on the perspective of programming languages.

2.2.1 Classification procedure

The classification procedure is as follows.

1. Aggregate the languages that constitute the open-source software recorded at <https://github.com/apache> and calculate the percentage of each language within each open-source software project. The types and percentages of programming languages are summarized on the GitHub page.
2. Create vectors based on the percentages of languages, treating them as components.
3. Perform tasks 1 and 2 for all types of open-source software, resulting in vectors for all types of open-source software.
4. Based on the vectors created in step 3, generate a matrix representing the percentages of programming languages, with dimensions (number of open-source software categories \times number of programming language types).
5. Apply clustering using the k-means algorithm to the matrix created in step 4.

By employing the above method, it is possible to classify open-source software based on the perspective of programming languages.

2.3 Classification regarding description texts of OSS

Here, we describe a classification method based on the perspective of the description texts of open-source software. Regarding the description texts, we referred to the ASF project list and aggregated the description texts from each homepage [3].

2.3.1 Classification procedure

The classification procedure is as follows.

1. Copy the description texts from the homepages of open-source software and save them as vectors.
2. Perform operations to remove punctuation and stop words from all words.
3. Use the natural language processing model "BERT" (2.3.2) to convert the texts into 768-dimensional vectors.
4. Perform tasks 1, 2, and 3 for all types of open-source software, resulting in 768-dimensional vectors for all types of open-source software.
5. Based on the vectors created in step 4, generate a matrix with dimensions (all types of open-source software \times all types of open-source software).
6. Apply clustering using the k-means algorithm to the matrix created in step 5.

By employing the above method, it is possible to classify open-source software based on the perspective of the description texts of open-source software.

2.3.2 BERT

BERT refers to "Bidirectional Encoder Representations from Transformers," a natural language processing model announced by Google on October 11, 2018. A notable feature of BERT is its high versatility. The model is pre-trained on a large amount of text data from sources like Wikipedia and BooksCorpus, allowing it to be applied to various tasks such as text comprehension and sentiment analysis.

2.3. *CLASSIFICATION REGARDING DESCRIPTION TEXTS OF OSS* 11

The 768-dimensional vectors transformed by BERT represent the features of the text. Similar texts result in close vectors. "Closeness" here refers to the distance between vectors. Despite having a high dimensionality, distances can be calculated similarly, and close distances indicate similar meanings in the context of understanding the semantics of sentences.

2.4 Experiment

In this chapter, we apply the classification methods described in sections [2.1](#), [2.2](#), [2.3](#) to real data and conduct experiments. We present the experimental results and provide discussions on them. The objective of the experiments is to classify and characterize open-source software based on three perspectives: bug-fixing capability, programming language, and description texts. The purpose is to analyze the capabilities of open-source software in fixing bugs and to characterize them based on programming language and description texts.

2.4.1 Overview of the experiment

The overview of the experiment is as follows. Note that the number of clusters for the k-means method used in the experiment is uniformly set to 3.

1. Bug-fixing capability: Create persistent diagrams, calculate kernels on the persistent diagrams using equation [2.2](#). Perform principal component analysis and cluster with k-means method.
2. Programming language: Vectorize the percentage of programming languages and cluster using k-means.
3. Description texts of open-source software: Aggregate and vectorize the description texts from the homepages of each open-source software, then cluster using the k-means method.

2.4.2 Experiment

2.4.2.1 Bug-fixing capability

There were a total of 174 open-source software projects managed by ASF. Persistent diagrams were created for all 174 projects based on the bug-fixing history (refer to Appendix [A](#)). Next, to calculate the similarity of persistent diagrams, kernels were computed using the formula [\(2.2\)](#) for all combinations of the 174 diagrams (174×174). This resulted in a similarity matrix of size 174×174 . Principal component analysis was then performed on this similarity matrix, setting the number of clusters to 3, and clustering was conducted using the k-means method.

2.4.2.2 Programming language

The sentence you provided discusses the creation of a matrix representing the composition of 174 open-source software projects based on 30 different programming languages (refer to Appendix [B](#)). The process involved calculating the percentage of each language’s presence in each software project, resulting in the generation of 174 vectors. Subsequently, a matrix of dimensions 174×30 was constructed to represent the language composition ratios. The next step involved applying the k-means clustering algorithm with a specified number of classes set to 3 to cluster the data.

2.4.2.3 Description texts of open-source software

The description text from the homepages of open-source software projects was copied and saved as vectors. Subsequently, punctuation and stop words were removed from all words in this text, as listed in Appendix [C](#). Following this, the processed descriptive text underwent transformation into 768-dimensional vectors using the natural language processing model "BERT." This series of operations was performed for all types of open-source software, resulting in the creation of a matrix for each open-source software, comprising 174×174 dimensions. Furthermore, the k-means clustering algorithm was applied to these matrices with the number of classes set to 3 for the purpose of clustering.

2.4.3 Experimental results

The experiments conducted in this paper classified 157 out of the 174 open-source software projects managed by ASF into six main groups. The results are presented in Table 2.1

Table 2.1: Contingency table

Group	Number of OSS	Bug-fixing capability	language	Description texts
A	32	3	1	3
B	30	2	1	3
C	28	1	1	3
D	25	2	1	2
E	22	3	1	2
F	20	1	1	2

Next, the clustering results for all types of open-source software are presented in Tables 2.2 through 2.7

Table 2.2: Clustering results

OSS	Bug-fixing capability	language	Description texts
Abdera	3	2	3
Accumulo	2	1	2
Ace	1	1	3
ActiveMQ	1	1	3
Airavata	2	1	3
Airflow	3	1	2
Ambari	2	3	3
Ant	3	1	3
Any23	2	1	2
ASF	1	1	3
Apex	1	1	3
Archiva	3	1	3
Aries	1	1	2
Arrow	3	1	2
Asterix	2	1	3
Aurora	1	1	1
Avro	2	1	2
Axis	1	1	2
Bahir	1	1	2
Batik	3	1	3
Beam	3	1	3
BigTop	3	1	2
BookKeeper	3	1	3
Brooklyn	2	1	2
Buildr	1	1	3
BVal	3	1	1
Calcite	2	1	2
Camel	2	2	3
CarbonData	3	1	3

Table 2.3: Clustering results

OSS	Bug-fixing capability	language	Description texts
Cassandra	2	1	3
Cayenne	1	1	3
Chemistry	2	1	3
Click	1	1	2
Cloudstack	2	3	1
Cocoon	2	1	2
Commons	1	1	3
Continuum	1	1	3
Cordova	3	1	3
CouchDB	2	1	3
Creadur	2	1	3
CXF	3	1	2
Daffodil	1	1	2
Datasketches	1	1	3
DB	1	1	2
DeltaCloud	2	1	3
Deltaspikes	1	1	2
Devicemap	3	1	3
Directory	2	1	3
DistributedLog	1	1	3
Drill	3	1	2
ECharts	1	1	2
ESME	3	1	3
Etch	2	1	2
Felix	2	1	3
Fineract	2	2	3
Flex	2	1	2
Flink	2	1	2
Flume	1	1	3

Table 2.4: Clustering results

OSS	Bug-fixing capability	language	Description texts
Freemarker	2	1	2
Geode	3	1	3
Geronimo	2	3	3
Giraph	3	1	2
Gobblin	3	1	3
Gora	3	1	2
Groovy	3	1	2
Guacamole	2	1	3
Gump	1	1	3
Hadoop	2	1	2
Hawq	3	1	2
Helix	1	1	3
Hivemall	3	1	3
HTTPServer	1	1	3
HTTPComponents	2	1	3
Ignite	3	1	3
Impala	1	1	2
Incubator	2	1	2
Infrastructure	3	1	3
Isis	2	1	3
Jackrabbit	2	1	2
James	1	1	3
jclouds	1	1	2
Jena	3	2	2
Joshua	1	1	3
JSPWiki	1	1	3
jUDDI	3	1	2

Table 2.5: Clustering results

OSS	Bug-fixing capability	language	Description texts
Kafka	3	1	2
Karaf	3	1	3
Knox	2	3	3
Kudu	2	1	3
Kylin	1	1	2
Libcloud	2	1	3
Logging	3	1	2
Lucene	2	1	2
Lucy	2	1	3
MADlib	1	1	3
Mahout	3	1	2
ManifoldCF	1	1	2
Maven	3	1	3
Mesos	2	1	2
MINA	3	1	3
Mnemonic	1	1	1
MRUnit	2	1	3
MyFaces	1	1	2
Mynewt	2	1	3
Myriad	1	1	2
NetBeans	3	1	3
NiFi	2	1	3
Nutch	2	1	3
ODE	3	2	2
OFBiz	3	1	3
Olingo	3	1	2
Onami	3	1	3
OODT	2	1	3
Oozie	3	1	3

Table 2.6: Clustering results

OSS	Bug-fixing capability	language	Description texts
OpenEJB	2	3	3
OpenJPA	3	1	2
Openmeetings	1	1	3
OpenNLP	2	1	2
OpenWebBeansP	1	1	2
Orc	1	1	3
Ozone	2	1	3
PDFBox	3	1	3
Phoenix	2	1	3
Pinot	3	1	2
Pivot	3	1	2
Polygene	2	1	3
Portals	3	1	3
Pulsar	3	1	3
Qpid	1	1	2
Ranger	3	1	3
Ratis	2	1	3
REEF	2	1	2
Retired	3	1	3
River	3	1	3
Roller	1	1	3
Rya	2	1	2
Samza	1	2	3
ServiceMix	1	1	3
Shindig	2	1	2
Shiro	2	1	2
Singa	1	1	2
SIS	1	1	3
Sling	1	3	3
Steve	1	1	3

Table 2.7: Clustering results

OSS	Bug-fixing capability	language	Description texts
Storm	1	1	3
Stratos	2	1	2
StrutsFramework	3	1	3
Superset	1	1	2
Synapse	2	1	3
Syncope	3	1	3
SystemDS	1	1	2
Tapestry	3	1	2
Taverna	2	1	3
Tez	1	1	3
Thrift	2	1	2
Tika	2	1	3
Tomcat	1	1	3
TomEE	2	1	2
TrafficServer	2	1	3
Turbine	1	1	3
Tuscany	1	1	2
UIMA	3	1	2
Unomi	2	2	3
Usergrid	2	1	3
Velocity	3	1	3
WebServices	3	1	2
Whimsy	3	1	2
Wicket	3	1	3
Xalan	1	3	3
XML	2	1	2
XMLBeans	3	1	3
Yetus	2	1	3
Zeppelin	2	1	2
ZooKeeper	2	1	2

2.4.4 Characteristics of each class

Describe the characteristics of each class based on the clustering results.

2.4.4.1 bug-fixing capability

When classifying based on bug-fixing ability, the characteristics of each class are as follows

- Class1: Contains a significant number of persistent charts, like the one shown in Figure [2.3](#), where bugs are consistently fixed over a certain period. Represents a set of software projects with variability in bug-fixing abilities. 73 out of 174 items are classified into Class 1, constituting approximately 42% of the total.
- Class2: Exhibits charts with a prolonged duration between bug discovery and resolution, resulting in a higher accumulation of unresolved bugs, as seen in Figure [2.4](#). Suggests a collection of software projects with relatively lower bug-fixing capabilities. 46 out of 174 items are classified into Class 2, constituting around 26% of the total.
- Class3: Displays charts with a short time between bug discovery and resolution, with many points plotted along the diagonal, resembling Figure [2.5](#). Indicates a set of software projects with relatively higher bug-fixing capabilities. 55 out of 174 items are classified into Class 3, constituting approximately 32% of the total.

2.4.4.2 Programming language

When classified based on the programming language composition, the characteristics of each class are as follows

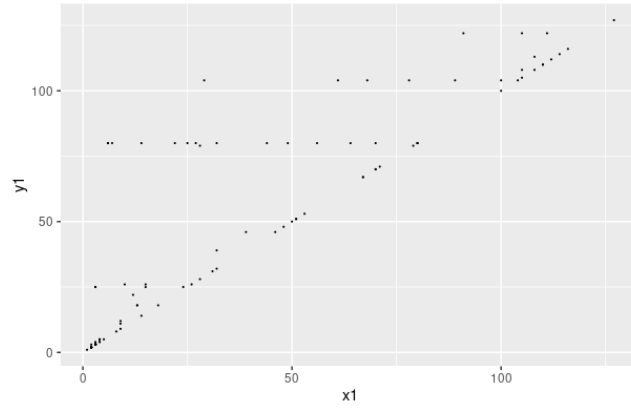
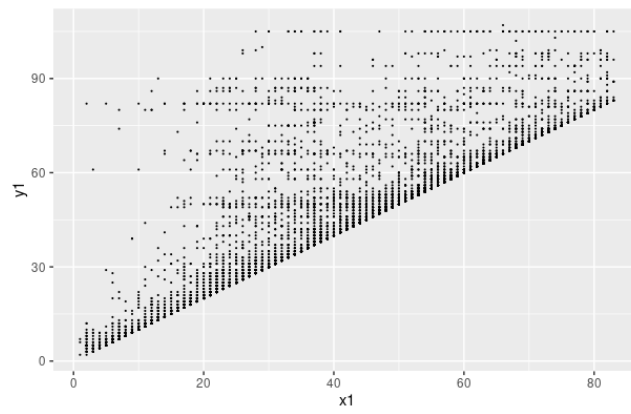
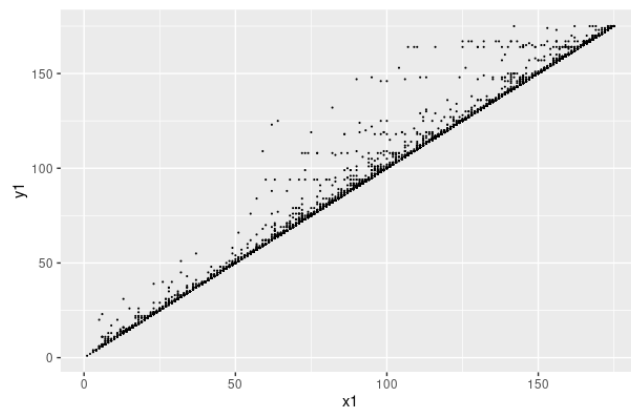
- Class1: Out of 174 items, 160 items are classified into Class 3. The majority of open-source software falls into this class. Various programming languages are used across Class 1. Java dominates in over 70% of Class 1, accounting for more than 80% of the class. When compared to Class 2 and Class 3, the usage of HTML, shell, TypeScript, and other languages is higher in Class 1.

- Class2: 7 out of 174 items are classified into Class 2. The majority of open-source software in this class has a Java dominance exceeding 90%, with JavaScript and HTML used in over half of the projects, albeit in small percentages. Overall, 11 different languages are used in Class 2.
- Class3: 7 out of 174 items are classified into Class 3. In comparison to Class 2, there are fewer open-source software projects with a high Java dominance. The variety of languages used is more extensive, with 13 different languages present. Prominent open-source software projects in Class 3 show high rates of Python and C++ usage.

2.4.4.3 Description texts of open-source software

When classified based on the characteristics of descriptive texts, the features of each class are as follows

- Class 1: 4 out of 174 items are classified into Class 1. Class 1 is characterized by small and concise homepages with a low word count. No distinctive features of natural language were observed.
- Class 2: 69 out of 174 items are classified into Class 2. Class 2 is characterized by a higher word count and a prevalence of descriptive texts for images. The software's name is frequently used within the homepage content (e.g., if the homepage is for 'jclouds,' the term 'jclouds' is commonly used).
-
- Class 3: 101 out of 174 items are classified into Class 3. Class 3 shares the characteristic of frequently using the software's name within the homepage content, similar to Class 2. Additionally, Class 3 is distinguished by the frequent use of the acronym "ASF" for "Apache," and some projects have larger homepages, resulting in a higher word count.

**Figure 2.3:** Class1**Figure 2.4:** Class2**Figure 2.5:** Class3

2.4.5 Discussion

This experiment classified the 174 types of open-source software managed by ASF, where approximately 90% (157 types) fell into the following six groups based on certain characteristics

1. Group A: Short time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS related to distributed processing frameworks like "Apache Hadoop," such as "Carbondata" and "Oozie."
2. Group B: Long time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting internet protocols, for example, "Guacamole" and "HTTPComponents."
3. Group C: Variability in the time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting Java-based applications like "ActiveMQ" and "Buildr."
4. Group D: Long time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting the development of applications, such as "REEF" and "Flink."
5. Group E: Short time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS with relevance to databases, like "juddi" and "Drill."
6. Group F: Variability in the time duration from bug discovery to resolution. High dominance of the Java language. Similar to Group A, inclusion of OSS related to "Apache Hadoop," such as "Impala" and "Bahir."

These classifications highlight distinctive features within each group, providing insights into the bug resolution times and the prevalence of the Java language across different types of open-source software.

Chapter 3

Evaluation of maintainability in OSS

Maintainability in open-source software refers to the characteristics related to bug fixing and modifications, which are also associated with the activity of the development community for that open-source software. Open-source software is available for free use; however, there is no guarantee regarding its operation. In other words, users are responsible for the outcomes of its use, even in the case of significant bugs. Since there is no assurance that the open-source software development community will address and fix critical bugs, evaluating the activity of the open-source software development community, or its maintainability, becomes crucial.

In this chapter, we investigate the relationship between the quantitative maintainability of open-source software obtained using literature [1] and indicators obtained from GitHub.

3.1 Related research

In the context of research related to the maintainability of open-source software, two notable references are cited, namely, [4] and [5].

In [4], the importance of continuous development and support for open-source software usage in enterprises is highlighted. The paper points out that in cases where open-source software projects fail, enterprises face challenges in either seeking alternative open-source software or developing the software in-house, leading to increased effort and costs. To address this concern, the paper adopts an approach involving the extraction and analysis of activity metrics, such as project activity information, from open-source software projects. The goal is to investigate which metrics are strongly correlated with the success or failure of a project.

On the other hand, [5] addresses the concern that the adoption risk of open-source software is perceived as high by enterprises due to uncertainties about the continuity of releases and the lack of maintenance activities, such as bug fixes. The paper notes that this perception often deters enterprises from embracing open-source software. To tackle this issue, the authors conduct a systematic review on the survivability of open-source software. The review explores the characteristics used in assessing the survivability of open-source software and leverages the insights gained from the systematic review to apply survival time analysis to open-source software.

In summary, both [4] and [5] contribute valuable insights into the challenges associated with the maintainability and adoption risks of open-source software in enterprise contexts. They employ different methodologies, including metrics analysis and systematic reviews, to investigate and address these challenges.

3.2 The maintainability assessment model

In references [4] and [5], the discussion revolves around the relationship between the activity levels of open-source software and metrics such as the number of commits. However, these studies do not explicitly address whether these metrics, particularly the commit-related indicators, impact the crucial aspect of maintainability, namely, "bug fixes." Therefore, in this section, we consider quantifying the bug-fixing capability of open-source software using literature [1].

In this study, we utilize the maintainability assessment model proposed in literature [1]. The model is an extension of the conventional software reliability growth model [1] and is constructed based on the following assumptions: (i) the number of latent bugs in the software follows a Poisson distribution, (ii) there is a correlation between the discovery time and the repair time for a single bug, which can be represented by some bivariate distribution, and (iii) each bug is independent, and the bivariate distribution for discovery and repair is identical for all bugs. In literature [1], an efficient method for estimating model parameters from actual data (bug-fixing history) related to bug discovery and repair is proposed by applying a bivariate Erlang distribution to the discovery time and repair time. Using the estimated parameters, it is possible to calculate the "probability that a discovered bug remains unfixed for a certain period" as a measure of maintainability for the given open-source software.

Specifically, when the bivariate probability density function $f(t, s)$ for the discovery time and repair time of a single bug is given, the probability that a bug remains unfixed after S hours since discovery (WIPFD: Work in Progress for Fixing Debugs) is represented by Equation 3.1. A smaller value of this indicator implies that the project fixes bugs promptly, indicating higher maintainability for the OSS in question.

$$\text{WIPFD}(S) = \int_0^\infty \int_S^\infty f(t, s) ds dt \quad (3.1)$$

3.3 Experiment

3.3.1 Experiment using ASF OSS projects.

In the experiment, the bug-fixing history for 159 OSS projects recorded in the ASF (Apache Software Foundation) JIRA¹ was utilized to assess the maintainability of OSS using WIPFD. The study also investigates the correlation with metrics readily obtainable from GitHub (refer to Table 3.1)

Table 3.1: Metrics collected from GitHub repository

Metric	Description
Commit	Cumulative number of commits to date
Contributors	Total number of contributors to date
LOC	Current total lines of code
Pull Request	Number of currently open pull requests
Closed Pull Request	Number of currently closed pull requests
Issue	Number of currently open issues
Closed Issue	Number of currently closed issues
Watch	Current repository watch count
Fork	Current repository fork count
Star	Current repository star count

The explanations for each metric are provided below.

- Commit

GitHub Commits are essential for tracking the history of changes in a project, providing a detailed record of the project's progress. Each Commit is accompanied by a commit message, which describes the changes made and the purpose of those changes. This commit message serves as a clear and concise summary of the alterations made to the codebase. Commits are typically done on branches, allowing contributors to work on specific features or bug fixes independently. The changes proposed in a commit often go through a code review process before being merged into the main branch. In summary, GitHub Commits are a fundamental aspect of collaborative development, providing transparency and accountability

¹<https://issues.apache.org/jira/>

for code changes in a project.

- Contributors

GitHub's "Contributors" feature provides a list of individuals or contributors who have actively contributed to a project. Contributors encompass those who have made additions or changes to the code, fixed bugs, improved documentation, and any other form of contribution to the project. The Contributors page serves as a way to acknowledge and express gratitude to those who have contributed to the project, fostering a sense of cooperation and empathy within the open-source community. It plays a crucial role in highlighting contributions to the project and recognizing individuals for their collaborative efforts, contributing to the transparency and culture of appreciation in open-source development.

- LOC

"LOC" stands for "Lines of Code," and it is a metric that measures the total number of lines in a software project's source code. This includes lines of actual code, comments, and blank lines. The LOC metric provides insights into the size and complexity of a codebase. It is used to evaluate the scale of a project and can be an indicator of the effort and resources invested in its development. Additionally, LOC can be useful for assessing maintainability, as larger codebases may require more effort to understand, debug, and update. However, it's essential to note that LOC alone doesn't determine code quality or efficiency, and other metrics and factors should be considered for a comprehensive evaluation.

- Pull Request

A "Pull Request" is a proposal submitted on platforms like GitHub to integrate changes made in a forked repository into the original project. It serves as a formal way for contributors to suggest modifications, have them reviewed by other contributors, and eventually merged into the original project. When someone makes changes to a project and wants those

changes to be included in the original project, they create a branch containing the modifications and submit a pull request to have those changes pulled into the original project. This process allows for a formal review of the proposed changes before integration. Project owners or maintainers review the pull request, examine the changes, and decide whether to merge them into the main project. Pull requests provide a structured mechanism for collaboration and contribute to maintaining code quality in open-source projects. Metrics related to pull requests, such as the number of open or closed pull requests, can be valuable indicators of a project's activity level, the involvement of the community, and the overall quality of code contributions.

- Issue

GitHub "Issue" are used to track tasks, enhancements, bugs, and other kinds of questions or discussions within a repository. They provide a centralized place for collaboration and communication among project contributors and users. In the context of GitHub repositories: An "Issue" can represent a task that needs to be completed, a bug that needs to be fixed, or a discussion point for the community. Users can create issues to report problems, suggest enhancements, or discuss topics related to the project. Issues can be labeled, assigned to specific contributors, and categorized to help organize and prioritize work. Conversations within issues can include comments, attachments, and other relevant information. Tracking Issue is a common practice in open-source projects to manage and prioritize tasks, foster collaboration, and ensure transparency within the development process. Metrics related to issues, such as the number of open or closed issues, can provide insights into the project's activity, the responsiveness of the development team, and the overall health of the project.

- Watch

The "Watch" feature on GitHub allows users to track updates and activities in a repository. Here are key points about the "Watch" feature:

Watching a Repository: Clicking the "Watch" button on the repository's main page allows users to start watching that repository. **Notifications:** By watching a repository, users receive notifications for various activities related to that repository. This includes new commits, pull requests, the creation of issues, discussions, and more. **Differences from Stars:** While "Starring" a repository is a way to bookmark it and show interest, "Watching" provides activity notifications for changes in the repository. **Ignoring:** Users have the option to temporarily ignore notifications for repositories they are watching. Using the "Watch" feature ensures that users stay informed about activities and changes in repositories they are interested in, without missing important updates.

- Fork

"Fork" is a feature on GitHub that allows users to create a personal copy of another user's repository. **Creating a Fork:** Users can fork a repository by clicking the "Fork" button on the repository's page. This creates a separate copy of the entire repository under their GitHub account. **Purpose:** Forking is commonly used when a user wants to contribute to someone else's project. The forked repository acts as an independent copy that the user can modify without affecting the original project. **Pull Requests:** After making changes in their forked repository, users can submit a pull request to the original repository. This signals to the original project that there are changes the user would like them to review and potentially merge into their project. **Independence:** A forked repository is independent and can be further modified without affecting the original project. It allows users to experiment with changes and contribute back to the original project if desired. **Kept in Sync:** While forks start as independent copies, users can choose to keep their forked repository in sync with the original repository to incorporate any updates made by the original project. In summary, forking is a mechanism on GitHub that facilitates collaborative development and contribution to open-source projects by creating independent copies of repositories.

- Star

"Star" on GitHub represents an action where users express their interest or support for a particular repository. Overview of Star: Users can click the "Star" button on a GitHub repository to show their appreciation or interest in that repository. Purpose: Stars indicate that a user is paying attention to a specific project or expressing support for it. When a repository accumulates many stars, it suggests that the project is popular and recognized for its value. Project Evaluation: The number of stars serves as a measure of the project's quality and usefulness. A higher number of stars implies that the project is noteworthy and considered reliable. Notifications: When a user stars a repository, they can receive notifications about updates or changes to that repository. Project Management: Project owners and maintainers can gauge the popularity and impact of their project by looking at the number of stars. This information can guide development and marketing strategies. In essence, stars on a GitHub repository indicate user interest, support, and popularity, serving as a measure of a project's value and influence.

From Figure 3.1, it is observed that there is a relatively high number of projects fixing bugs within one week, with approximately 80% of projects having WIPFD values below 0.1 (indicating a probability of completing bug fixes within one week is 0.9 or higher). Based on this result, it can be inferred that projects with WIPFD below 0.1 are indicative of good maintainability when using a one-week timeframe.

Similarly, Figure 3.2 illustrates that about 80% of projects have WIPFD values below 0.01 when considering a one-month timeframe. This indicates a probability of completing bug fixes within one month is 0.99 or higher. Consequently, projects with WIPFD below 0.01 can be considered as having good maintainability when using a one-month timeframe.

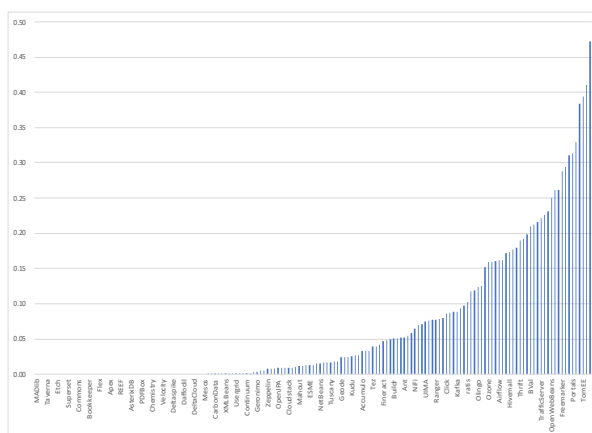


Figure 3.1: WIPFD over a one-week period.

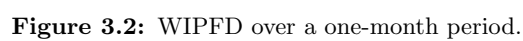


Figure 3.2: WIPFD over a one-month period.

Next, a classification tree was generated with eight repository metrics as input variables and binary output variables (0 and 1) representing two classes: Class 1 for good maintainability and Class 0 for poor maintainability. First, Figure 3.3 represents the classification tree based on a one-week timeframe. Regarding the interpretation of the classification tree, if the branching condition is true, we follow the left branch; if false, we follow the right branch. The circular nodes indicate the percentage of instances out of 159 nodes and the proportion of those nodes belonging to Class 1.

From Figure 3.3, it is evident that the conditions strongly influencing the three nodes with a high proportion of Class 1 (good maintainability) are LOC less than 3,100,000, allissue count greater than or equal to 148, Fork count less than 27, and star count greater than or equal to 41.

Similarly, Figure 3.4 indicates that star count greater than or equal to 83 and Fork count less than 27 are indicators of good maintainability.

When considering a one-week timeframe, having a low Lines of Code (LOC) count tends to be an indicator of good maintainability in a project. The idea is that a smaller codebase reduces the potential for bugs, making maintenance easier. With fewer lines of code, there are fewer potential sources of issues, contributing to better maintainability over time.

Similarly, having a high number of issues is considered an indicator of good maintainability in an OSS project. A high number of issues suggests active interest in the project, indicating the presence of a vibrant developer community. Issues serve as a communication channel between developers and users, enhancing project transparency. Addressing problems and improvements through effective communication between developers and users is believed to contribute to improved maintainability.

Next, the notion that a low Fork count is an indicator of good maintainability is based on the idea that, with fewer Forks, contributors can focus on a single codebase where project progress and fixes are concentrated. This concentration is seen as contributing to improved maintainability.

Finally, a high number of Stars is considered an indicator of good maintainability because it suggests high utility and demand for the project. The attractiveness of the project draws in more contributors, potentially enhancing

maintainability.

When extending the analysis to a one-month timeframe, the factors contributing to the good maintainability associated with low Fork counts and high Star counts remain consistent. Over time, projects with a consistently low number of Forks may indicate a cohesive community that maintains stability and reliability. Projects with an increasing number of Stars over time suggest gaining popularity and trust, attracting more users and contributors, thus contributing to improved maintainability.

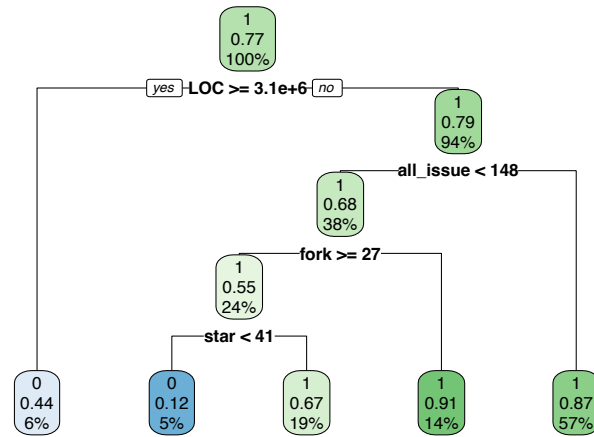


Figure 3.3: The classification tree for WIPFD based on one week

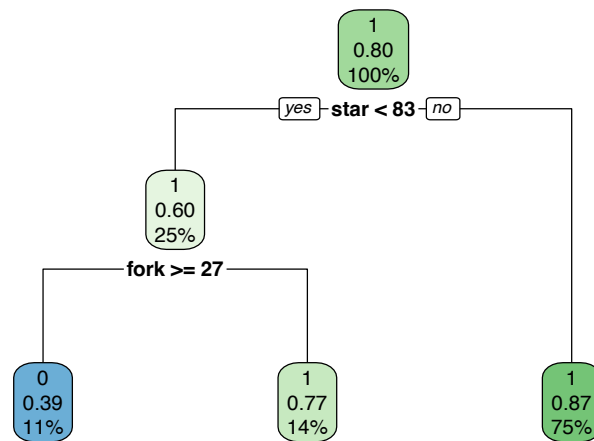


Figure 3.4: The classification tree for WIPFD based on one month

3.3.2 Experiment using Mozilla OSS projects.

Furthermore, experiments similar to those conducted in [3.3.1](#) were carried out using open-source software (OSS) projects other than ASF. In this study, Mozilla's OSS projects were used as the data source. Mozilla's OSS (Open Source Software) projects are initiatives promoted by the Mozilla Foundation, focusing primarily on the development of web technologies and products. The following are some of the major Mozilla OSS projects:

1. Mozilla Firefox

A prominent open-source web browser known for its emphasis on security, privacy, and fast browsing experience. Customizable with extensions and themes.

2. Mozilla Thunderbird

An open-source email client facilitating functions such as sending and receiving emails, organization, search, and filtering. Maintained by the Mozilla Foundation.

3. Mozilla Servo

Next-generation browser engine developed in the Rust programming language. Aims for improved parallel processing and security.

In the experiment, we utilized data from 110 open-source software (OSS) projects registered on Mozilla's Github repositories. These projects were selected based on having a high number of stars and the availability of bug data. It is noteworthy that the metrics collected from these 110 projects are of the same types as those used in the experiment described in Section [3.3.1](#) (refer to [3.1](#)).

Figures [3.5](#) and [3.6](#) present the calculated WIPFD for 109 projects at Mozilla, based on one-week and one-month timeframes, respectively. The projects are listed in ascending order of WIPFD, and the X-axis includes only a subset of project names.

From Figure [3.5](#), it is observed that a significant number of projects fix bugs within one week, with approximately 80% of projects having a WTPFD below $1E-30$. This suggests that, based on a one-week timeframe, projects with

a WTPFD below $1E-30$ are indicative of good maintainability. On the other hand, Figure 3.6 shows that many projects have a WIPFD of 0, and nearly 80% of projects complete bug fixes within one month. From this result, it can be inferred that, based on a one-month timeframe, projects with a WIPFD of 0, indicating completion of bug fixes, are indicative of good maintainability.

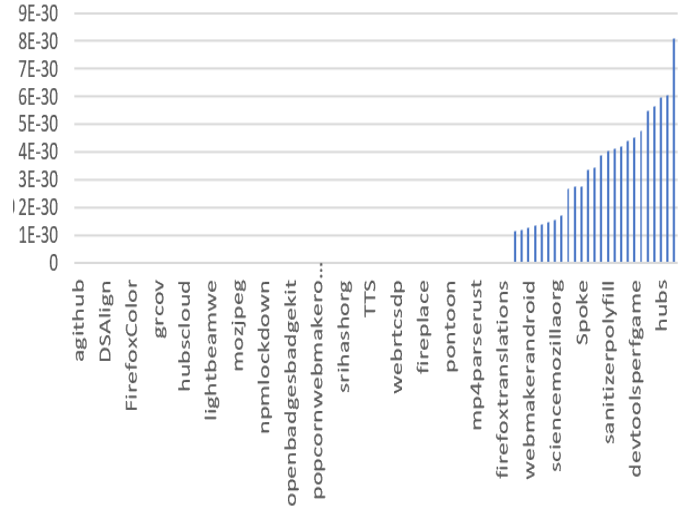


Figure 3.5: WIPFD over a one-week period.

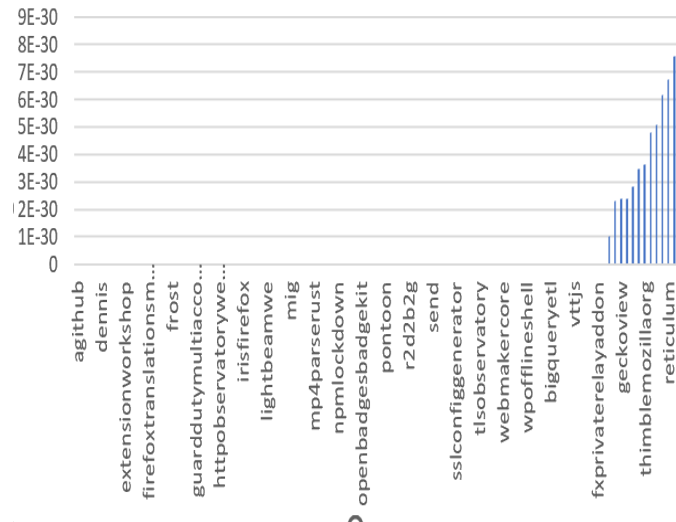


Figure 3.6: WIPFD over a one-month period.

Regarding the generated classification trees, Figure 3.7 reveals that nodes with a higher proportion of Class 1, indicating good maintainability, are strongly influenced by conditions such as having Watch count of 11 or more and Commits count below 4587. Similarly, Figure 3.8 indicates that projects with a Commits count below 1187 and LOC (Lines of Code) equal to or above 49000 are indicative of good maintainability.

In the context of a one-week timeframe, the number of "Watch" events emerges as a significant indicator of project maintainability. The count of "Watch" events reflects the level of interest in the project. A high number of "Watch" events suggests that users and developers are closely monitoring the project, receiving notifications about changes and progress. This continuous follow-up on the project implies an increased level of maintainability, as stakeholders are actively engaged and informed.

Additionally, a lower count of "Commit" events has been identified as a positive indicator for the maintainability of an open-source software (OSS) project. A lower number of commits suggests that changes are transparent. When project changes are well-planned, and communication is thorough, unnecessary modifications tend to be minimized. High transparency allows developers and community members to easily comprehend changes, leading to improved maintainability.

When extending the analysis to a one-month timeframe, the findings indicate that not only a lower count of commits but also a moderate number of lines of code (LOC) is associated with good project maintainability. This suggests that it's not merely about having fewer commits but also about considering the ratio of commits to lines of code. The balance between a low commit count and a substantial LOC implies a level of efficiency and effectiveness in the development process that positively influences maintainability.

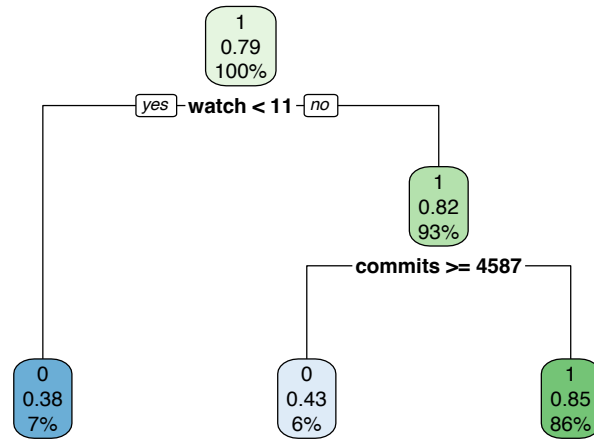


Figure 3.7: The regression tree for WIPFD based on one week

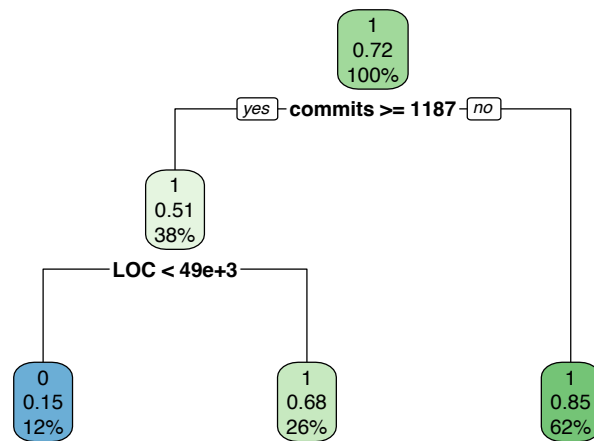


Figure 3.8: The regression tree for WIPFD based on one month

3.3.3 Discussion

From the above experiments, it is evident that for ASF (Apache Software Foundation) OSS projects, particularly having a low number of Forks and a high number of Stars have a positive impact on the maintainability of OSS. In contrast, for Mozilla OSS projects, having a low number of Commits is identified as having a positive impact on maintainability.

Here, ASF and Mozilla Foundation are organizations with distinct characteristics and purposes. The following are some of the key differences:

Apache Software Foundation (ASF): Purpose and Role: ASF is a non-profit organization with the aim of providing open-source software projects and supporting their communities. Through offering software licenses and legal support, ASF assists projects in growing and maintaining healthy communities. **Organizational Structure:** ASF hosts multiple software projects as independent entities, each with its unique development model and community. Each project operates independently, and ASF provides legal frameworks and infrastructure support. **License Policy:** ASF advocates for several open-source licenses, including the Apache License. Projects under ASF typically feature licenses that are flexible regarding code reuse and modifications.

Mozilla Foundation: Purpose and Role: Mozilla Foundation is dedicated to advancing the evolution and adoption of open web technologies. While known primarily as the developer of the Firefox web browser, Mozilla contributes to enhancing web openness and privacy through various projects and initiatives. **Organizational Structure:** Mozilla Foundation closely relates to Mozilla Corporation, a for-profit subsidiary. Mozilla Corporation manages commercial product development and revenue, while the Foundation supports activities pursuing the public interest. **Projects:** Mozilla Foundation supports projects beyond the development of the Firefox browser, including Thunderbird (email client) and the Mozilla Developer Network (MDN).

Considering the factors mentioned earlier, ASF (Apache Software Foundation) OSS projects are generally operated independently, with each project being autonomously managed. This autonomy enables the community to collaborate on maintenance activities even in projects with fewer forks. Additionally, since ASF projects focus on foundational software such as middleware and infras-

tructure, they are likely to garner Stars from diverse user communities, making them popular and easily supported.

On the other hand, Mozilla’s OSS projects predominantly focus on large and complex software projects like Firefox and related products. These projects prioritize maintaining stability and compatibility with existing features. Given the need to carefully consider the impact of changes on other parts of the system, modifications are made cautiously, potentially resulting in a lower number of commits. This approach helps avoid unnecessary changes and redundant code, contributing to easier maintenance and enhancing overall maintainability.

In conclusion, summarizing the impact of each metric on the maintainability of open-source software (OSS) projects based on the experiments conducted in this study, it was observed that four key metrics—Lines of Code (LOC), Forks, Stars, and Commits—strongly influence the maintainability of OSS projects. Additionally, Watch and Issues were identified as metrics that also contribute to the impact on maintainability.

1. Strongly Influencing Metrics:

Lines of Code (LOC): The size and complexity of the codebase, as indicated by LOC, significantly affect maintainability.

Forks: The number of forks, representing project duplications, is indicative of collaboration and community involvement, positively influencing maintainability. Stars: The number of stars, reflecting user interest and endorsement, is correlated with higher project maintainability.

Commits: The frequency of commits indicates project activity and is closely tied to maintainability.

2. Contributing Metrics:

Watch: The number of Watch events, showcasing project interest, was identified as contributing positively to maintainability.

Issues: The presence and management of issues were found to be associated with improved project maintainability.

3. Non-Influencing Metrics:

Contributors: The number of contributors was determined not to be a

significant factor influencing project maintainability.

Pull Requests: The metric of pull requests was also found not to have a notable impact on project maintainability.

Chapter 4

Conclusion

4.1 Summary

In Chapter 2, we conducted an experiment to classify and characterize open-source software based on three perspectives: bug-fixing ability, programming language, and the descriptive texts of the software. From this experiment, it was observed that among the 174 types of open-source software, approximately 90% (157 types) could be classified into the following six groups:

1. Group A: Short time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS related to distributed processing frameworks like "Apache Hadoop," such as "Carbondata" and "Oozie."
2. Group B: Long time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting internet protocols, for example, "Guacamole" and "HTTPComponents."
3. Group C: Variability in the time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting Java-based applications like "ActiveMQ" and "Buildr."
4. Group D: Long time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS supporting the development of applications, such as "REEF" and "Flink."
5. Group E: Short time duration from bug discovery to resolution. High dominance of the Java language. Inclusion of OSS with relevance to databases,

like "juDDI" and "Drill."

6. Group F: Variability in the time duration from bug discovery to resolution. High dominance of the Java language. Similar to Group A, inclusion of OSS related to "Apache Hadoop," such as "Impala" and "Bahir."

The result obtained from the experiment is the classification of these into six groups.

In Chapter 3, the focus was on quantifying bug-fixing capabilities using the maintainability evaluation model introduced in literature [1]. This involved investigating the relationship between "measurable indicators in OSS projects" and the quantified "maintainability" using the model from literature [1]. The experiment targeted ASF JIRA and Mozilla OSS projects, analyzing the correlation between WIPFD, calculated from OSS bug-fixing histories, and eight repository metrics collected from GitHub, using a classification tree for analysis.

The results revealed that, for ASF's OSS projects, a low number of Forks and a high number of Stars notably positively influenced OSS maintainability. This observation aligns with the characteristic of ASF projects being generally operated independently, with each project autonomously managed. The autonomy contributes to a low Fork count, suggesting cohesive collaboration around a unified codebase. Additionally, ASF projects, focusing on foundational software like middleware and infrastructure, receive support from a diverse user community. The low number of Forks implies collaborative work within a single codebase, and the extensive support is reflected in a high number of Stars, potentially contributing positively to maintainability.

On the other hand, for Mozilla's OSS projects, especially those with a low Commit count, it was found that this positively affected OSS maintainability. Despite the projects being large and complex, a low Commit count indicates effective management of changes with minimal unnecessary complexity. In such projects, where there are no excessive changes or redundant code additions, maintenance becomes easier, contributing to enhanced maintainability.

To summarize, the collaborative and focused nature of ASF projects, along with diverse user support, and the effective change management in Mozilla

projects with a low Commit count, contribute to their respective maintainability characteristics. Additionally, as a conclusion based on the experiments conducted in this study, the impact of each metric on the maintainability of open-source software (OSS) projects was summarized. The results indicated that four key metrics—Lines of Code (LOC), Forks, Stars, and Commits—strongly influence the maintainability of OSS projects. Additionally, Watch and Issues were identified as metrics that also contribute to the impact on maintainability. Conversely, Contributors and Pull Requests were found not to significantly impact maintainability.

1. Strongly Influencing Metrics:

Lines of Code (LOC): The size and complexity of the codebase, as indicated by LOC, significantly affect maintainability.

Forks: The number of forks, representing project duplications, is indicative of collaboration and community involvement, positively influencing maintainability. Stars: The number of stars, reflecting user interest and endorsement, is correlated with higher project maintainability.

Commits: The frequency of commits indicates project activity and is closely tied to maintainability.

2. Contributing Metrics:

Watch: The number of Watch events, showcasing project interest, was identified as contributing positively to maintainability.

Issues: The presence and management of issues were found to be associated with improved project maintainability.

3. Non-Influencing Metrics:

Contributors: The number of contributors was determined not to be a significant factor influencing project maintainability.

Pull Requests: The metric of pull requests was also found not to have a notable impact on project maintainability.

4.2 Future work

As future challenges, it is essential to investigate whether the observations made in this study regarding the strong impact of four metrics - Lines of Code (LOC), Forks, Stars, and Commits - on the maintainability of OSS projects hold true for projects outside of ASF and Mozilla. Additionally, exploring whether the metrics of Contributors and Pull Requests exhibit a significant influence on project maintainability across diverse OSS projects is crucial.

This investigation will contribute to a broader understanding of the generalizability of the identified metrics in influencing maintainability and will provide insights into the unique dynamics of various open-source projects. It would be valuable to analyze a more extensive range of OSS projects to establish a comprehensive and representative view of the factors influencing maintainability in the open-source software ecosystem.

Appendix A

Persistent diagram

In this paper, when conducting experiments for the classification of open-source software, we utilized persistent diagrams, programming languages, and natural language. Here, we present all the data used in the experiments.

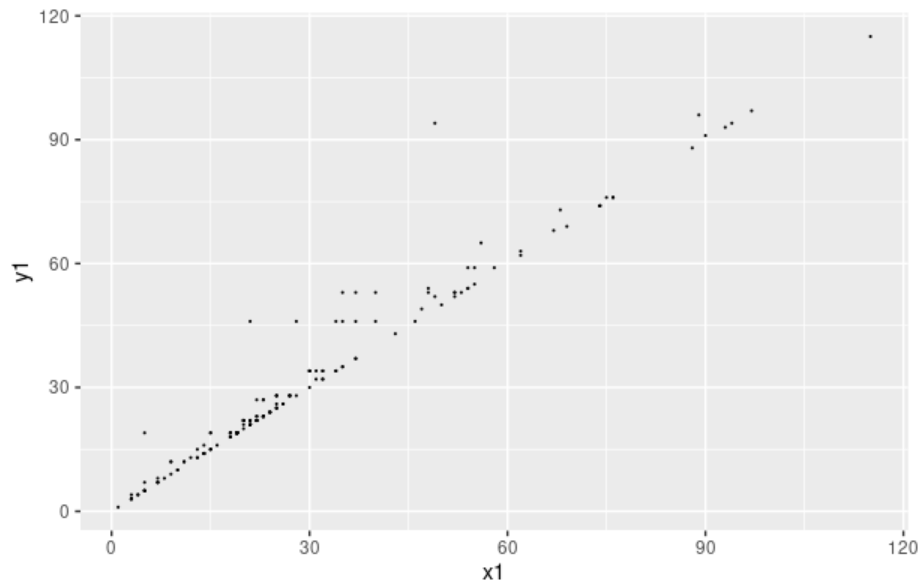


Figure A.1: Abdera

- Observation period: Approximately 120 months
- Total number of bugs: 215
- Percentage of languages (%): Java 99.9, XSLT 0.1
- URL for the software overview: <https://github.com/apache/abdera>

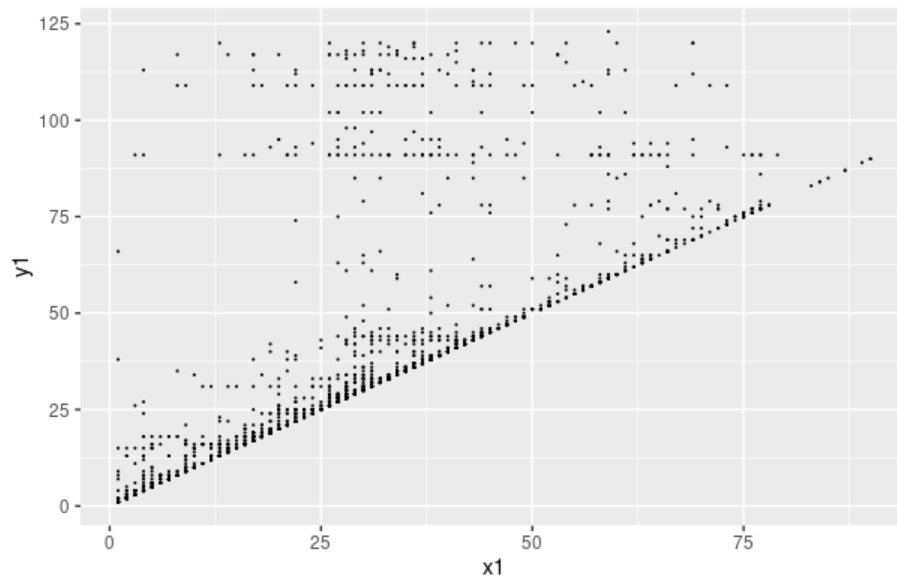


Figure A.2: Accumulo

- Observation period: Approximately 125 months
- Total number of bugs: 2250
- Percentage of languages (%): Java98.6, JavaScript0.4, shell0.3, freemarker0.3, Thrift0.2, C++0.2
- URL for the software overview: <https://accumulo.apache.org/>

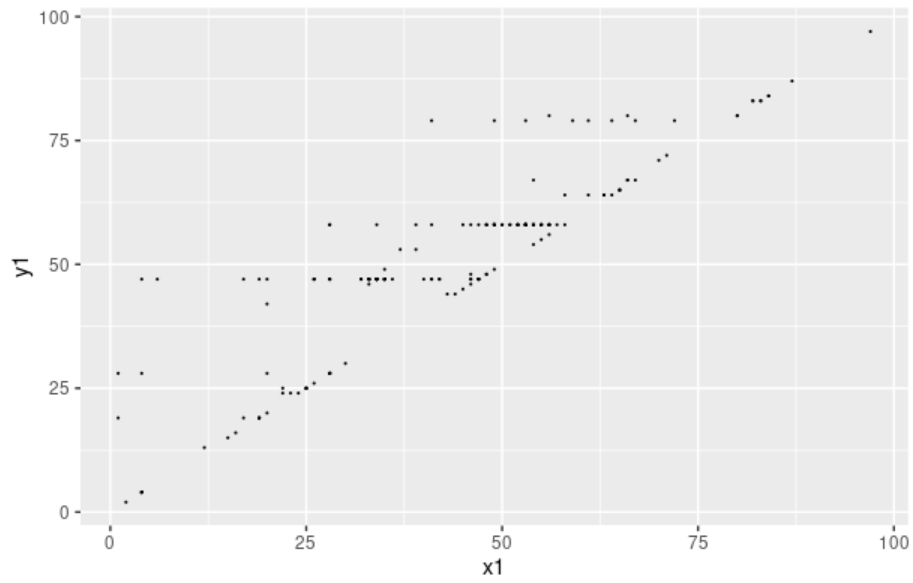


Figure A.3: Ace

- Observation period: Approximately 100 months
- Total number of bugs: 194
- Percentage of languages (%): Java95.1, CSS4.1, shell0.8
- URL for the software overview: <https://ace.apache.org/>

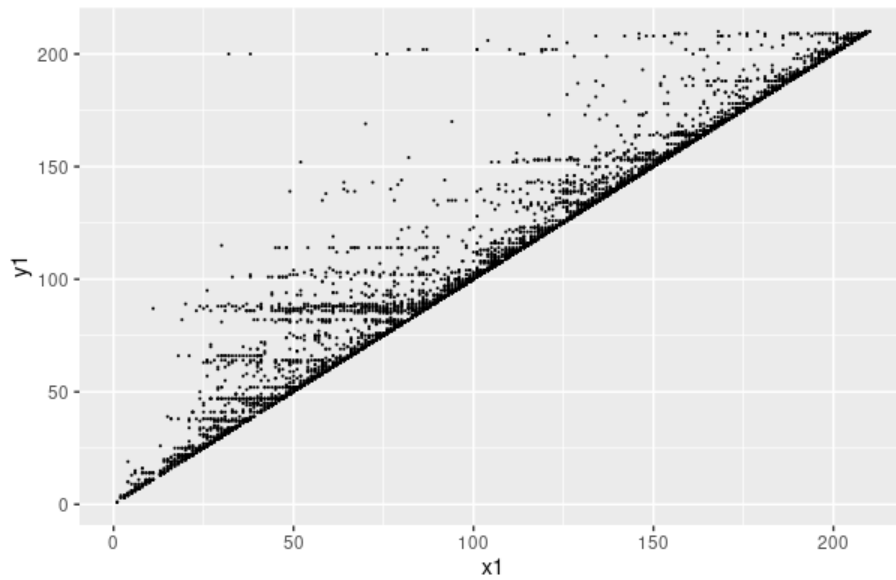


Figure A.4: ActiveMQ

- Observation period: Approximately 200 months
- Total number of bugs: 8190
- Percentage of languages (%): Java96.9,JavaScript1.7,HTML0.6,shell0.3,CSS0.1,C0.1,other0.3
- URL for the software overview: <https://activemq.apache.org/>

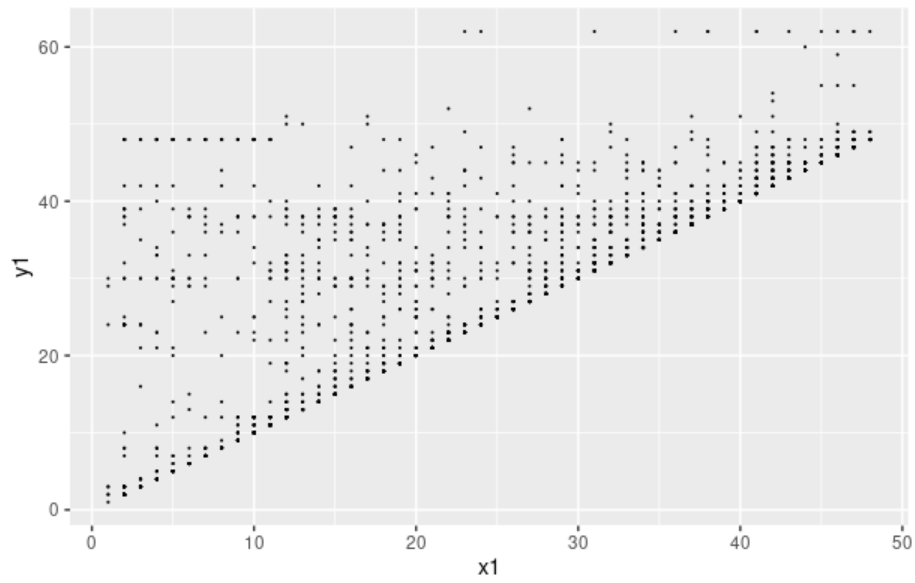


Figure A.5: Airflow

- Observation period: Approximately 50 months
- Total number of bugs: 3139
- Percentage of languages (%): python93.3, shell3.7, TypeScript0.9, JavaScript0.8, HTML0.8, dockerfile0.2, other0.3
- URL for the software overview: <https://airflow.apache.org/>

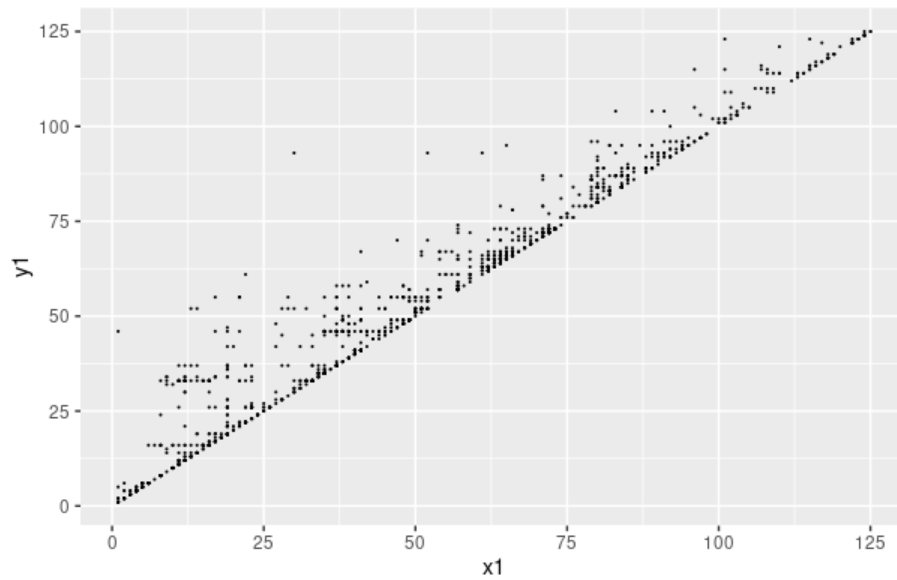


Figure A.6: Airvata

- Observation period: Approximately 125 months
- Total number of bugs: 1636
- Percentage of languages (%): Java74.6, C++17.3, php2.5, python1.3, Thrift1.0, other 3.3
- URL for the software overview: <https://airavata.apache.org/>

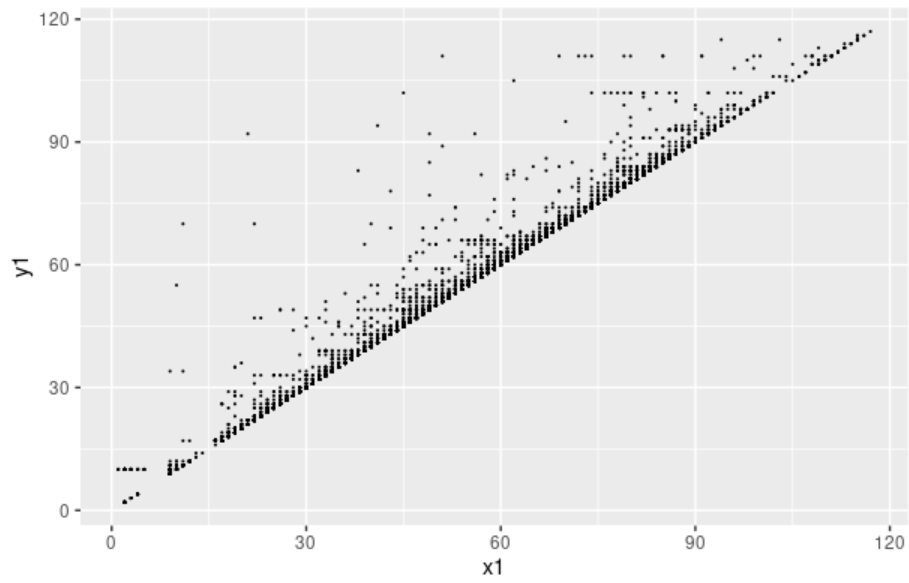


Figure A.7: Ambari

- Observation period: Approximately 120 months
- Total number of bugs: 17959
- Percentage of languages (%): Java45.9, JavaScript25.2, python12.4,HTML6.4, other11.1
- URL for the software overview: <https://github.com/apache/ambari>

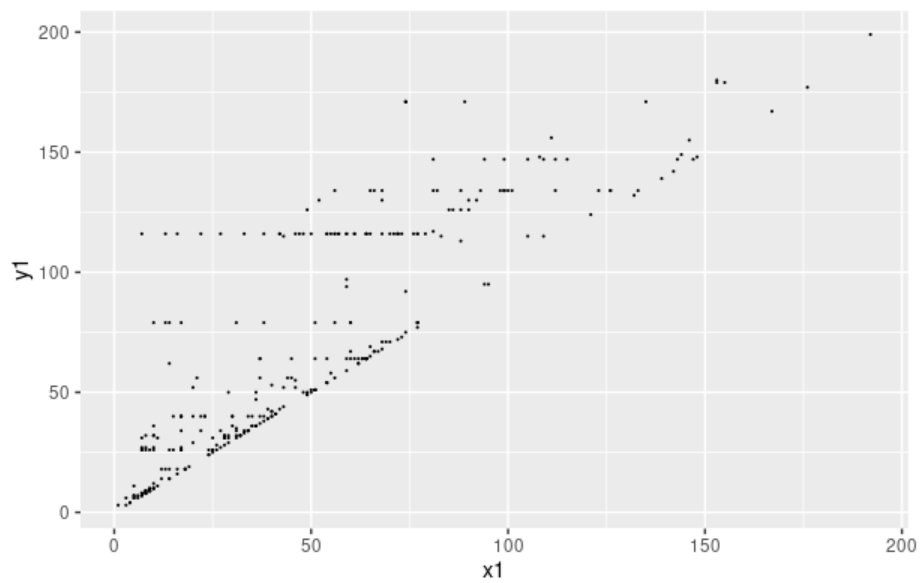


Figure A.8: Ant

- Observation period: Approximately 200 months
- Total number of bugs: 331
- Percentage of languages (%): Java77.7, HTML 17.8, XSLT3.6, shell0.3,batchfile0.2,other0.4
- URL for the software overview: <https://ant.apache.org/>

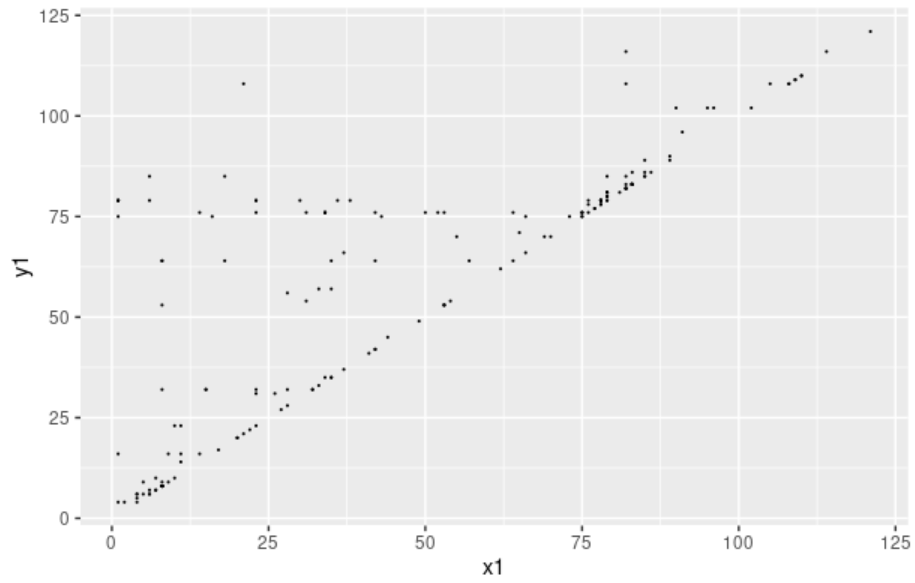


Figure A.9: Any23

- Observation period: Approximately 125 months
- Total number of bugs: 186
- Percentage of languages (%): HTML59.2, Java40.0, other0.8
- URL for the software overview: <https://any23.apache.org/>

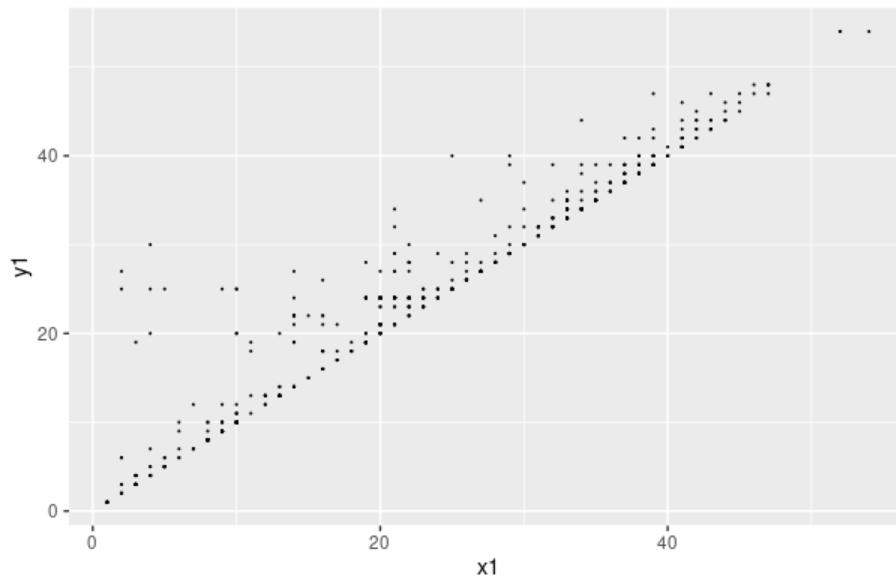


Figure A.10: Apex

- Observation period: Approximately 50 months
- Total number of bugs: 802
- Percentage of languages (%): Java99.7, other 0.3
- URL for the software overview: <https://github.com/apache/apex-core>

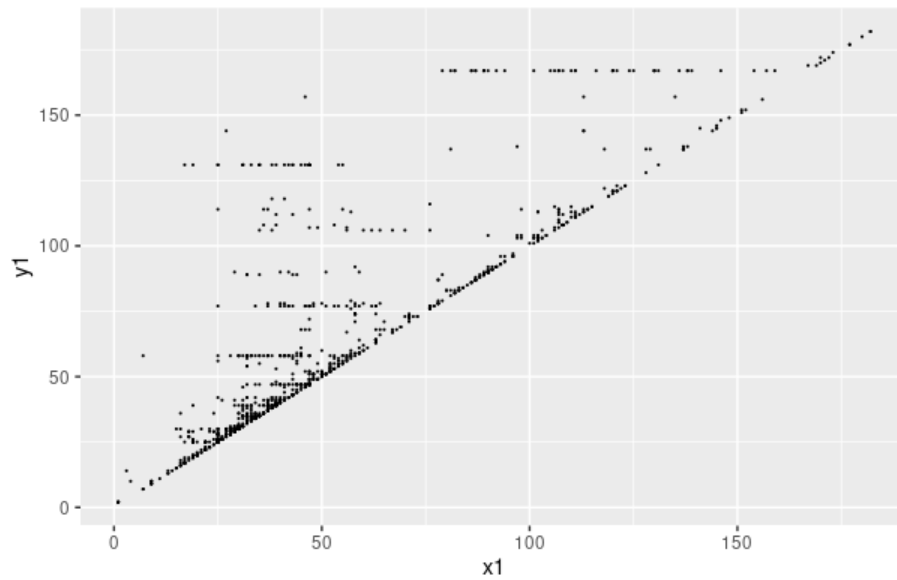


Figure A.11: Archiva

- Observation period: Approximately 175 months
- Total number of bugs: 1151
- Percentage of languages (%): Java79.7, JavaScript9.5, TypeScript3.8, HTML3.6, CSS2.4, shell0.4, other0.6
- URL for the software overview: <https://archiva.apache.org/>

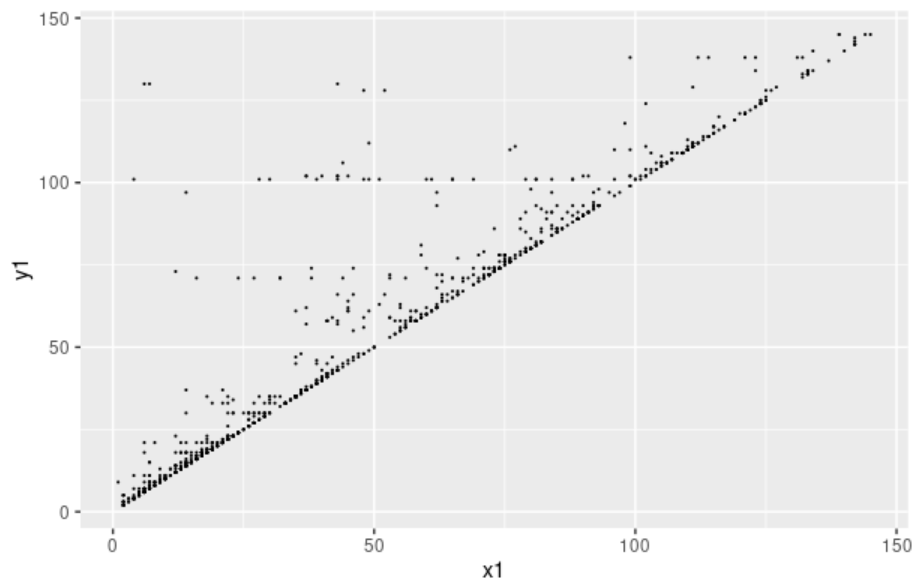


Figure A.12: Aries

- Observation period: Approximately 150 months
- Total number of bugs: 1198
- Percentage of languages (%): Java96.6, JavaScript2.0, HTML0.9, CSS0.3, Groovy0.1, shell0.1
- URL for the software overview: <https://aries.apache.org/documentation/index.HTML>

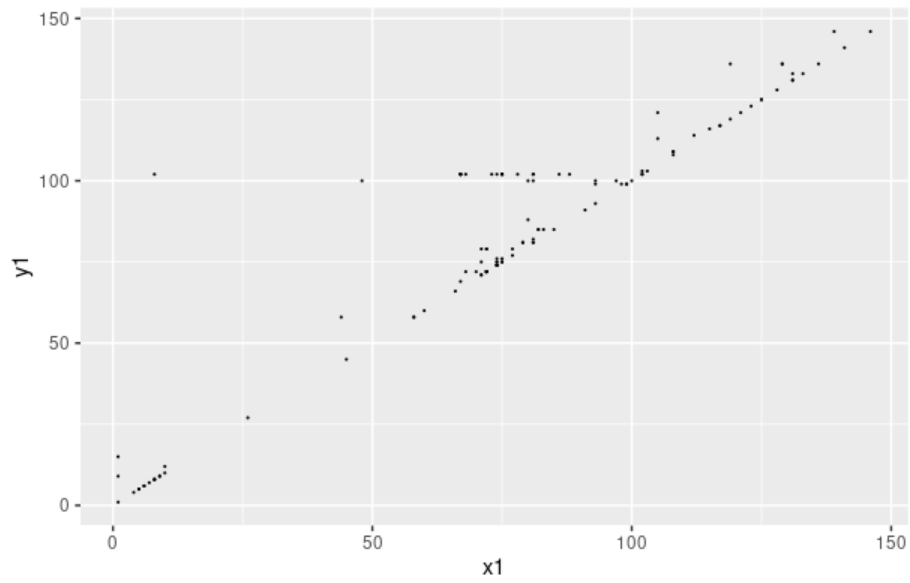


Figure A.13: ASF

- Observation period: Approximately 150 months
- Total number of bugs: 153
- Percentage of languages (%): shell100
- URL for the software overview: <https://www.apache.org/>

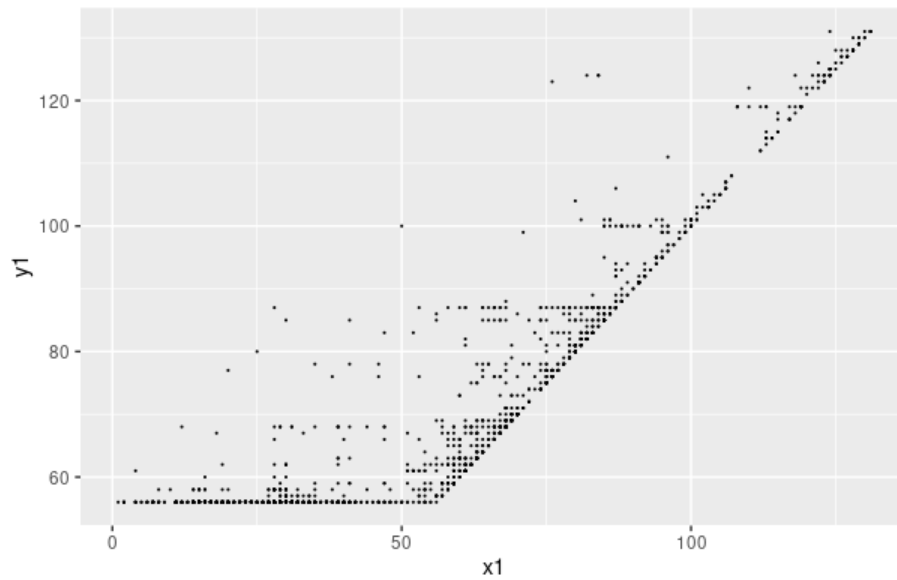


Figure A.14: AsterixDB

- Observation period: Approximately 140 months
- Total number of bugs: 2276
- Percentage of languages (%): Java95.7,python1.0,shell0.8,TypeScript0.7,JavaScript0.7,HTML0.5,other 0.6
- URL for the software overview: <https://asterixdb.apache.org/>

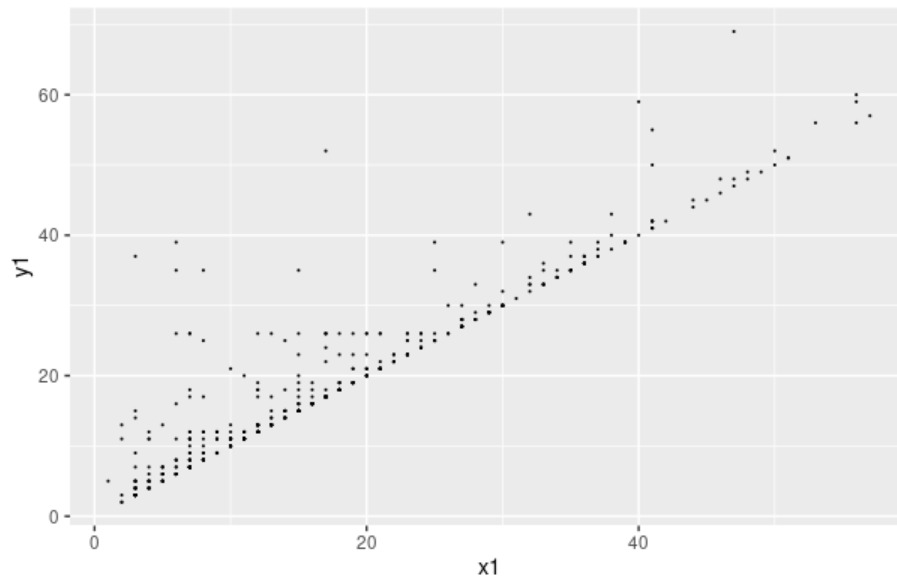


Figure A.15: Aurora

- Observation period: Approximately 70 months
- Total number of bugs: 628
- Percentage of languages (%): Java63.6, python28.0, JavaScript3.8, shell1.6, Thrift1.0, other2.0
- URL for the software overview: <https://github.com/apache/aurora>

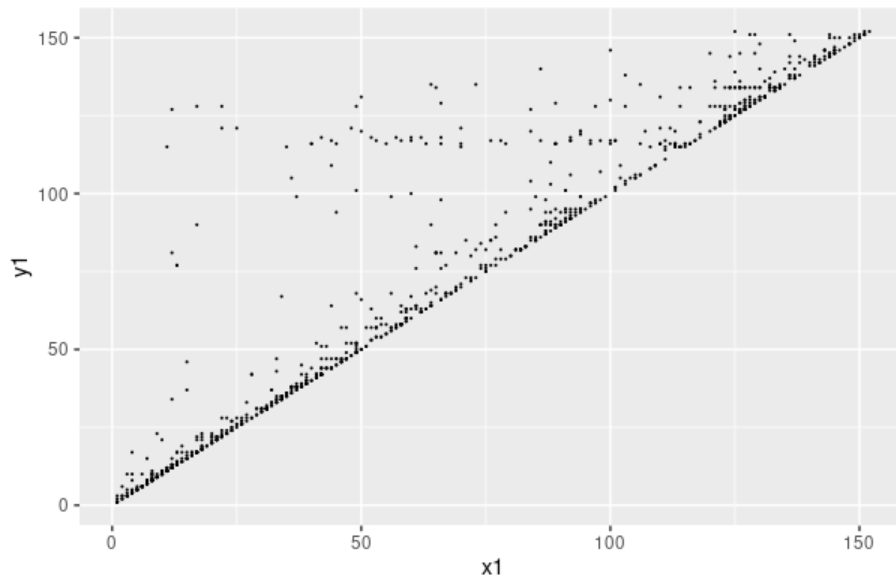
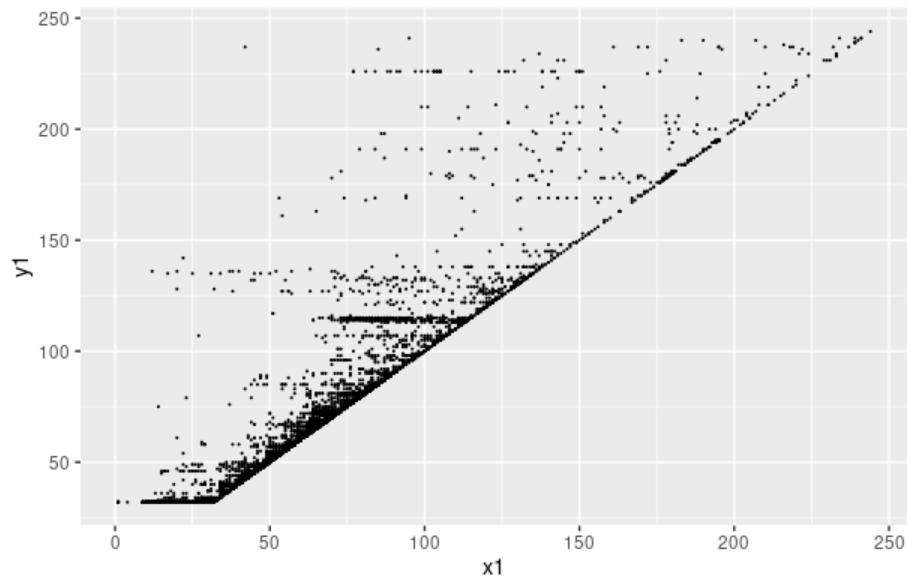


Figure A.16: Avro

- Observation period: Approximately 150 months
- Total number of bugs: 1446
- Percentage of languages (%): Java44.4, C#13.5, C11.0, C++8.7, python4.6, RUST4.2, other13.6
- URL for the software overview: <https://avro.apache.org/>

**Figure A.17:** Axis

- Observation period: Approximately 250 months
- Total number of bugs: 9306
- Percentage of languages (%): Java93.0, XSLT6.5, HTML0.2, batchfile0.1, shell0.1, other0.1
- URL for the software overview: <https://axis.apache.org/>

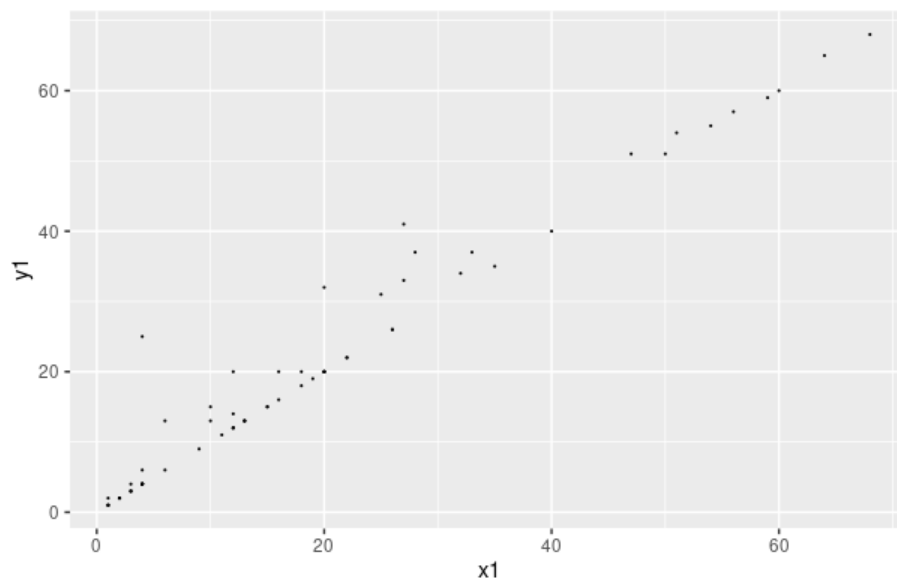


Figure A.18: Bahir

- Observation period: Approximately 70 months
- Total number of bugs: 84
- Percentage of languages (%): scala82.6, Java8.9, python4.4, shell4.1
- URL for the software overview: <https://bahir.apache.org/>

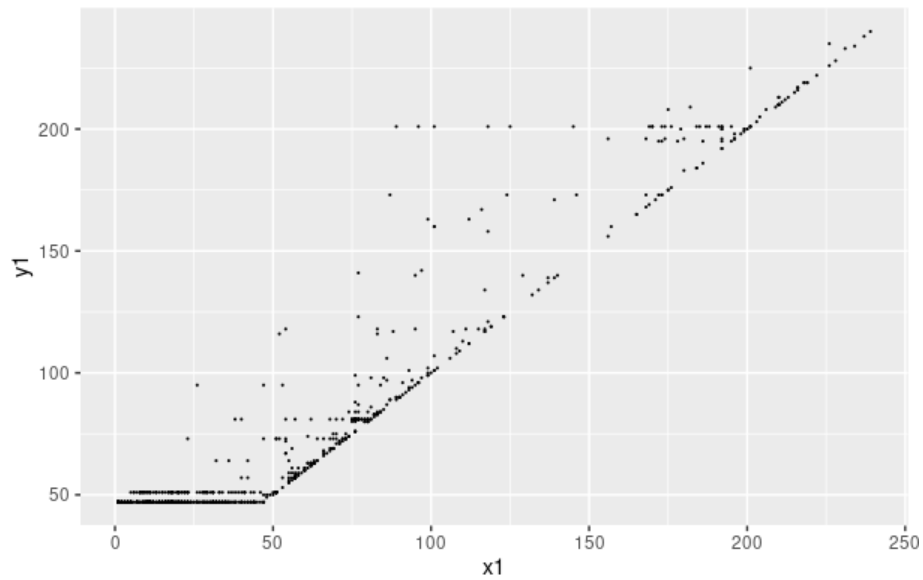


Figure A.19: Batik

- Observation period: Approximately 250 months
- Total number of bugs: 1202
- Percentage of languages (%): Java99.3, XSLT0.3, HTML0.2, JavaScript0.1, batchfile0.1
- URL for the software overview: <https://github.com/apache/xmlgraphics-batik>

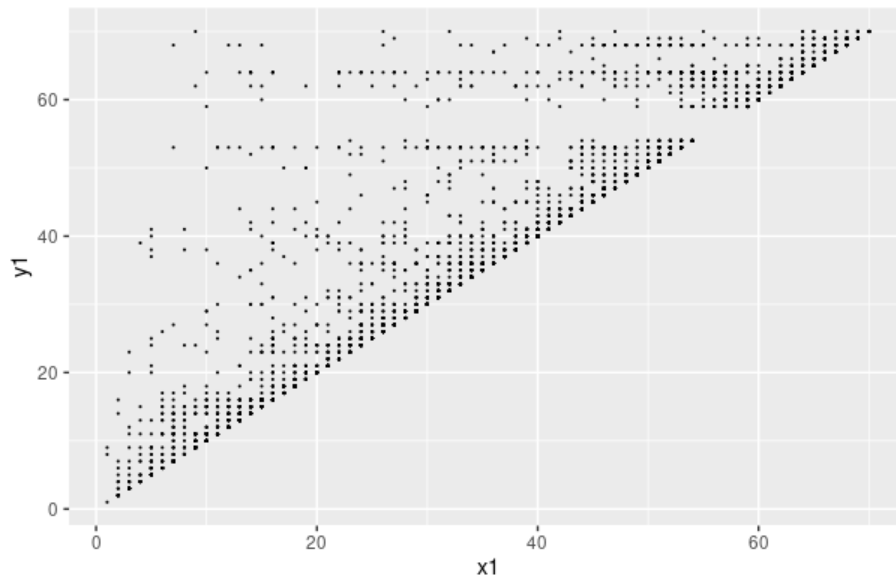


Figure A.20: Beam

- Observation period: Approximately 70 months
- Total number of bugs: 5549
- Percentage of languages (%): Java70.4, python17.7, GO7.8, Groovy1.5, shell0.6
- URL for the software overview: <https://beam.apache.org/>

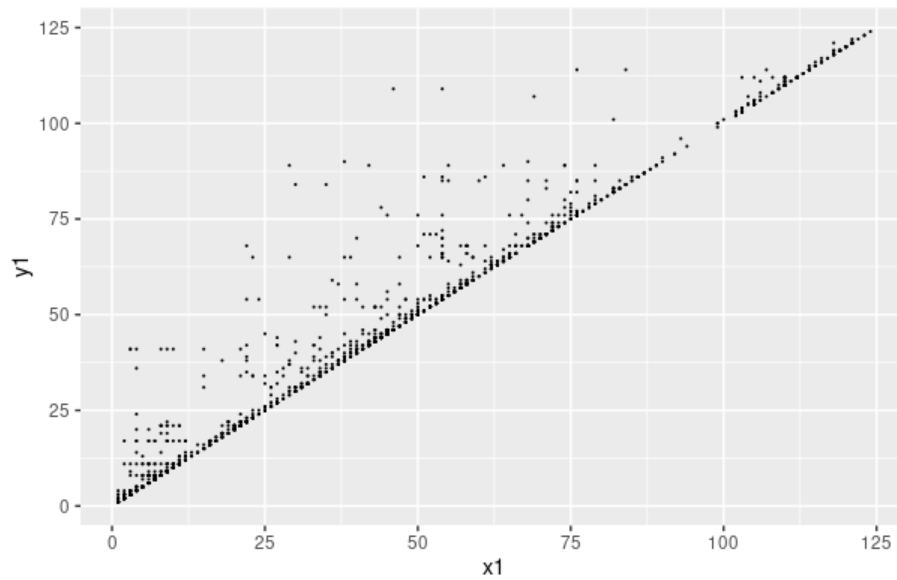


Figure A.21: Bigtop

- Observation period: Approximately 125 months
- Total number of bugs: 1901
- Percentage of languages (%): Groovy22.4, Java21.2, python20.9, shell19.6, scala3.3, other12.6
- URL for the software overview: <https://bigtop.apache.org/>

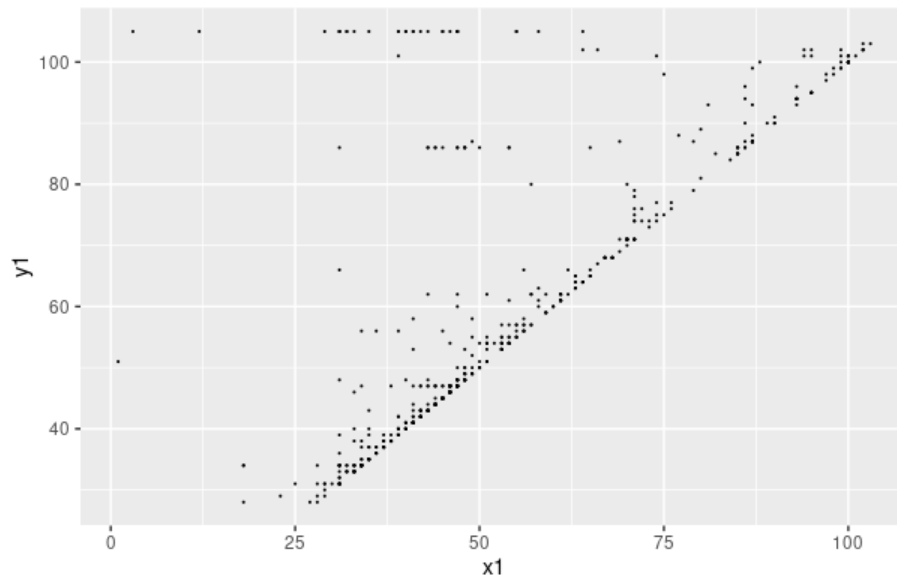


Figure A.22: Bookkeeper

- Observation period: Approximately 100 months
- Total number of bugs: 570
- Percentage of languages (%): Java95.9, shell1.5, python1.4, Groovy0.4, roff0.3, JavaScript0.2, other0.3
- URL for the software overview: <https://bookkeeper.apache.org/>

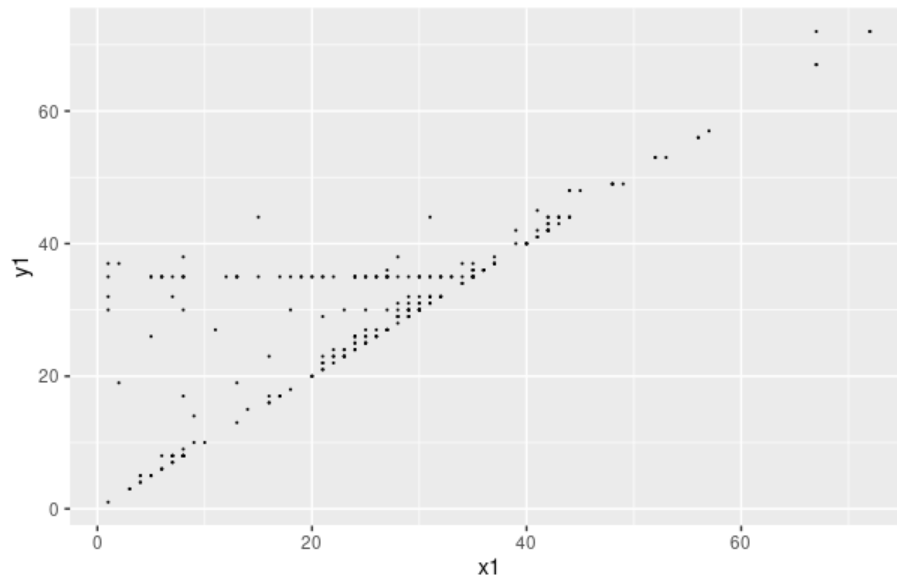


Figure A.23: Brooklyn

- Observation period: Approximately 70 months
- Total number of bugs: 454
- Percentage of languages (%): dockerfile100
- URL for the software overview: <https://brooklyn.apache.org/>

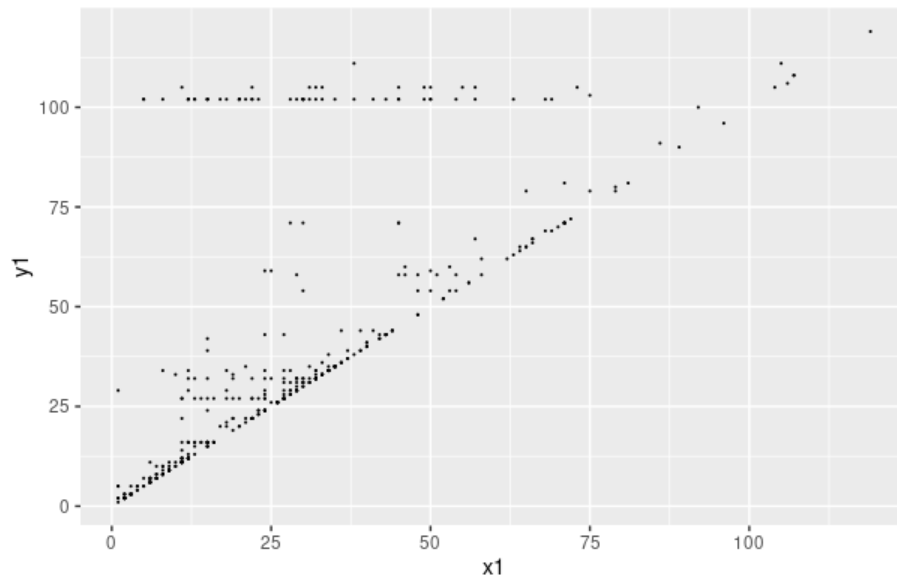


Figure A.24: Buildr

- Observation period: Approximately 120 months
- Total number of bugs: 433
- Percentage of languages (%): Ruby97.0, Java1.5, XSLT1.3, other0.2
- URL for the software overview: <https://buildr.apache.org/>

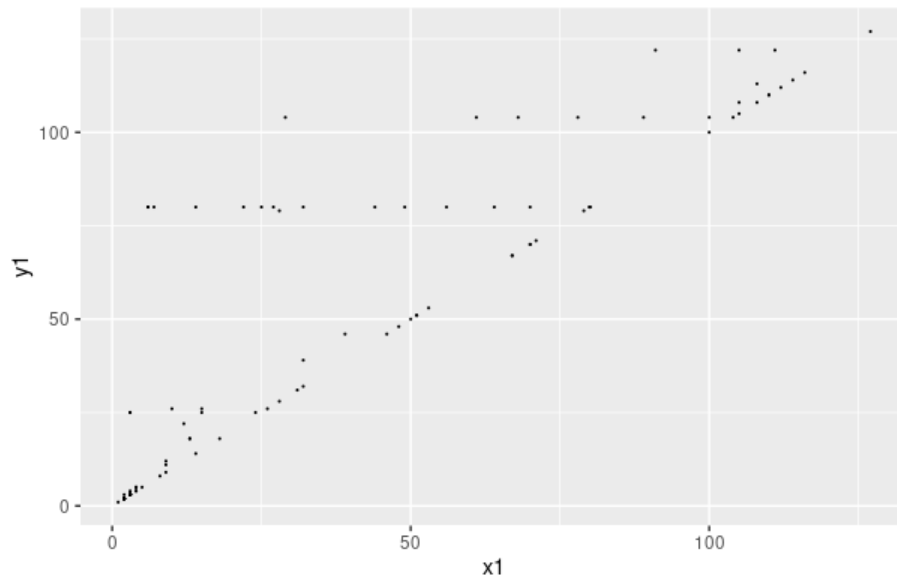


Figure A.25: BVal

- Observation period: Approximately 125 months
- Total number of bugs: 121
- Percentage of languages (%): Java100
- URL for the software overview: <https://bval.apache.org/>

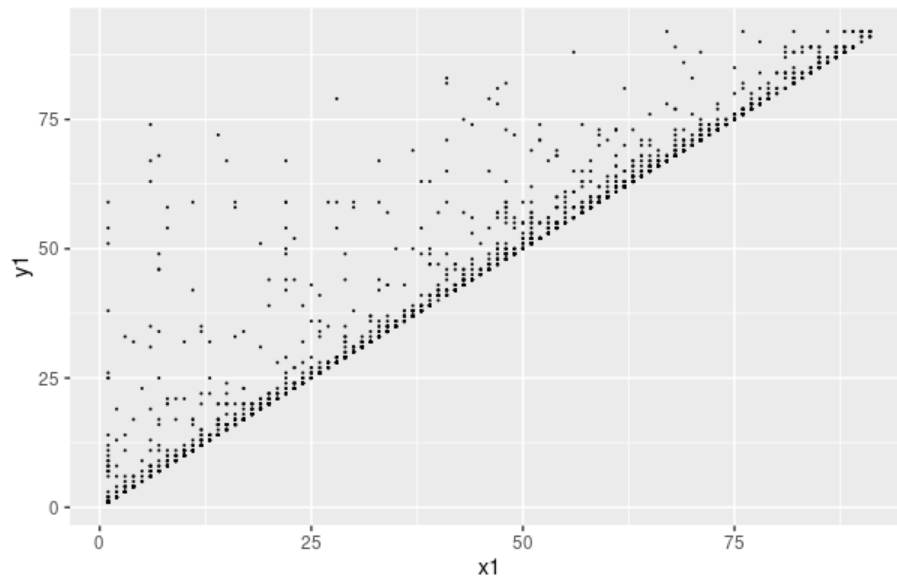


Figure A.26: Calcite

- Observation period: Approximately 90 months
- Total number of bugs: 3096
- Percentage of languages (%): Java98.9, HTML0.1, freemarker0.1, other0.9
- URL for the software overview: <https://calcite.apache.org/>

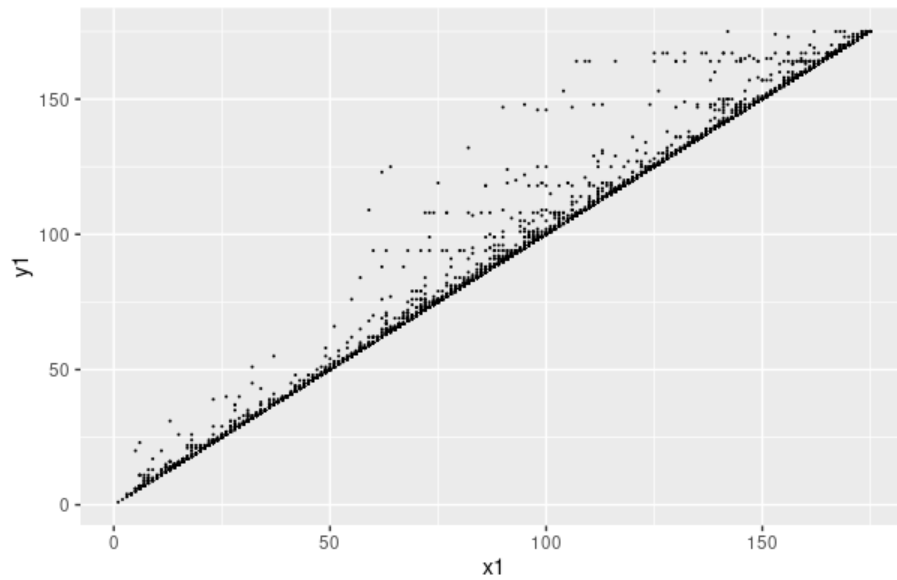


Figure A.27: Camel

- Observation period: Approximately 175 months
- Total number of bugs: 5551
- Percentage of languages (%): Java99.1, Groovy0.3, XSLT0.3, HTML0.2, JavaScript0.1
- URL for the software overview: <https://camel.apache.org/>

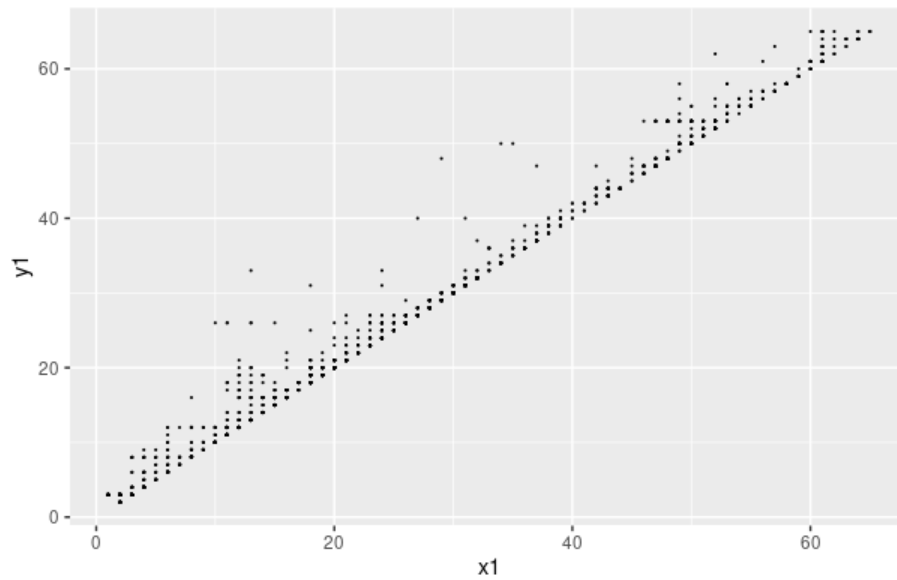


Figure A.28: CarbonData

- Observation period: Approximately 60 months
- Total number of bugs: 2384
- Percentage of languages (%): scala58.7, Java38.8, python1.8, C++0.5, Thrift0.1
- URL for the software overview: <https://carbondata.apache.org/>

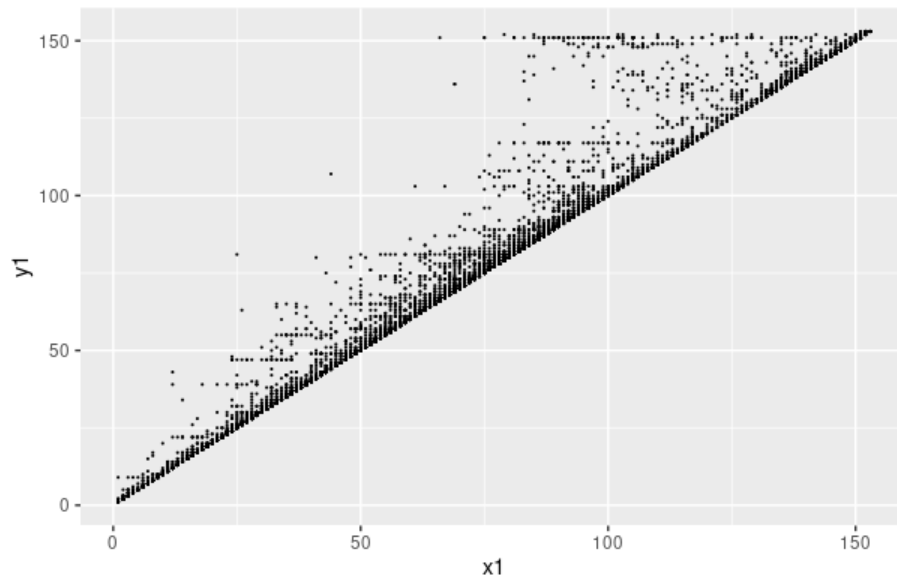


Figure A.29: Cassandra

- Observation period: Approximately 150 months
- Total number of bugs: 9406
- Percentage of languages (%): Java96.5, python1.9, HTML0.9, shell0.4, other0.3
- URL for the software overview: https://cassandra.apache.org/_/index.html

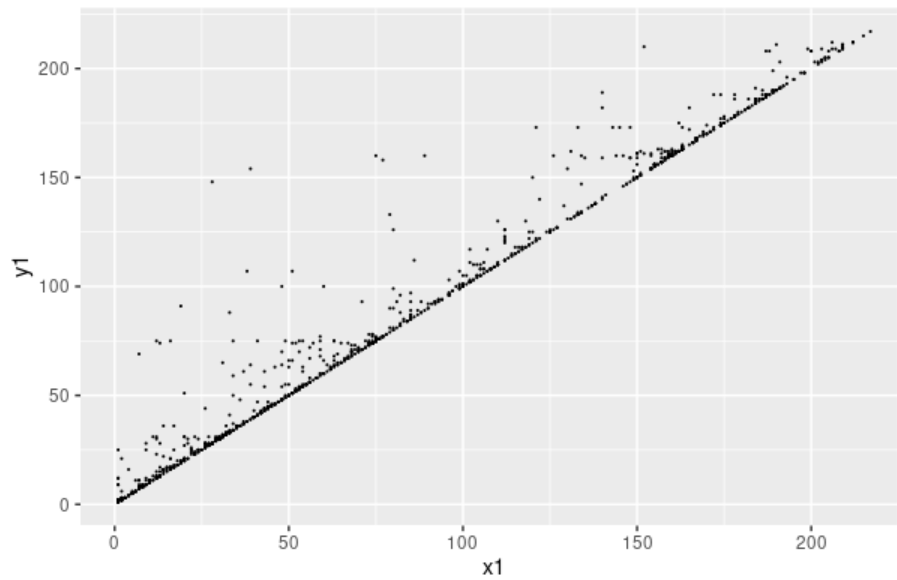


Figure A.30: Cayenne

- Observation period: Approximately 200 months
- Total number of bugs: 1363
- Percentage of languages (%): Java99.8, other0.2
- URL for the software overview: <https://cayenne.apache.org/>

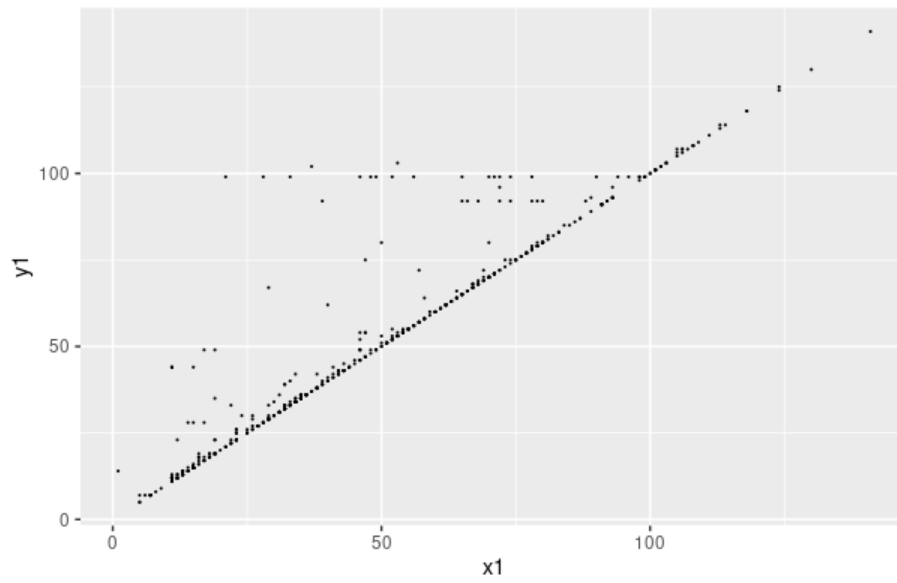


Figure A.31: Chemistry

- Observation period: Approximately 150 months
- Total number of bugs: 609
- Percentage of languages (%): Java100
- URL for the software overview: <https://chemistry.apache.org/>

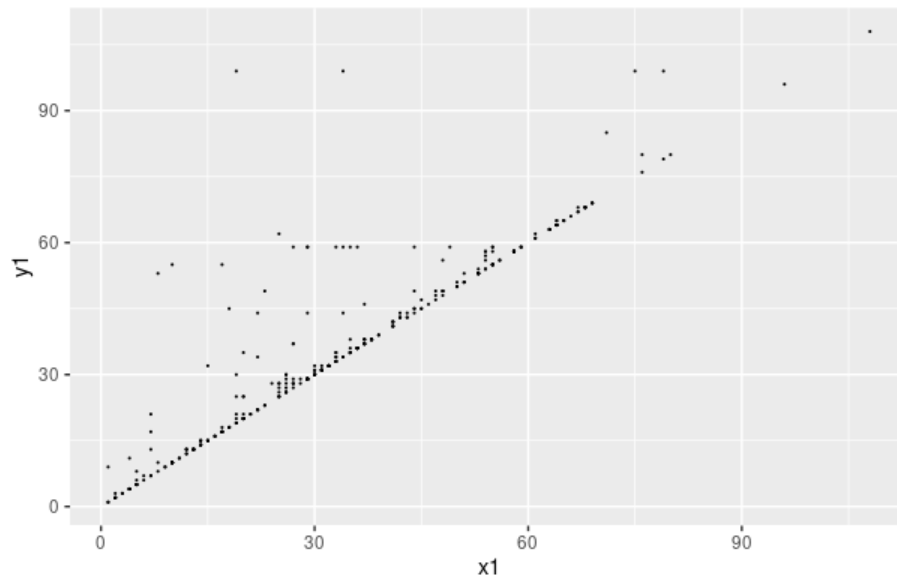


Figure A.32: Click

- Observation period: Approximately 100 months
- Total number of bugs: 369
- Percentage of languages (%): Java91.7, JavaScript5.4, HTML1.6, CSS1.3
- URL for the software overview: <https://github.com/apache/click>

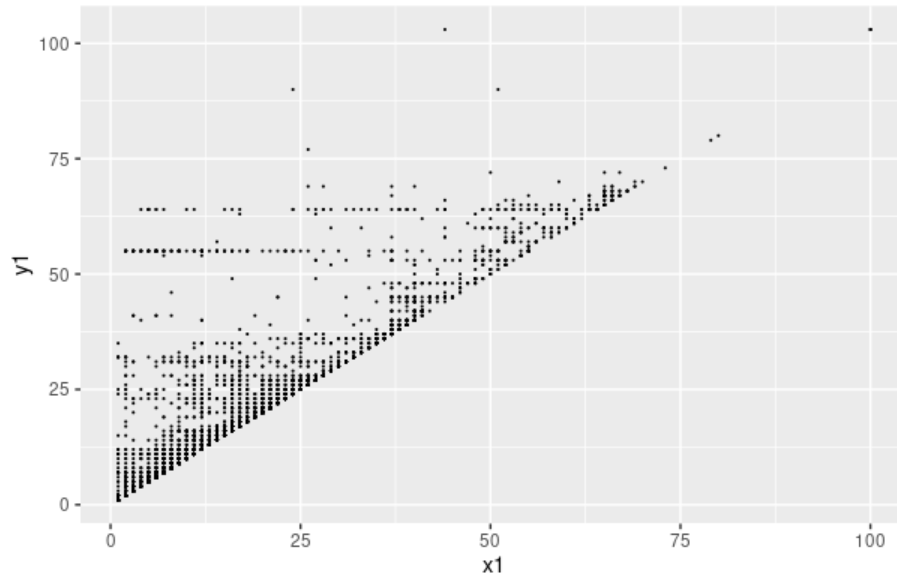


Figure A.33: Cloudstack

- Observation period: Approximately 100 months
- Total number of bugs: 7861
- Percentage of languages (%): Java66.7, python21.9, C#3.9, JavaScript1.7, shell1.3, other4.5
- URL for the software overview: <https://cloudstack.apache.org/>

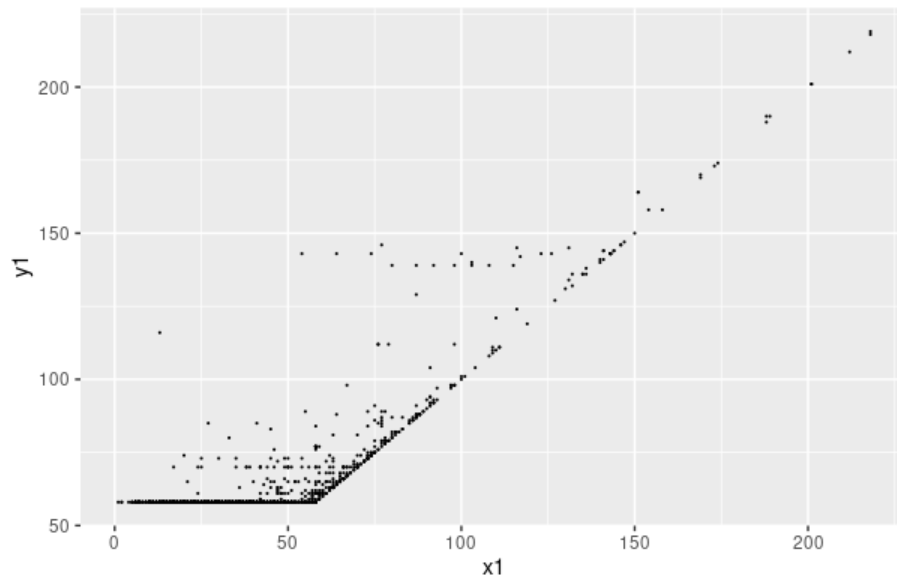


Figure A.34: Cocoon

- Observation period: Approximately 200 months
- Total number of bugs: 1771
- Percentage of languages (%): Java76.7,XSLT9.6,JavaScript8.9,HTML2.4,php1.5,CSS0.8,other0.1
- URL for the software overview: <https://cocoon.apache.org/>

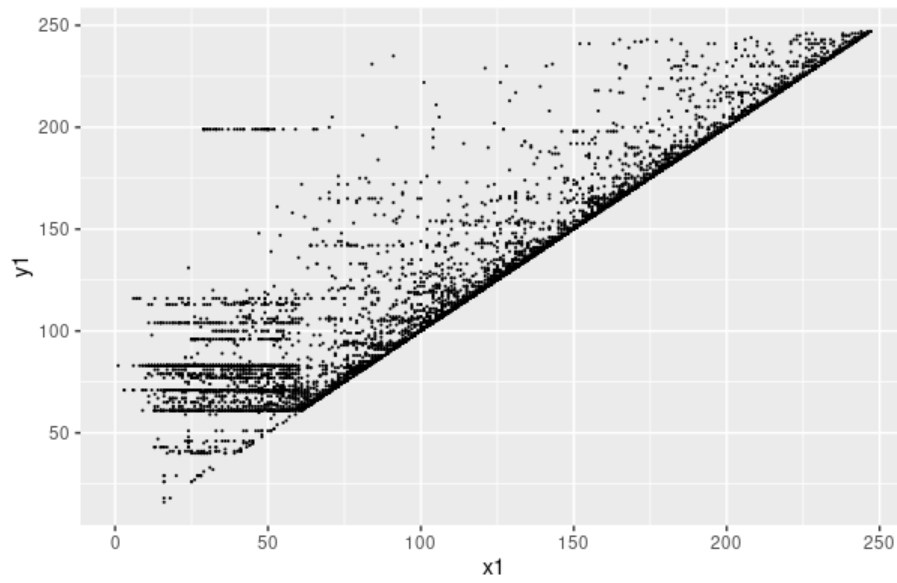


Figure A.35: Commons

- Observation period: Approximately 250 months
- Total number of bugs: 8769
- Percentage of languages (%): Java100
- URL for the software overview: <https://commons.apache.org/>

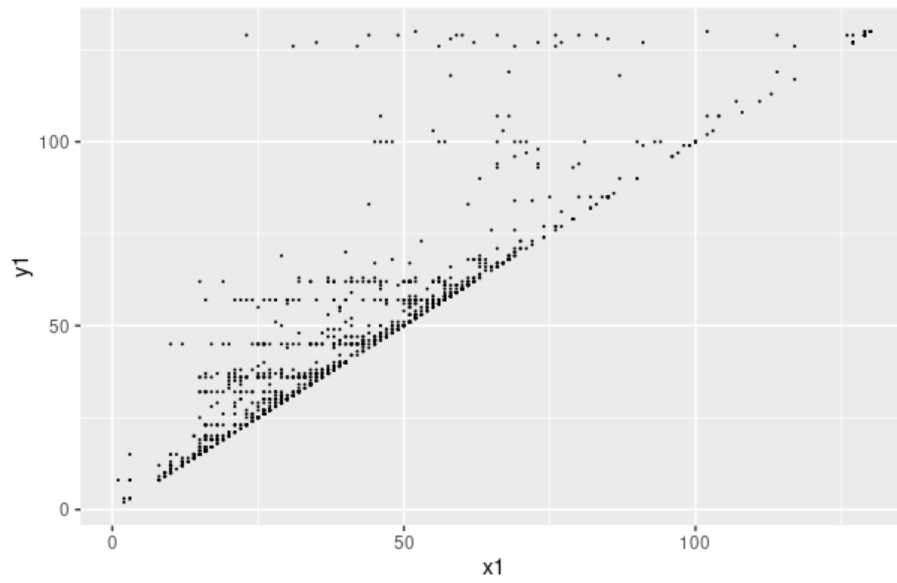


Figure A.36: Continuum

- Observation period: Approximately 130 months
- Total number of bugs: 1541
- Percentage of languages (%): Java98.1, other1.9
- URL for the software overview: <https://github.com/apache/continuum>

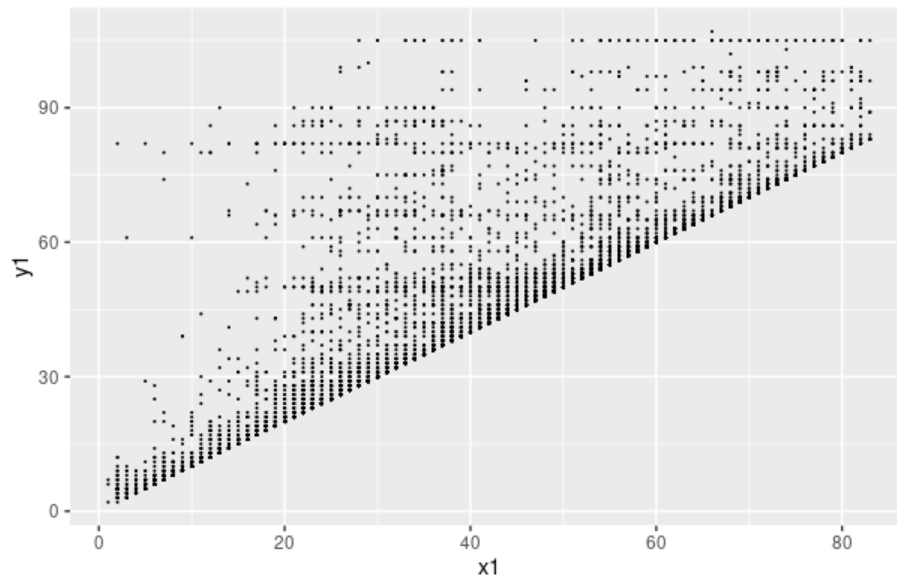


Figure A.37: Cordova

- Observation period: Approximately 80 months
- Total number of bugs: 8781
- Percentage of languages (%): JavaScript100
- URL for the software overview: <https://cordova.apache.org/>

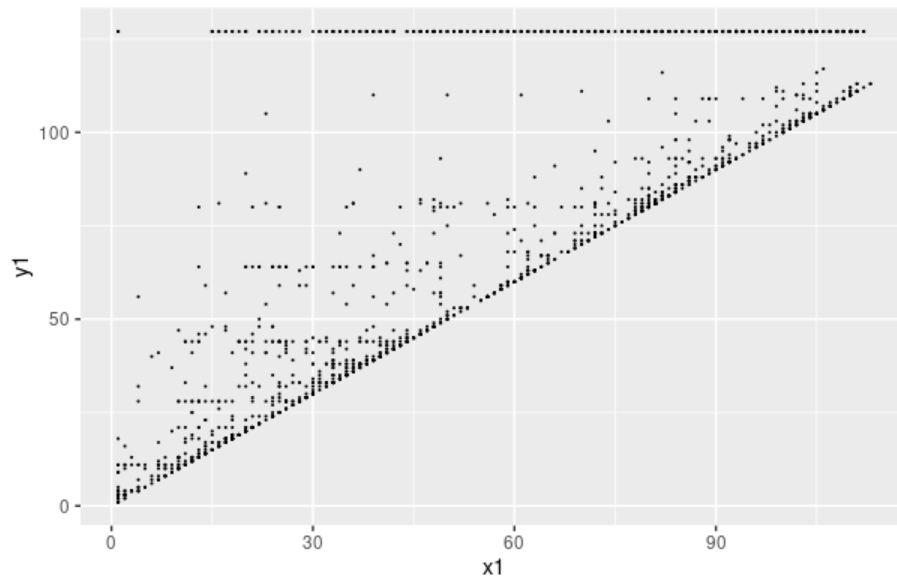


Figure A.38: CouchDB

- Observation period: Approximately 125 months
- Total number of bugs: 1878
- Percentage of languages (%): ELANG68.1, other13.3, JavaScript9.1, python4.9
- URL for the software overview: <https://couchdb.apache.org/>

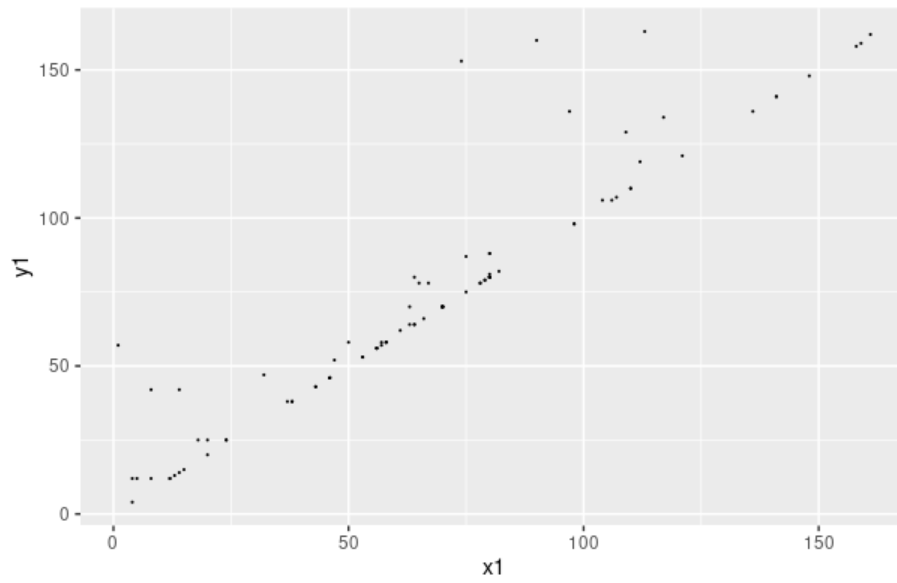


Figure A.39: Creadur

- Observation period: Approximately 150 months
- Total number of bugs: 126
- Percentage of languages (%): Java100
- URL for the software overview: <https://creadur.apache.org/>

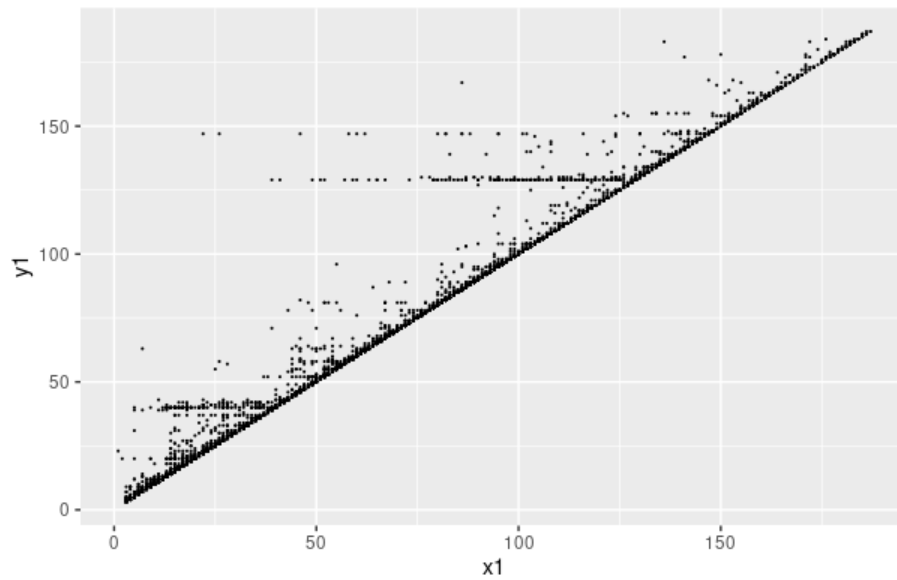


Figure A.40: CXF

- Observation period: Approximately 200 months
- Total number of bugs: 5342
- Percentage of languages (%): Java98.9, JavaScript0.3, XSLT0.3, shell0.2, HTML0.1, batchfile0.1, other0.1
- URL for the software overview: <https://cxf.apache.org/>

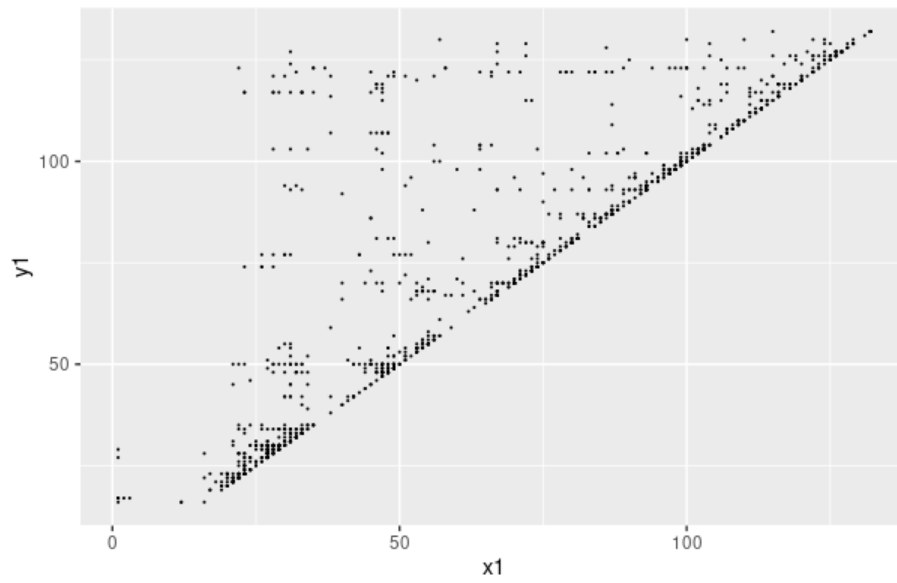


Figure A.41: Daffodil

- Observation period: Approximately 150 months
- Total number of bugs: 1432
- Percentage of languages (%): scala92.3, XSLT2.9, C2.2, Java2.2, shell0.3, HTML0.1
- URL for the software overview: <https://daffodil.apache.org/>

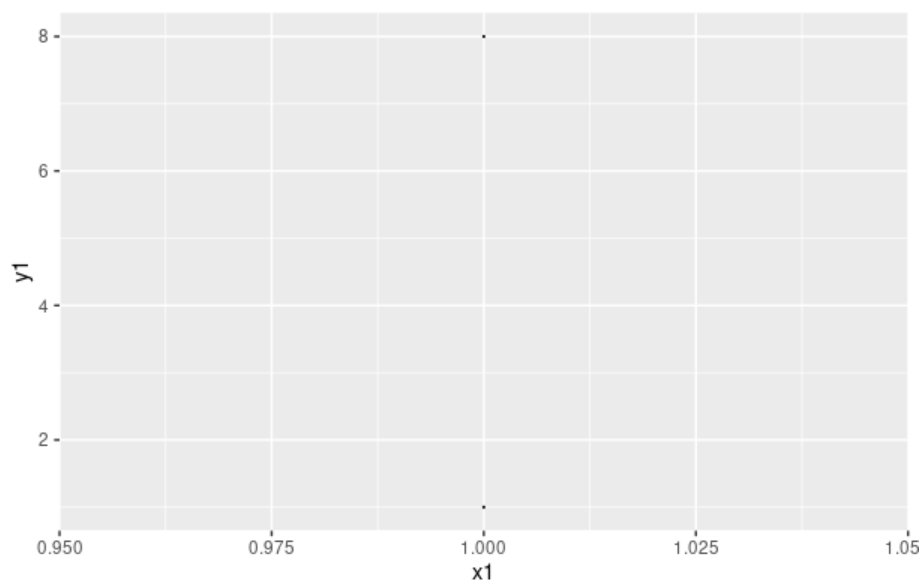
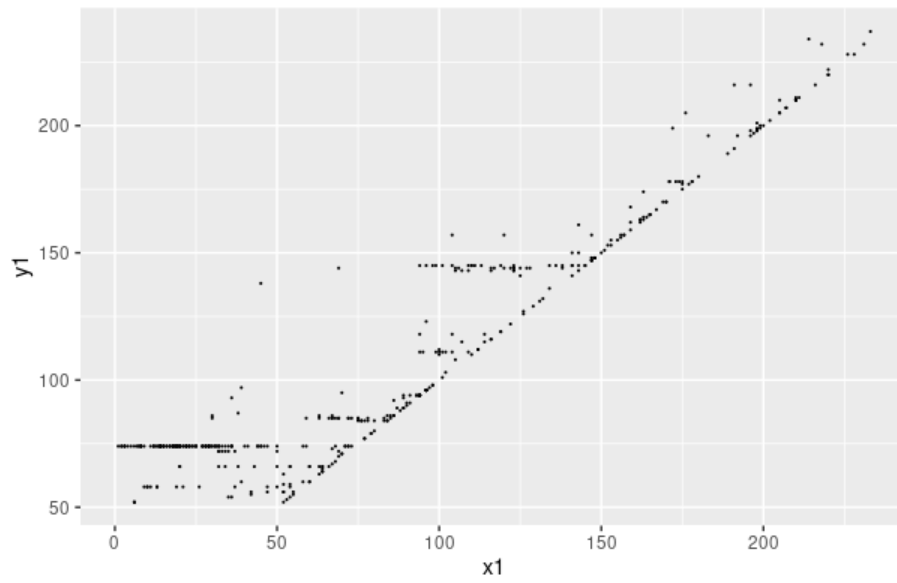


Figure A.42: Datasketches

- Observation period: Approximately 8 months
- Total number of bugs: 2
- Percentage of languages (%): Java100
- URL for the software overview: <https://datasketches.apache.org/>

**Figure A.43:** DB

- Observation period: Approximately 250 months
- Total number of bugs: 452
- Percentage of languages (%): Java98.2 HTML1.2 other0.6
- URL for the software overview: <https://db.apache.org/>

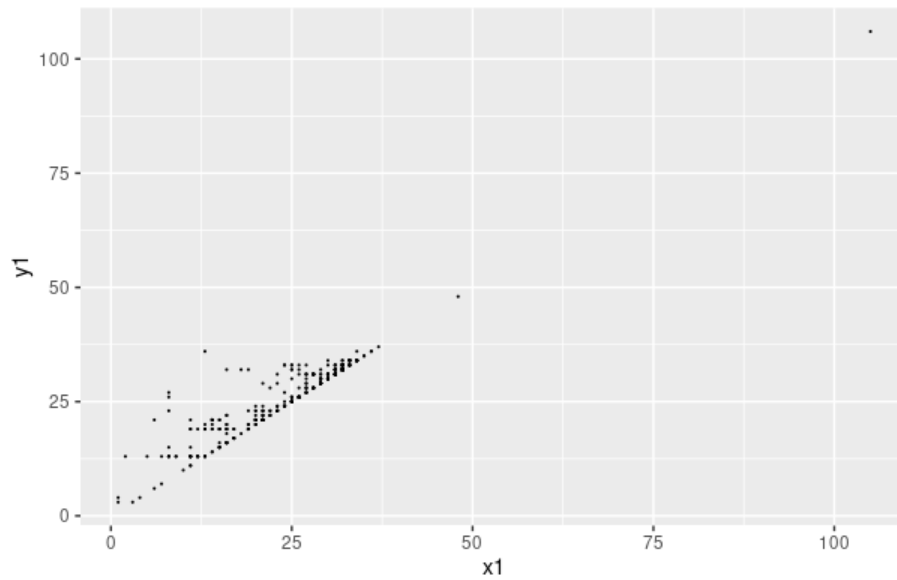


Figure A.44: Deltacloud

- Observation period: Approximately 100 months
- Total number of bugs: 483
- Percentage of languages (%): Ruby 52.0, HTML31.3, Java7.5, CSS5.5, JavaScript1.3, other2.4
- URL for the software overview: <https://deltacloud.apache.org/>

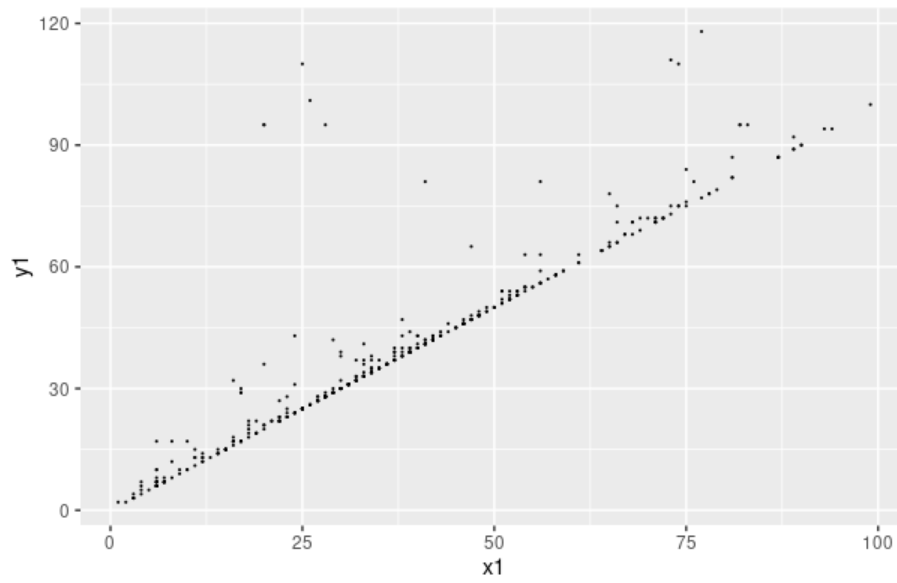


Figure A.45: Deltaspikes

- Observation period: Approximately 120 months
- Total number of bugs: 465
- Percentage of languages (%): Java97.3, HTML2.1, other0.6
- URL for the software overview: <https://deltaspikes.apache.org/>

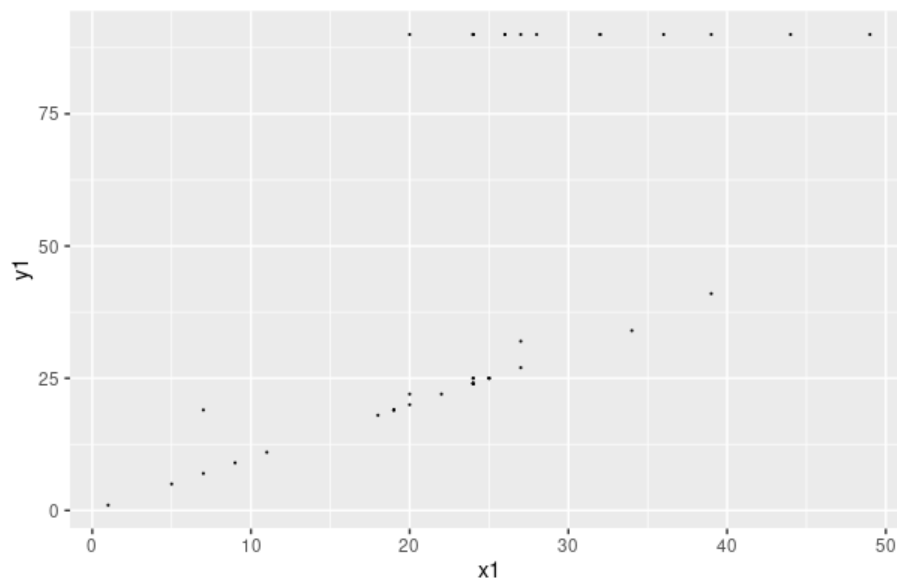


Figure A.46: Devicemap

- Observation period: Approximately 90 months
- Total number of bugs: 42
- Percentage of languages (%): JavaScript92.2, CSS4.4, HTML2.7, shell0.7
- URL for the software overview: <https://github.com/apache/devicemap-browsermap>

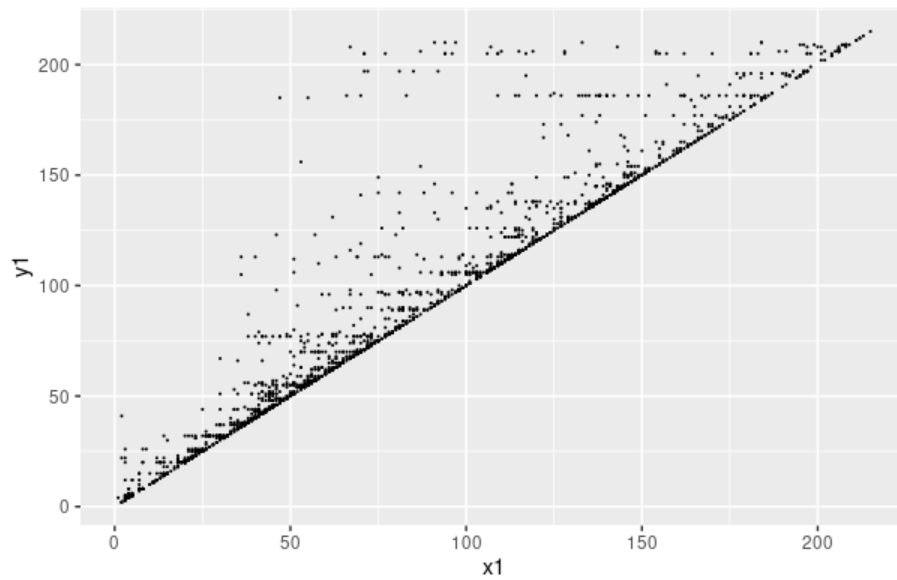


Figure A.47: Directory

- Observation period: Approximately 200 months
- Total number of bugs: 2640
- Percentage of languages (%): shell100
- URL for the software overview: <https://directory.apache.org/>

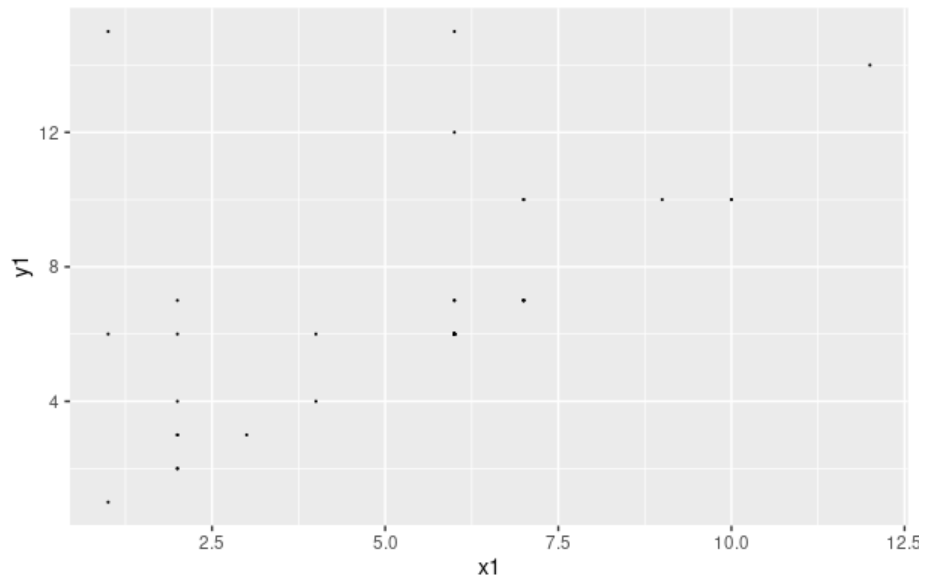


Figure A.48: Distributedlog

- Observation period: Approximately 15 months
- Total number of bugs: 55
- Percentage of languages (%): Java91.8, shell3.9, HTML1.4, python1.2, CSS1.2, Ruby0.3, other0.2
- URL for the software overview: <https://github.com/apache/distributedlog>

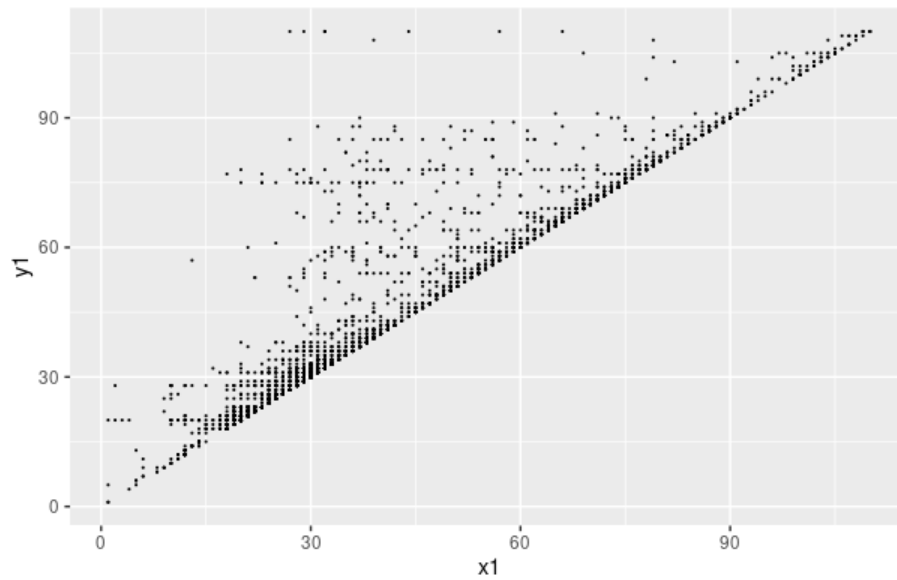


Figure A.49: Drill

- Observation period: Approximately 100 months
- Total number of bugs: 5347
- Percentage of languages (%): Java96.8, C++1.7, freemarker0.6, shell0.4, JavaScript0.2, C0.1, other0.2
- URL for the software overview: <https://drill.apache.org/>

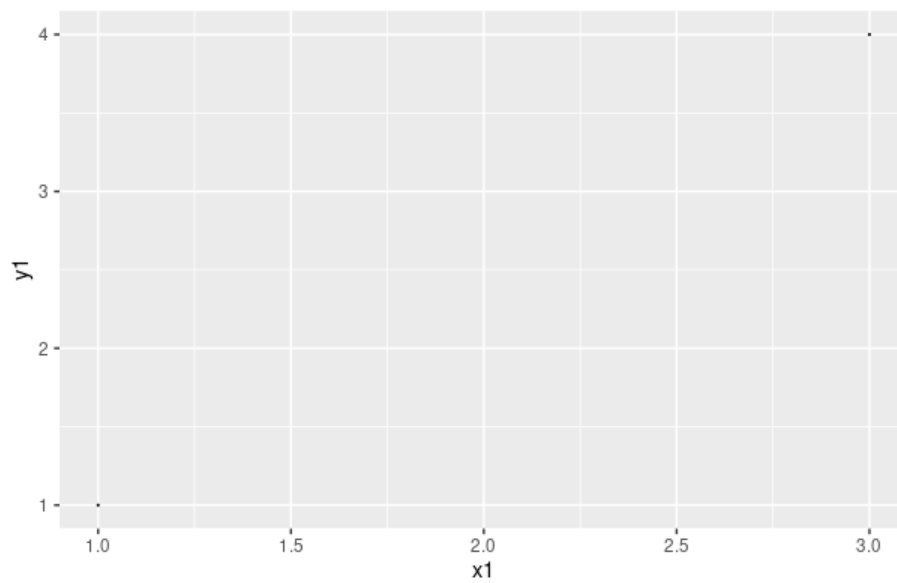
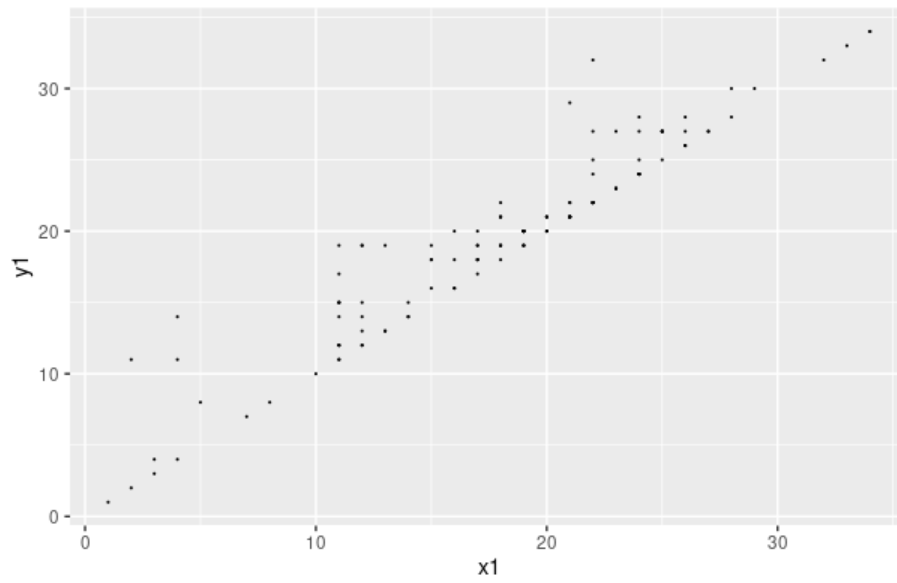


Figure A.50: ECharts

- Observation period: Approximately 4 months
- Total number of bugs: 8
- Percentage of languages (%): TypeScript89.8, JavaScript10.0, other0.2
- URL for the software overview: <https://echarts.apache.org/en/index.html>.

**Figure A.51:** ESME

- Observation period: Approximately 35 months
- Total number of bugs: 139
- Percentage of languages (%): scala74.1, HTML12.9, CSS6.9, JavaScript6.1
- URL for the software overview: <https://github.com/apache/esme>

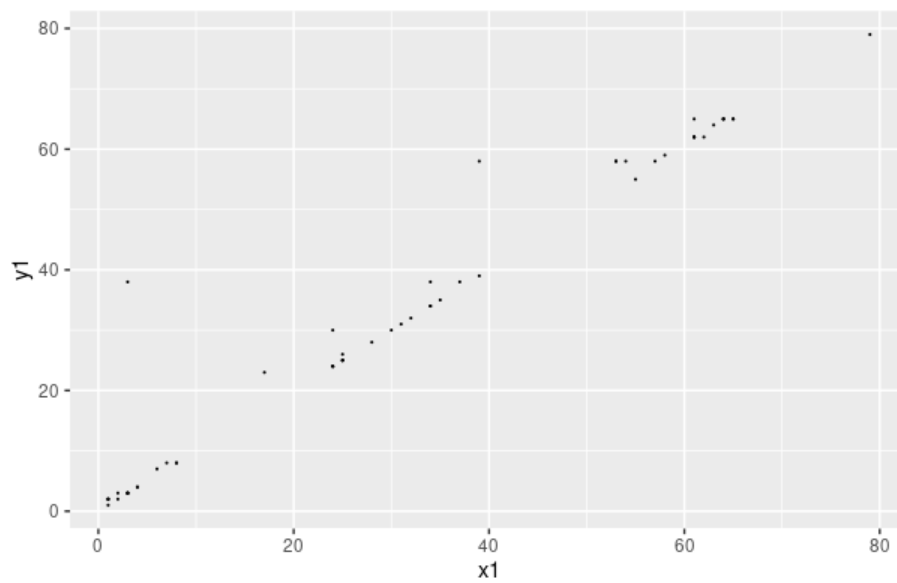


Figure A.52: Etch

- Observation period: Approximately 80 months
- Total number of bugs: 75
- Percentage of languages (%): Java61.4, C#36.4, batchfile0.7, shell0.4, other1.1
- URL for the software overview: <https://github.com/apache/etch>

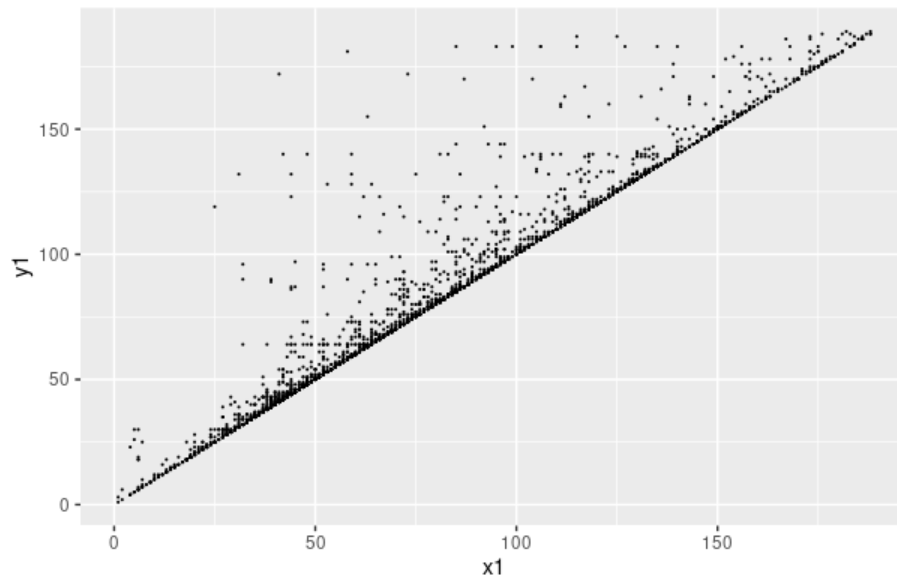


Figure A.53: Felix

- Observation period: Approximately 175 months
- Total number of bugs: 3588
- Percentage of languages (%): Java96.8, HTML1.3, JavaScript1.1, XSLT0.5, CSS0.2, scala0.1
- URL for the software overview: <https://felix.apache.org/documentation/index.HTML>

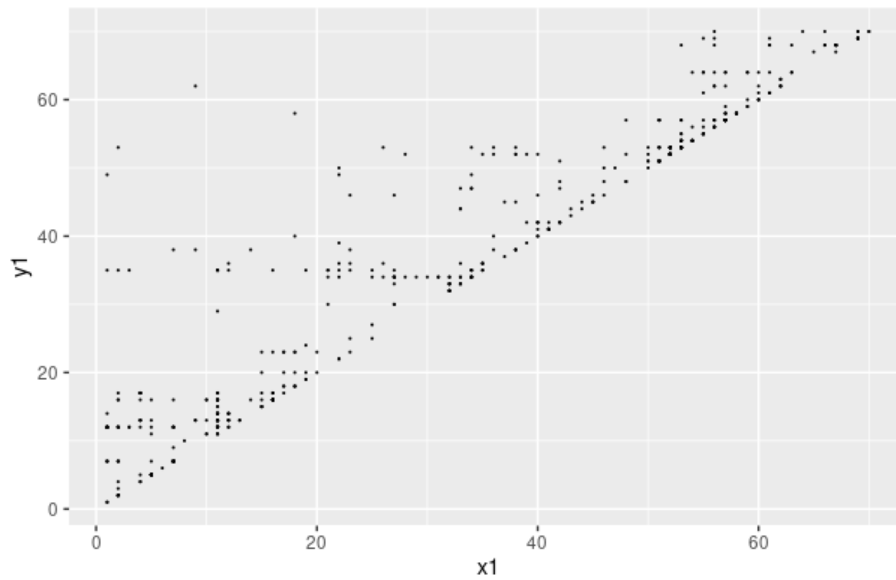
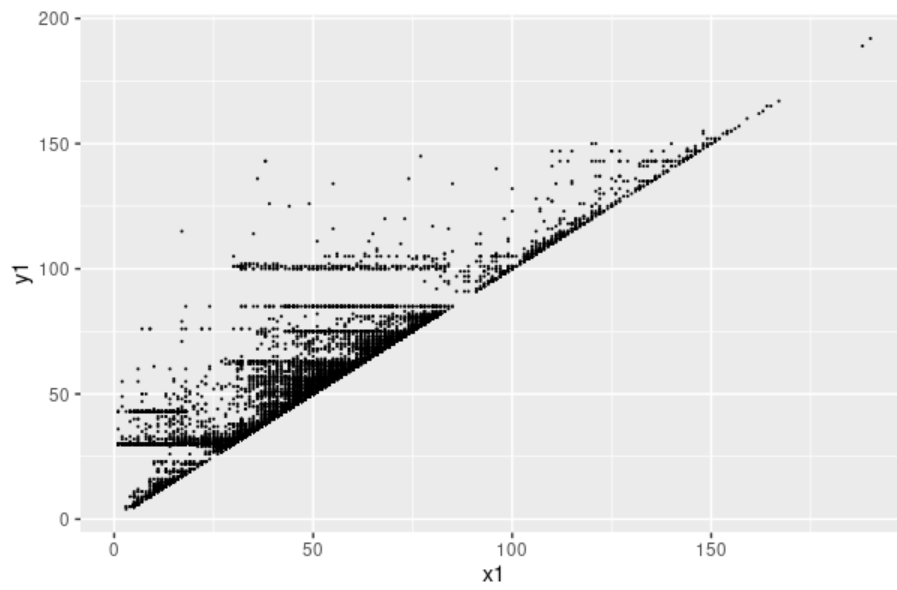


Figure A.54: Fineract

- Observation period: Approximately 75 months
- Total number of bugs: 838
- Percentage of languages (%): Java92.6, HTML7.2, other0.1, CSS0.1
- URL for the software overview: <https://fineract.apache.org/>

**Figure A.55:** Flex

- Observation period: Approximately 200 months
- Total number of bugs: 30691
- Percentage of languages (%): HTML100
- URL for the software overview: <https://flex.apache.org/>

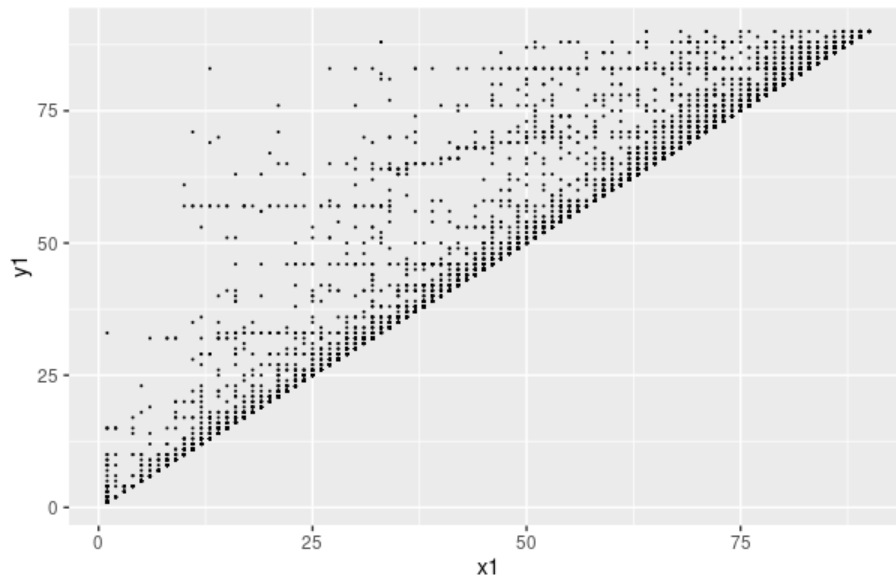


Figure A.56: Flink

- Observation period: Approximately 100 months
- Total number of bugs: 9928
- Percentage of languages (%): Java85.5, scala10.5, python2.5, shell0.5, TypeScript0.3, HTML0.1, other0.6
- URL for the software overview: <https://flink.apache.org/>

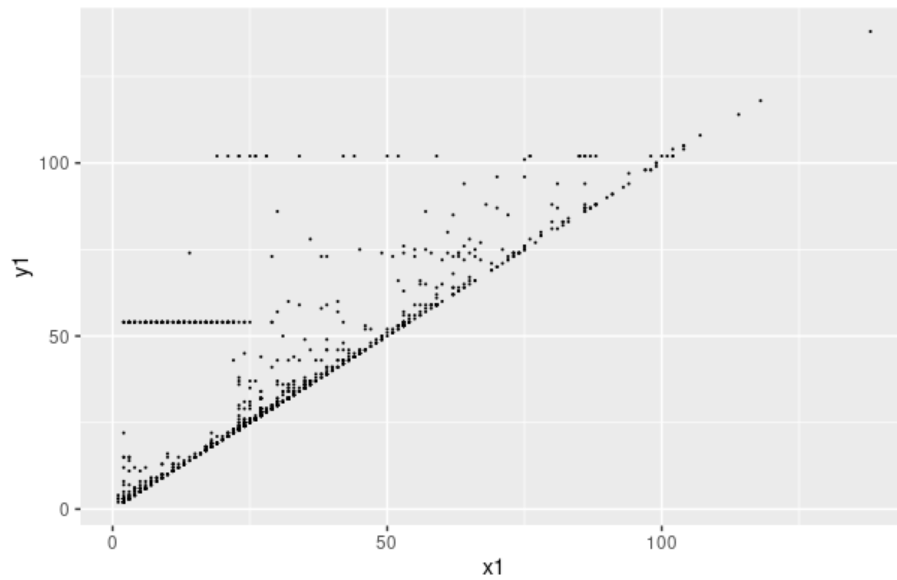


Figure A.57: Flume

- Observation period: Approximately 150 months
- Total number of bugs: 1679
- Percentage of languages (%): Java97.5, richtextformat1.3, shell0.5, other0.3, python0.3, thrift0.1
- URL for the software overview: <https://flume.apache.org/>

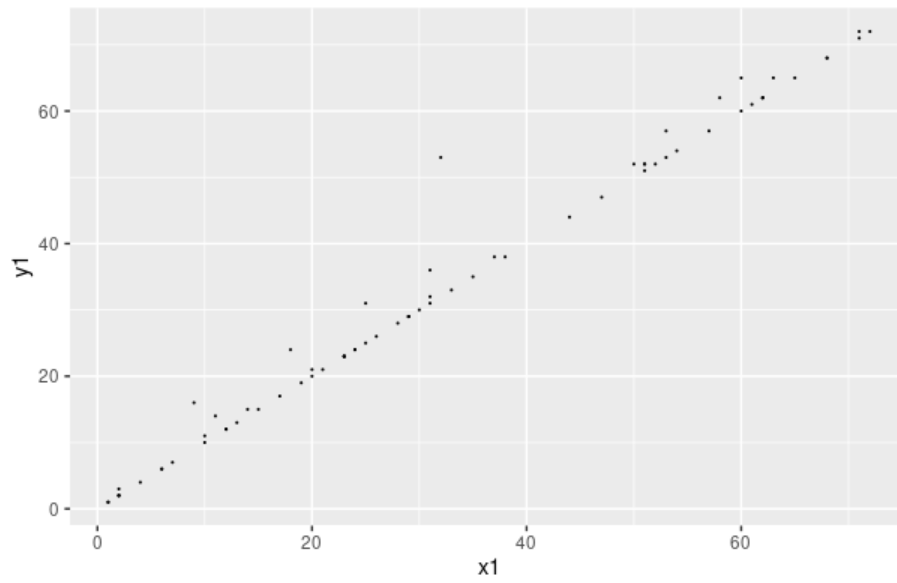


Figure A.58: Freemarker

- Observation period: Approximately 80 months
- Total number of bugs: 93
- Percentage of languages (%): Java93.2, freemarker6.5, HTML0.3
- URL for the software overview: <https://freemarker.apache.org/>

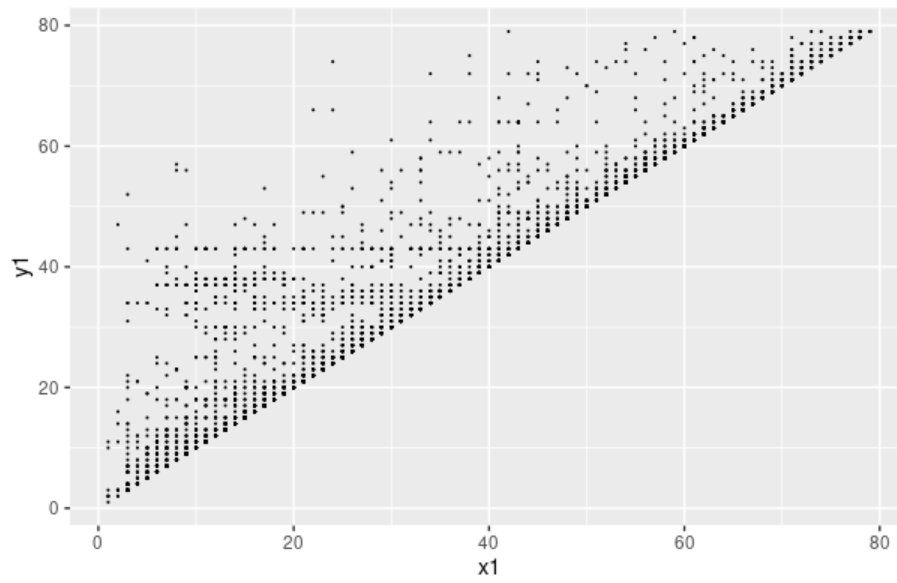


Figure A.59: Geode

- Observation period: Approximately 80 months
- Total number of bugs: 5137
- Percentage of languages (%): Java84.9,HTML9.6,JavaScript4.3,shell0.6,CSS0.3,GO0.1,other0.2
- URL for the software overview: <https://geode.apache.org/>

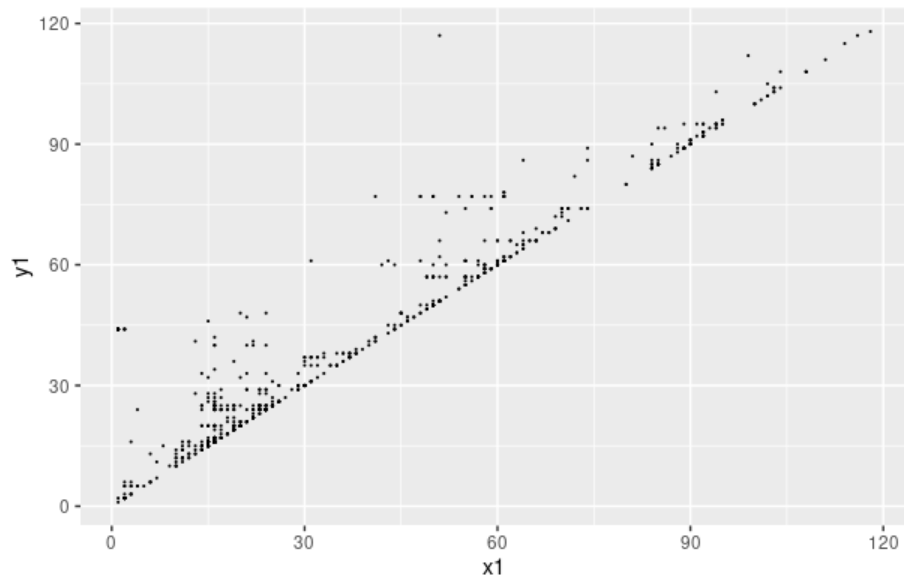


Figure A.60: Geronimo

- Observation period: Approximately 120 months
- Total number of bugs: 802
- Percentage of languages (%): Java90.5, HTML8.4, CSS0.5, shell0.3, batch-file0.3
- URL for the software overview: <https://geronimo.apache.org/>

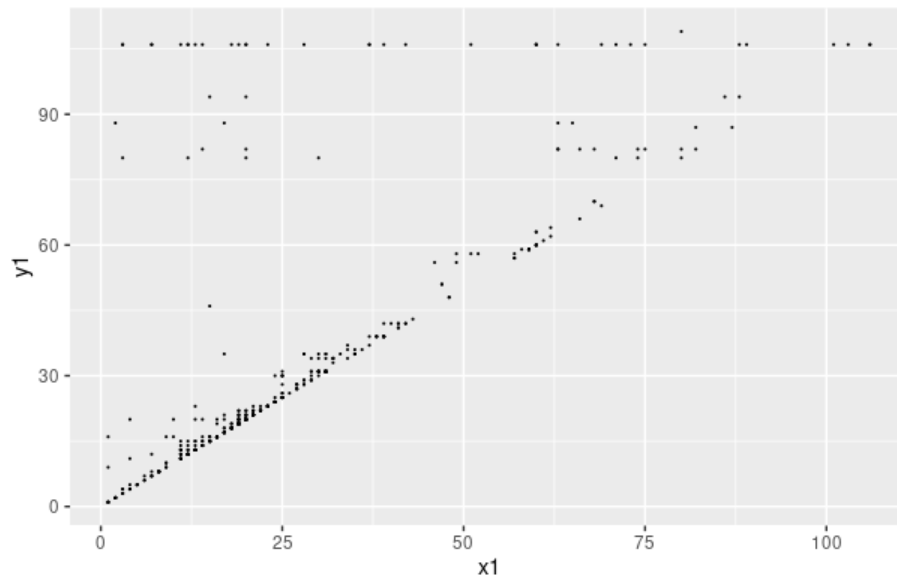


Figure A.61: Giraph

- Observation period: Approximately 100 months
- Total number of bugs: 544
- Percentage of languages (%): Java97.0, JavaScript1.8, shell0.8, HTML0.2, CSS0.2
- URL for the software overview: <https://giraph.apache.org/>

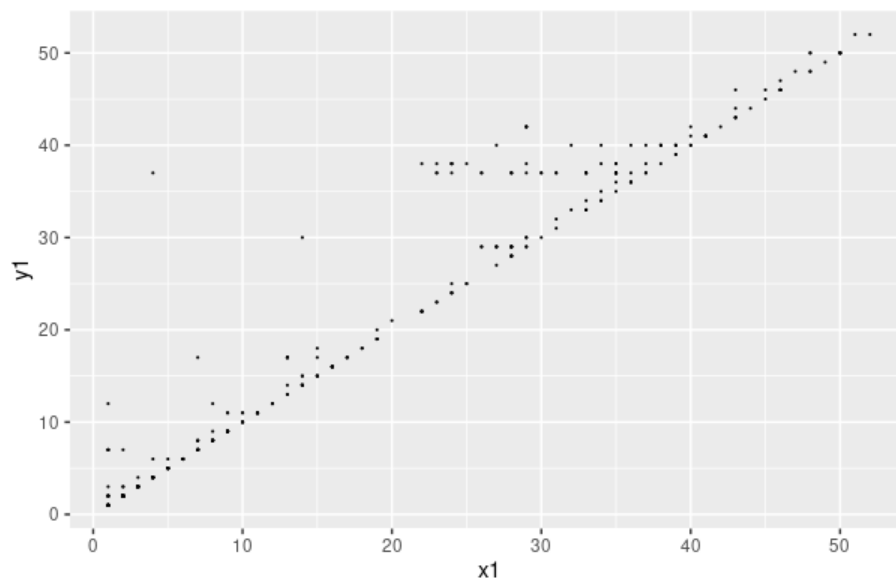


Figure A.62: Gobblin

- Observation period: Approximately 50 months
- Total number of bugs: 569
- Percentage of languages (%): Java98.5, shell0.7, python0.3, JavaScript0.3, CSS0.1, HTML0.1
- URL for the software overview: <https://gobblin.apache.org/>

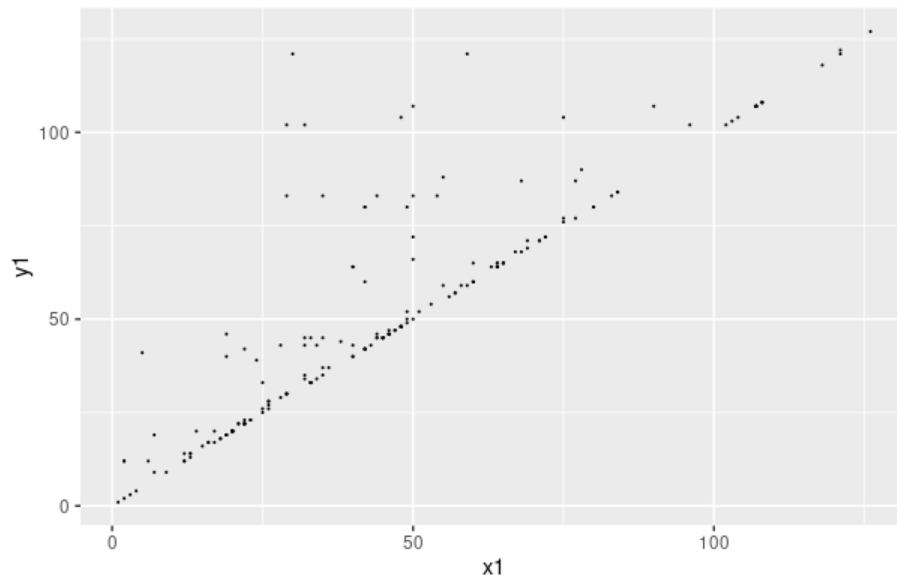


Figure A.63: Gora

- Observation period: Approximately 125 months
- Total number of bugs: 207
- Percentage of languages (%): Java99.3, other0.7
- URL for the software overview: <https://gora.apache.org/>

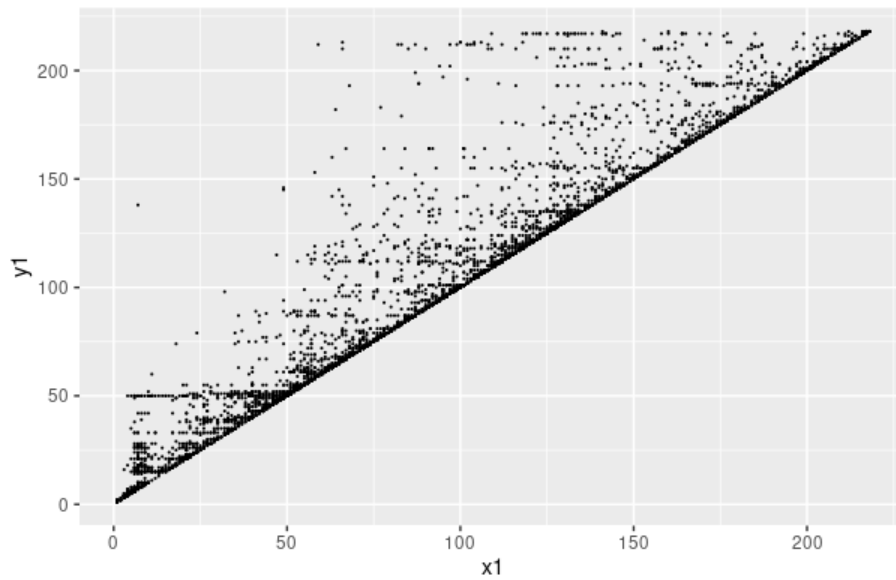


Figure A.64: Groovy

- Observation period: Approximately 200 months
- Total number of bugs: 6728
- Percentage of languages (%): Java55.3, Groovy43.6, HTML0.3, CSS0.3, shell0.2, other0.3
- URL for the software overview: <https://Groovy.apache.org/>

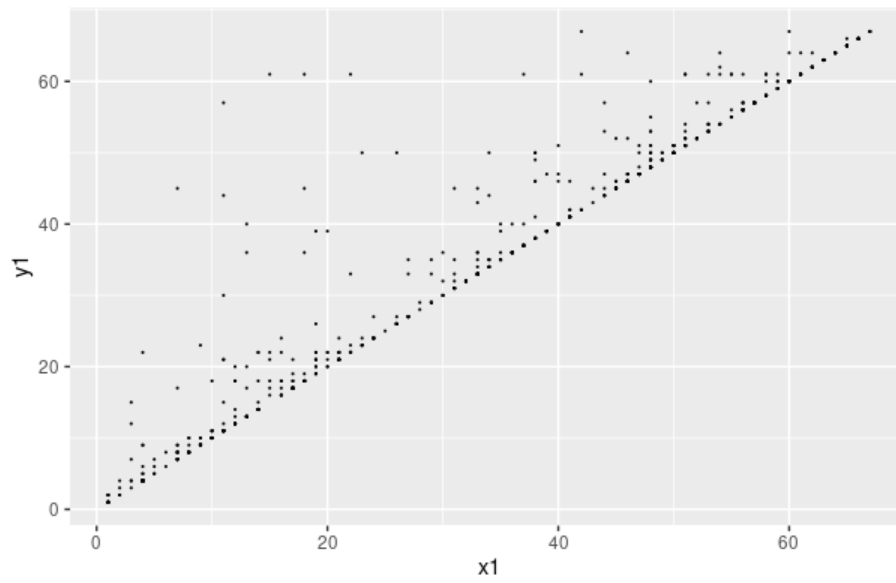


Figure A.65: Guacamole

- Observation period: Approximately 60 months
- Total number of bugs: 772
- Percentage of languages (%): Java60.7, JavaScript32.7, CSS2.8, HTML1.7, shell0.8, other1.3
- URL for the software overview: <https://guacamole.apache.org/>

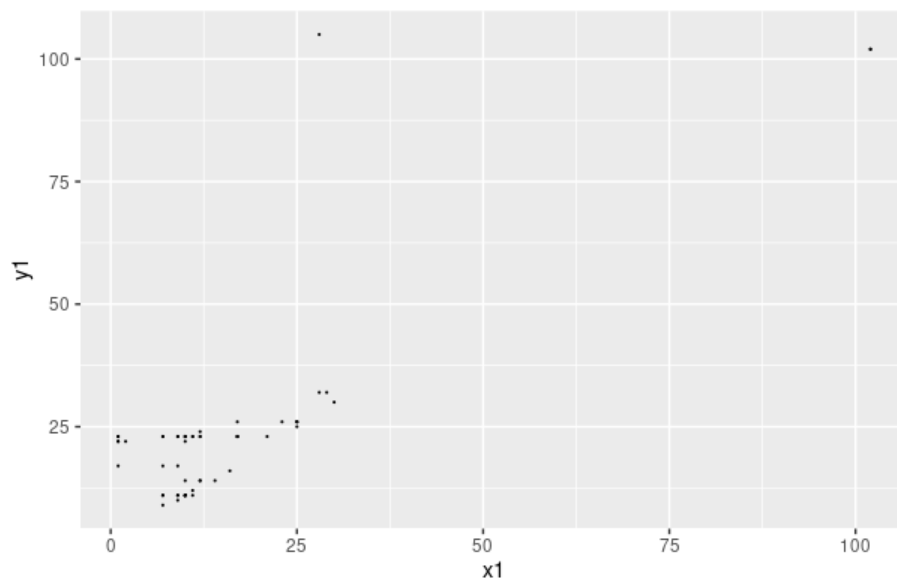


Figure A.66: Gump

- Observation period: Approximately 100 months
- Total number of bugs: 65
- Percentage of languages (%): python100
- URL for the software overview: <https://gump.apache.org/>

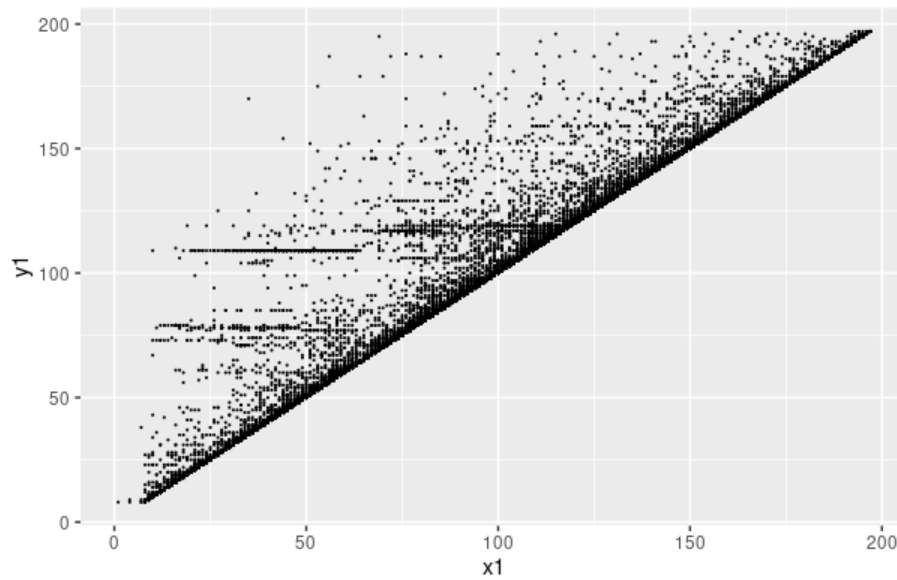


Figure A.67: Hadoop

- Observation period: Approximately 200 months
- Total number of bugs: 23533
- Percentage of languages (%): Java92.9, C++2.8, C1.9, JavaScript1.2, shell0.5
- URL for the software overview: <https://hadoop.apache.org/>

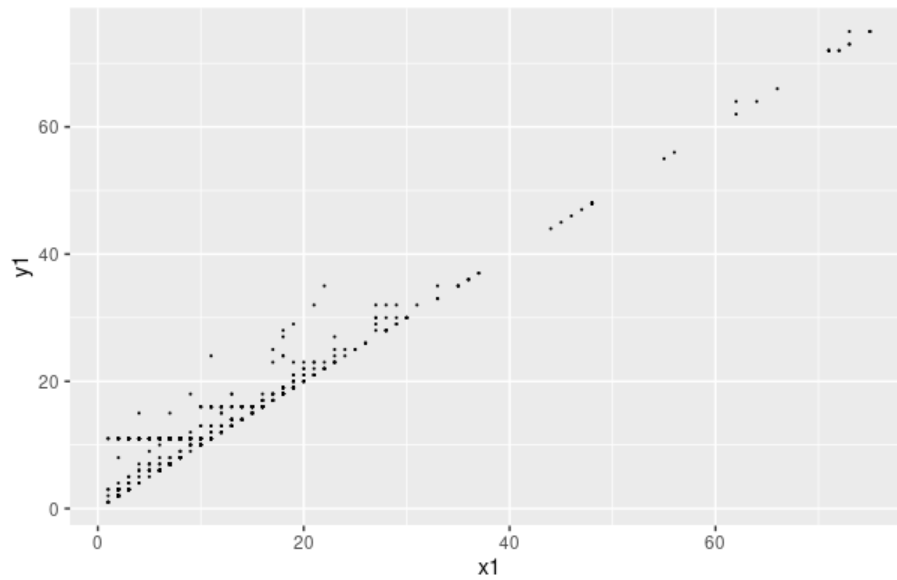


Figure A.68: Hawq

- Observation period: Approximately 80 months
- Total number of bugs: 890
- Percentage of languages (%): C70.1, python9.0, C++7.2, Java5.0, perl1.8, other6.6
- URL for the software overview: <https://hawq.apache.org/>

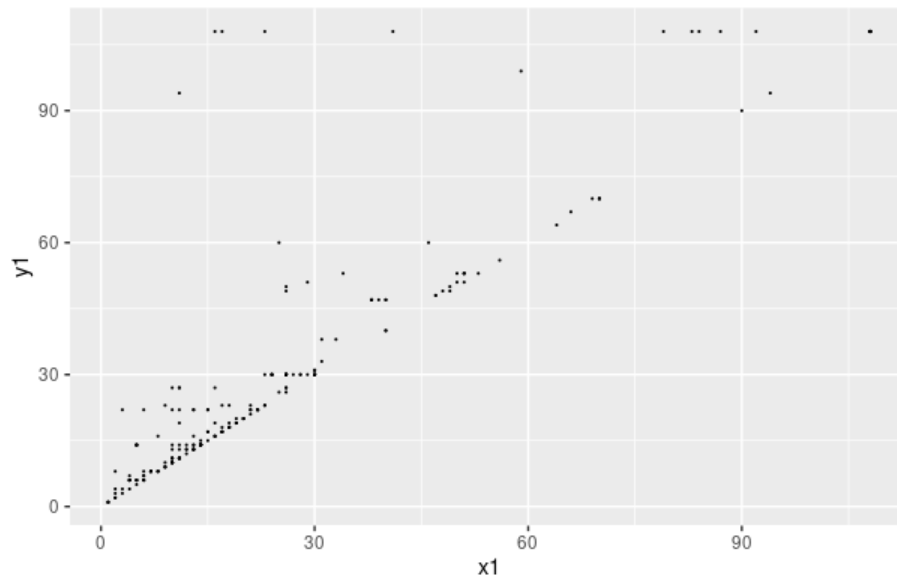


Figure A.69: Helix

- Observation period: Approximately 100 months
- Total number of bugs: 332
- Percentage of languages (%): Java94.7, python1.7, TypeScript1.5, shell1.3, HTML0.7, other0.1
- URL for the software overview: <https://helix.apache.org/>

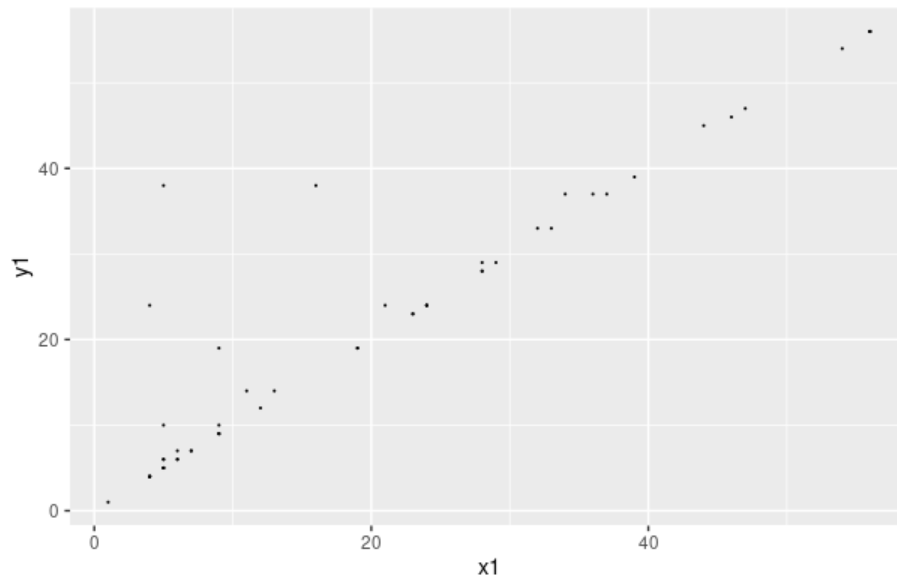


Figure A.70: Hivemall

- Observation period: Approximately 60 months
- Total number of bugs: 58
- Percentage of languages (%): Java98.6, other1.4
- URL for the software overview: <https://hive.apache.org/>

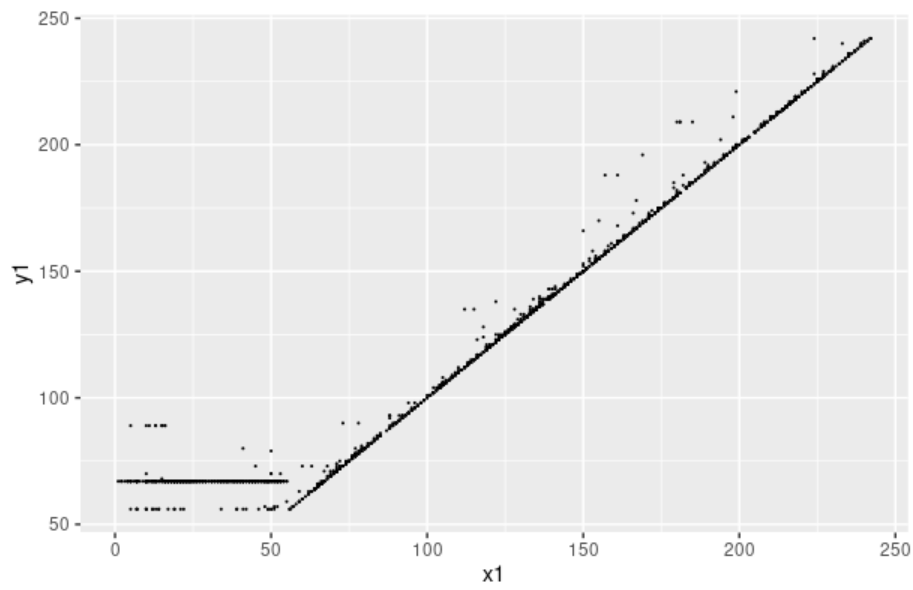


Figure A.71: HTTPComponents

- Observation period: Approximately 250 months
- Total number of bugs: 1882
- Percentage of languages (%): Java99.9, other0.1
- URL for the software overview: <https://hc.apache.org/>

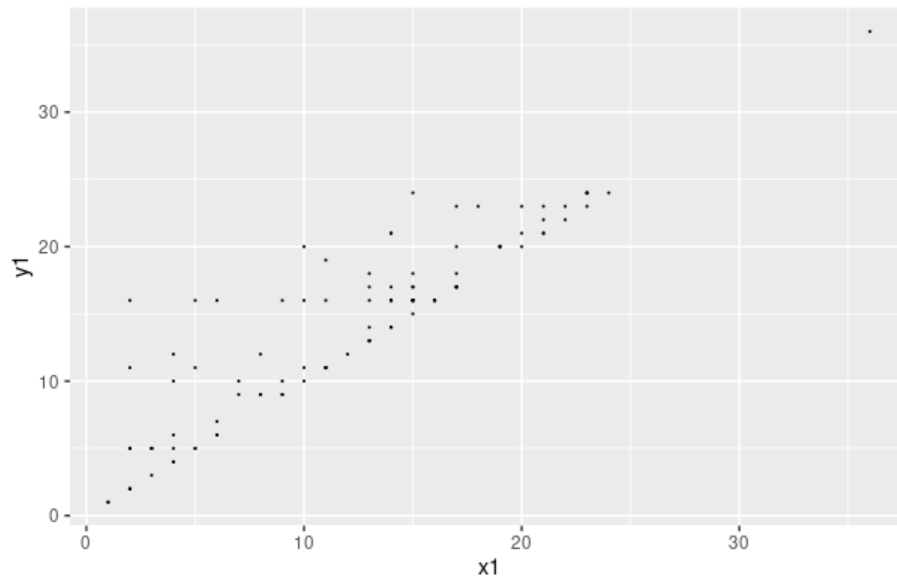


Figure A.72: HTTPServer

- Observation period: Approximately 40 months
- Total number of bugs: 146
- Percentage of languages (%): C91.8, python4.2, shell0.9 makefile0.3, other2.4
- URL for the software overview: <https://httpd.apache.org/>

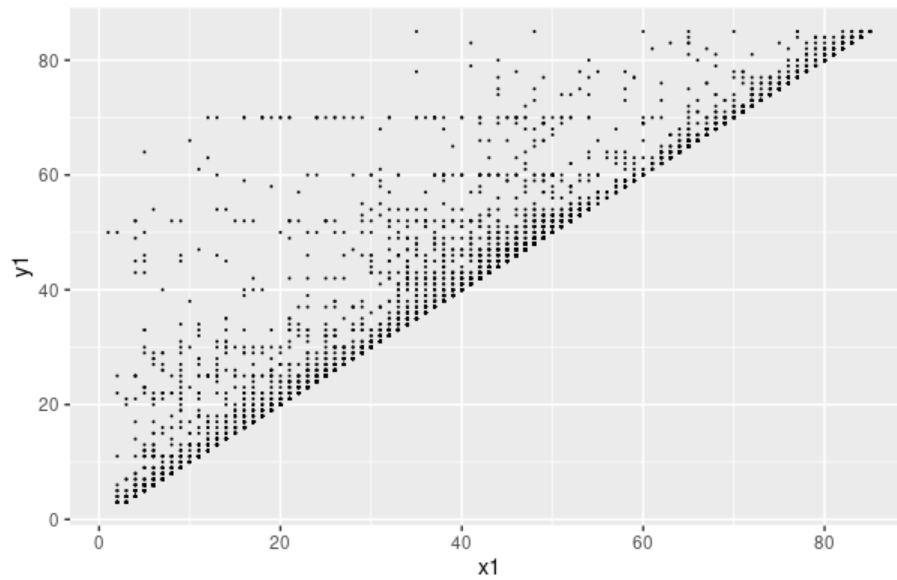


Figure A.73: Ignite

- Observation period: Approximately 80 months
- Total number of bugs: 6777
- Percentage of languages (%): Java75.7, C#12.9, C++7.1, scala2.4, shell1.1, python0.5, other0.3
- URL for the software overview: <https://ignite.apache.org/>

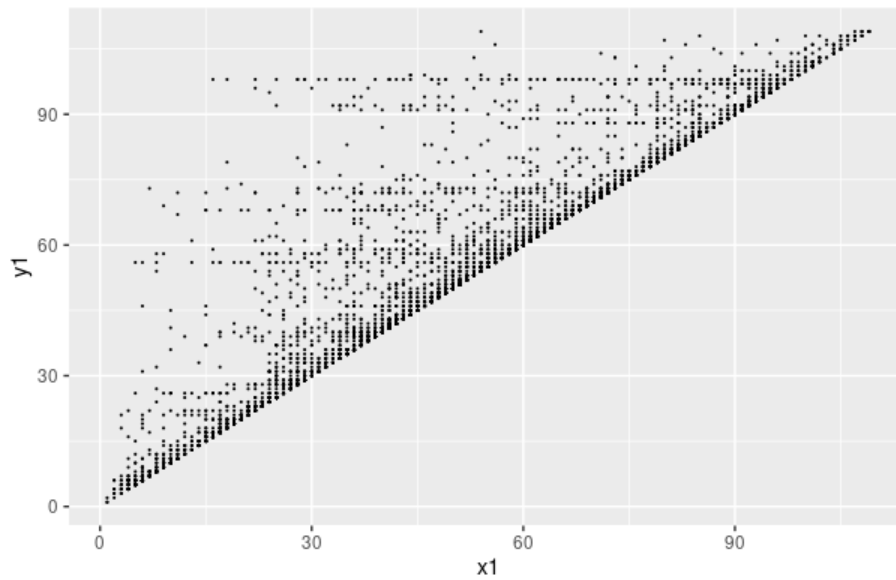


Figure A.74: Impala

- Observation period: Approximately 100 months
- Total number of bugs: 6011
- Percentage of languages (%): C++52.7,Java25.5,python14.9,C2.3,JavaScript1.4,thrift1.1,other2.1
- URL for the software overview: <https://impala.apache.org/>

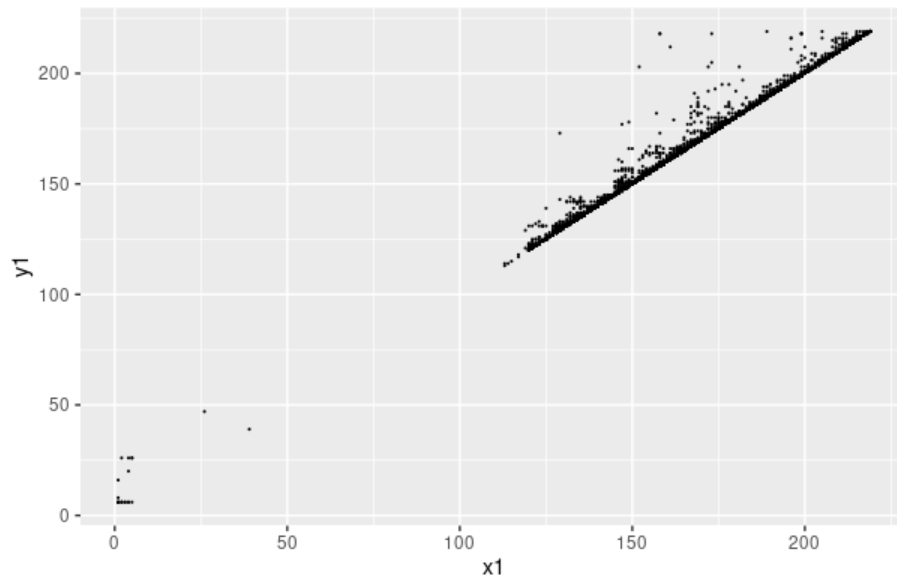


Figure A.75: Infrastructure

- Observation period: Approximately 200 months
- Total number of bugs: 4972
- Percentage of languages (%): Java95.7, HTML3.7, shell0.6
- URL for the software overview: <https://github.com/apache/infrastructure-website>

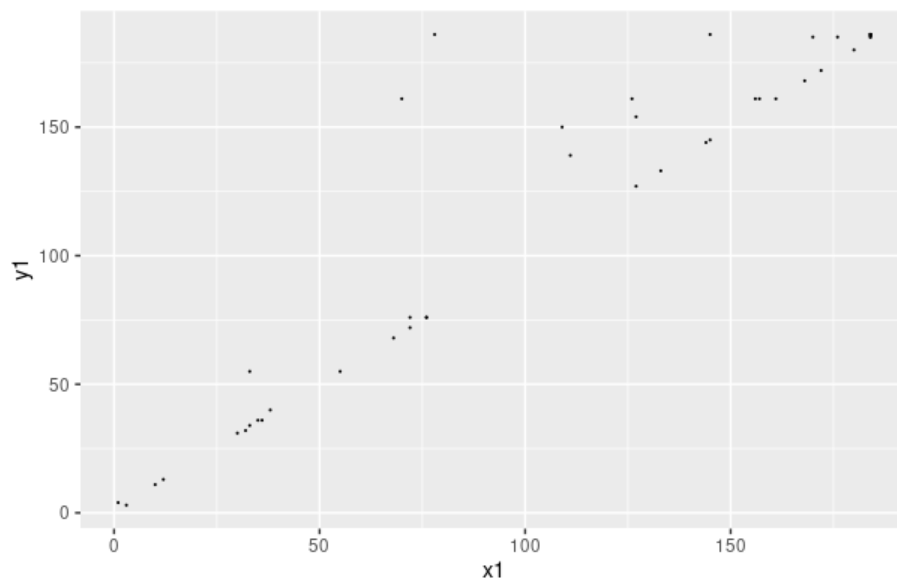
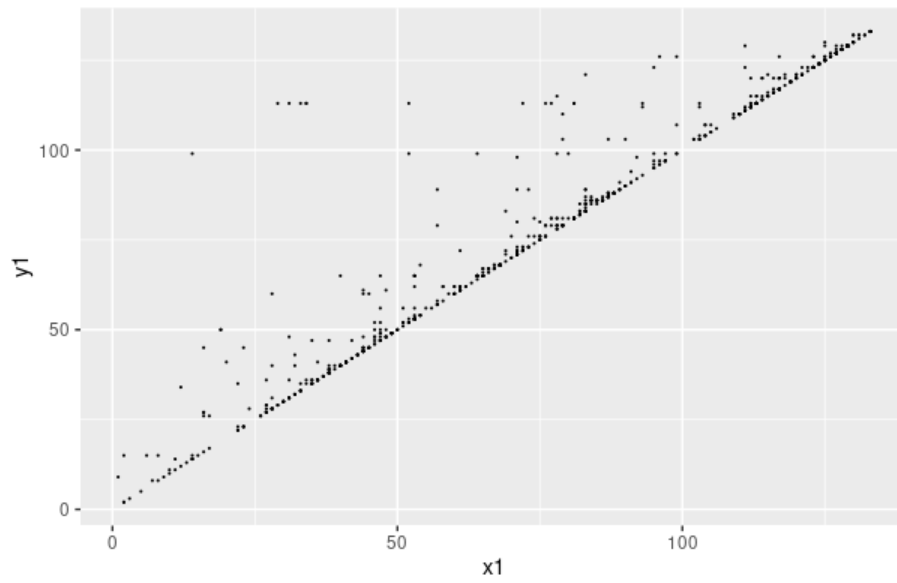


Figure A.76: Incubator

- Observation period: Approximately 175 months
- Total number of bugs: 44
- Percentage of languages (%): CSS50.8, Groovy29.1, JavaScript12.8, shell6.2
- URL for the software overview: <https://incubator.apache.org/>

**Figure A.77**

- Observation period: Approximately 125 months
- Total number of bugs: 799
- Percentage of languages (%): Java72.8,richtextformat10.1,JavaScript6.5,HTML1.3,CSS0.9,other8.4
- URL for the software overview: <https://isis.apache.org/>

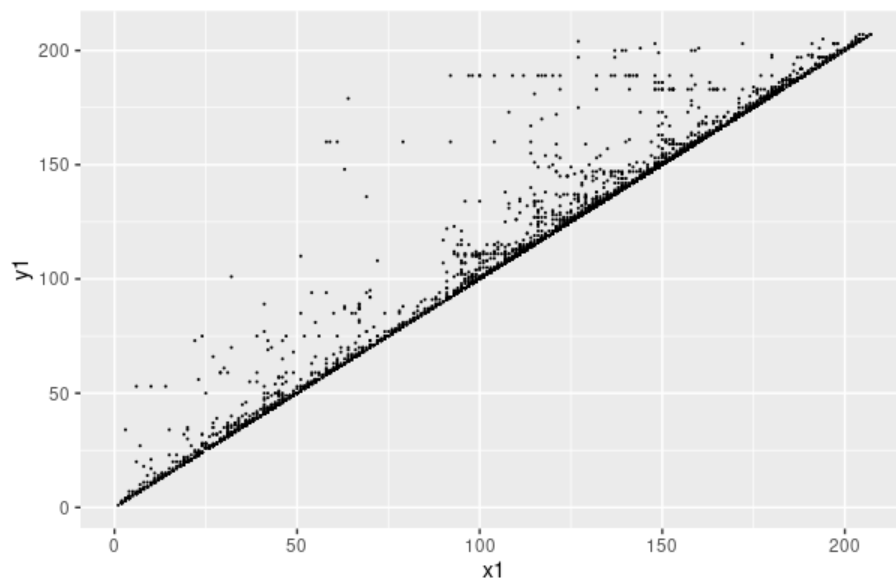


Figure A.78: Jackrabbit

- Observation period: Approximately 200 months
- Total number of bugs: 6240
- Percentage of languages (%): Java99.7, other0.3
- URL for the software overview: <https://jackrabbit.apache.org/jcr/index.HTML>

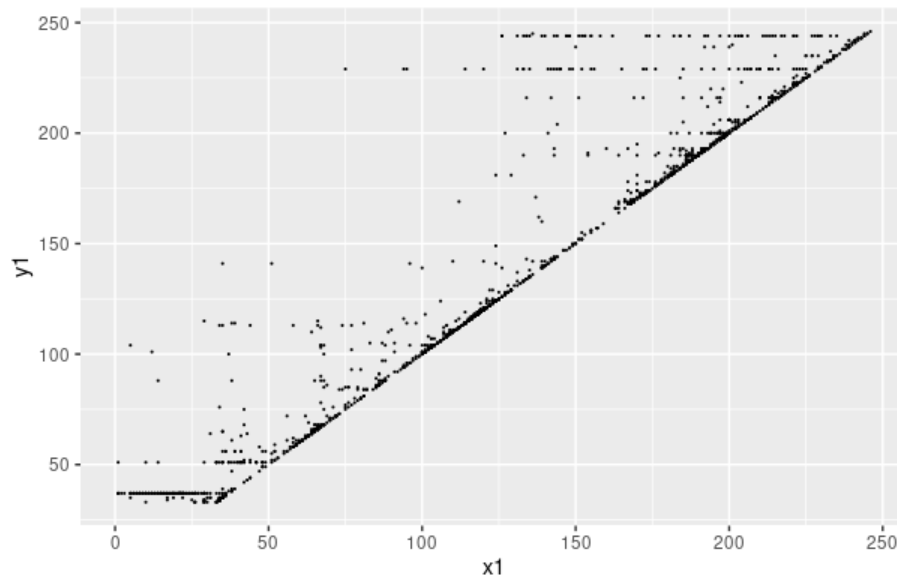


Figure A.79: James

- Observation period: Approximately 250 months
- Total number of bugs: 1968
- Percentage of languages (%): Java88.5, scala9.8, HTML0.8, other0.5, CSS0.3,JavaScript0.1
- URL for the software overview: <https://james.apache.org/>

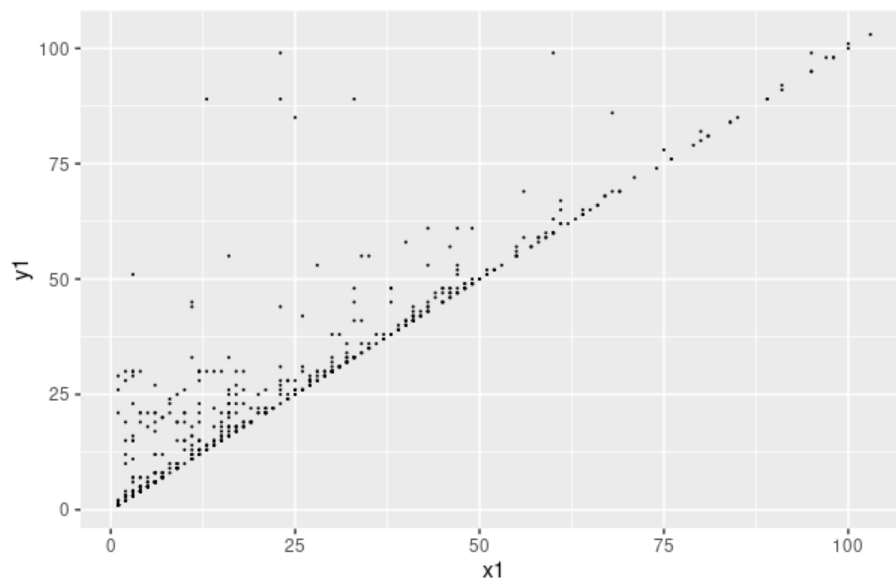
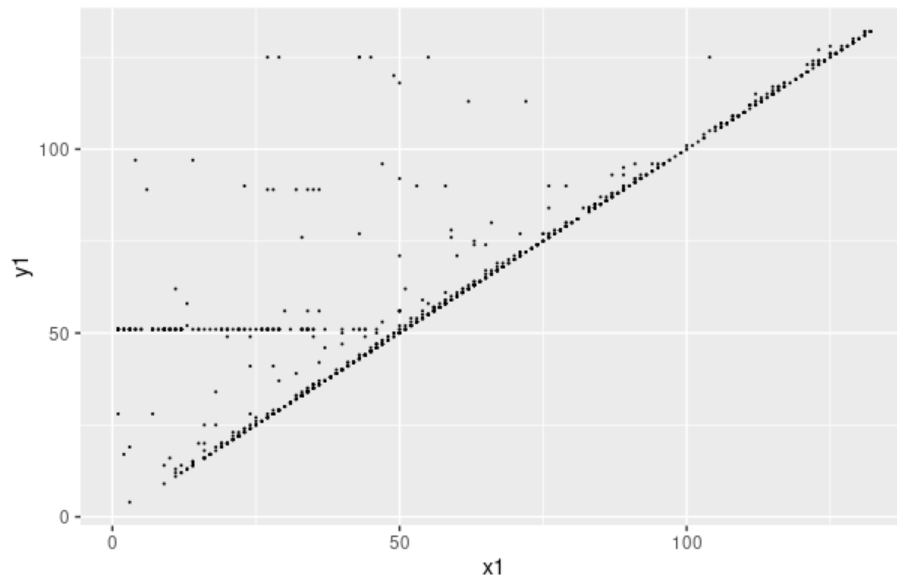


Figure A.80: jclouds

- Observation period: Approximately 100 months
- Total number of bugs: 818
- Percentage of languages (%): Java99.4, other0.6
- URL for the software overview: <https://jclouds.apache.org/>

**Figure A.81:** Jena

- Observation period: Approximately 150 months
- Total number of bugs: 1074
- Percentage of languages (%): Java95.3, JavaScript2.5, shell0.7, Ruby0.5, HTML0.3, other0.7
- URL for the software overview: <https://jena.apache.org/>

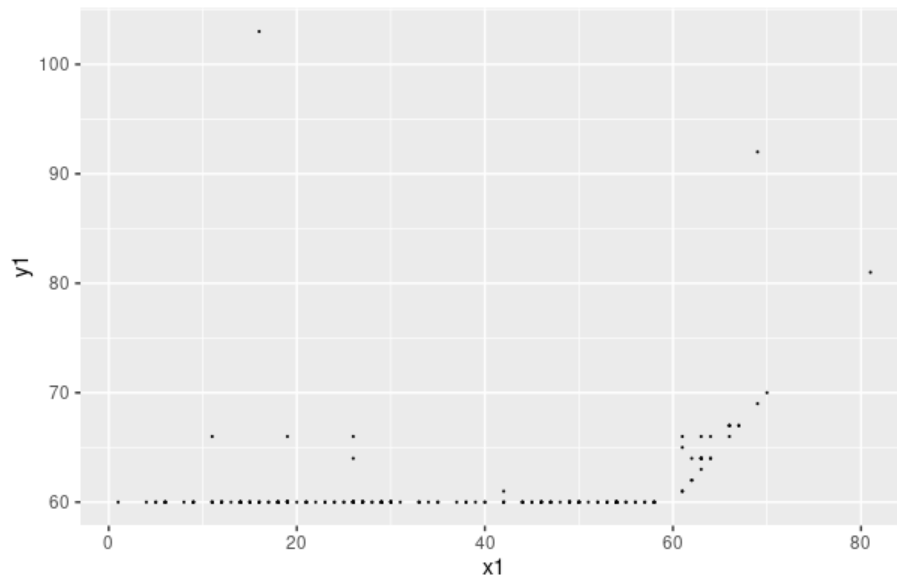


Figure A.82: Joshua

- Observation period: Approximately 100 months
- Total number of bugs: 274
- Percentage of languages (%): Java79.5, Perl7.1, other4.7, python3.1, roff2.3
- URL for the software overview: <https://github.com/apache/joshua>

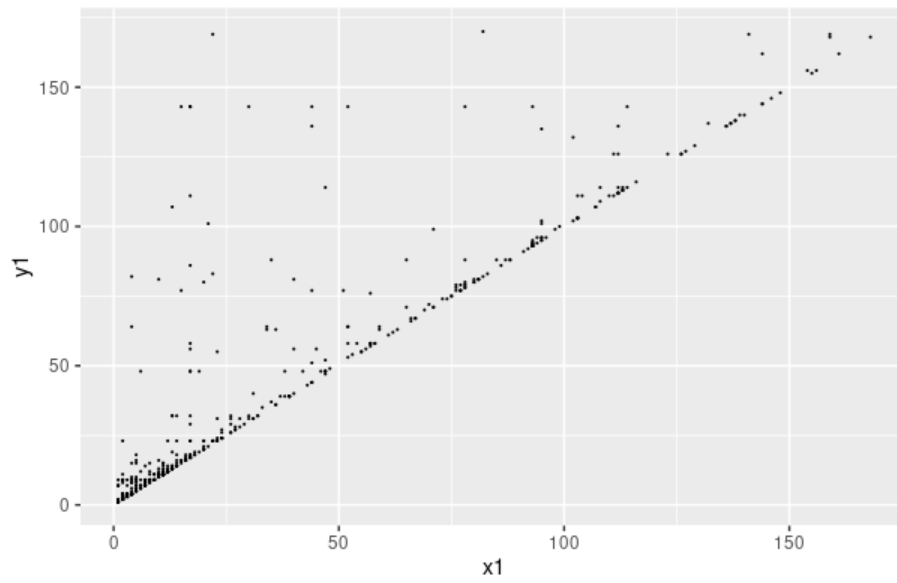


Figure A.83: JSPWIKI

- Observation period: Approximately 175 months
- Total number of bugs: 577
- Percentage of languages (%): Java79.4, JavaScript10.4, HTML4.4, other3.1, CSS2.6, dockerfile0.1
- URL for the software overview: <https://jspwiki.apache.org/>

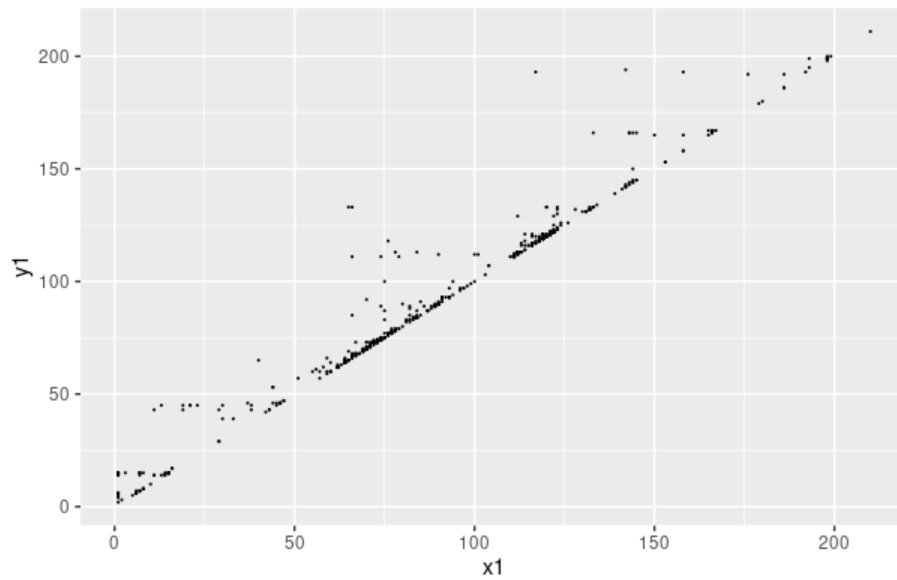


Figure A.84: JUDDI

- Observation period: Approximately 200 months
- Total number of bugs: 582
- Percentage of languages (%): Java80.3, C#16.8, JavaScript1.9, HTML0.4, batchfile0.3, shell0.2, CSS0.1
- URL for the software overview: <https://juddi.apache.org/>

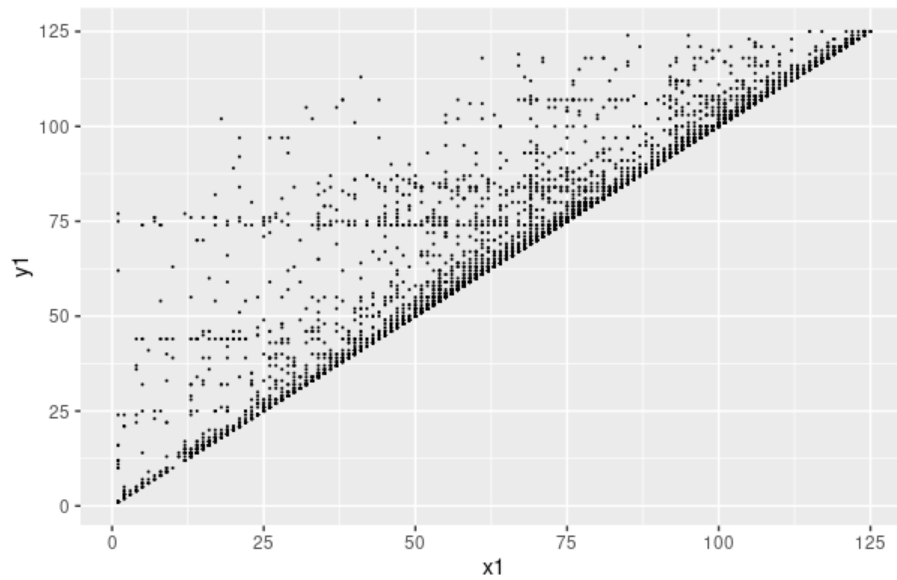


Figure A.85: Kafka

- Observation period: Approximately 125 months
- Total number of bugs: 6503
- Percentage of languages (%): Java73.6, scala23.1, python2.8, shell0.3, roff0.1, batchfile 0.1
- URL for the software overview: <https://kafka.apache.org/>

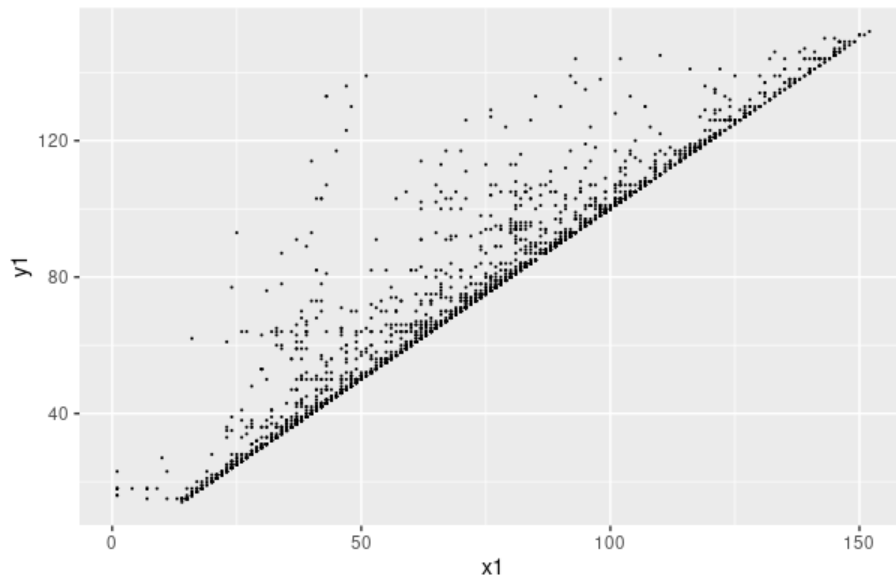


Figure A.86: Karaf

- Observation period: Approximately 150 months
- Total number of bugs: 3123
- Percentage of languages (%): Java97.1, shell1.3, batchfile0.7, JavaScript0.4, XSLT0.3, CSS0.1, other0.1
- URL for the software overview: <https://karaf.apache.org/>

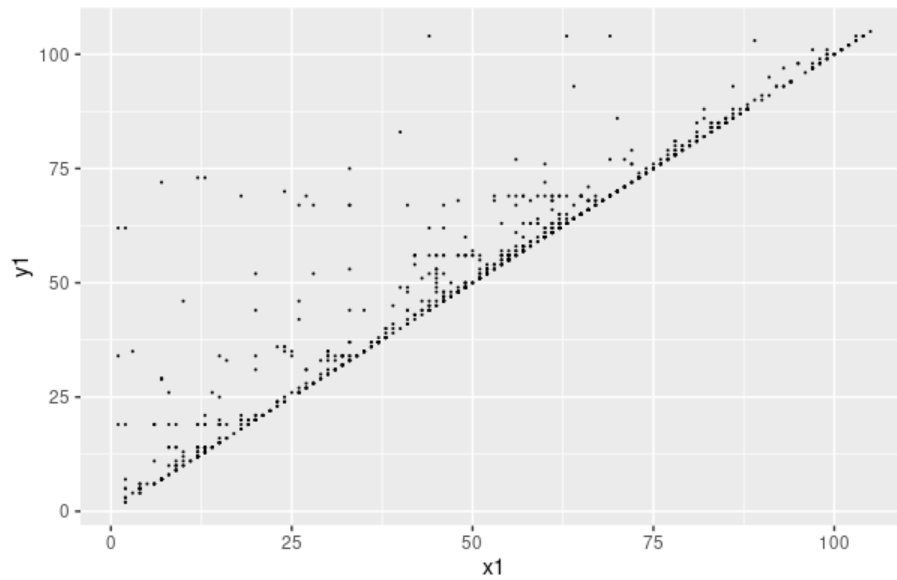


Figure A.87: Knox

- Observation period: Approximately 100 months
- Total number of bugs: 1183
- Percentage of languages (%): Java92.1, TypeScript3.7, HTML1.1, CSS1.1, shell1.0, Groovy0.8, other0.2
- URL for the software overview: <https://knox.apache.org/>

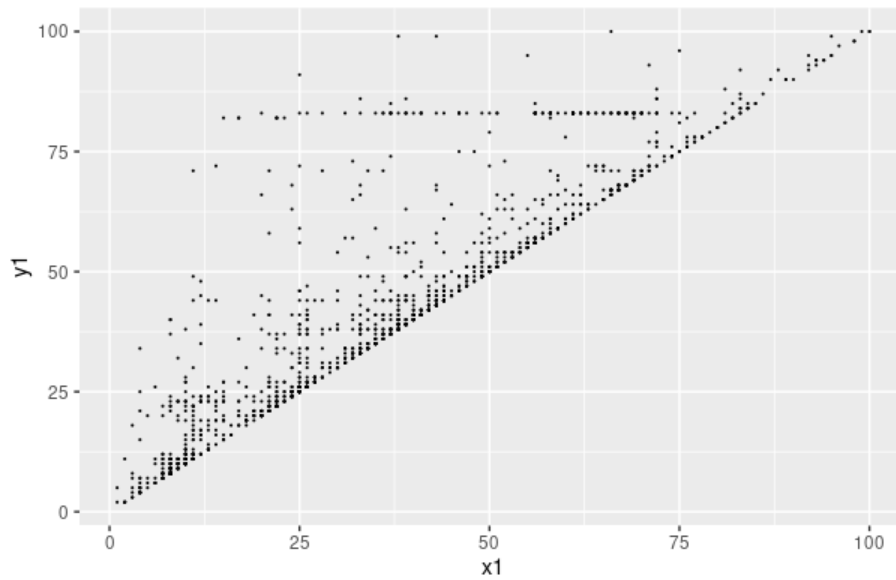


Figure A.88: Kudu

- Observation period: Approximately 100 months
- Total number of bugs: 1715
- Percentage of languages (%): C++80.1, Java10.4, C2.6, python1.9, scala1.5, other3.5
- URL for the software overview: <https://kudu.apache.org/>

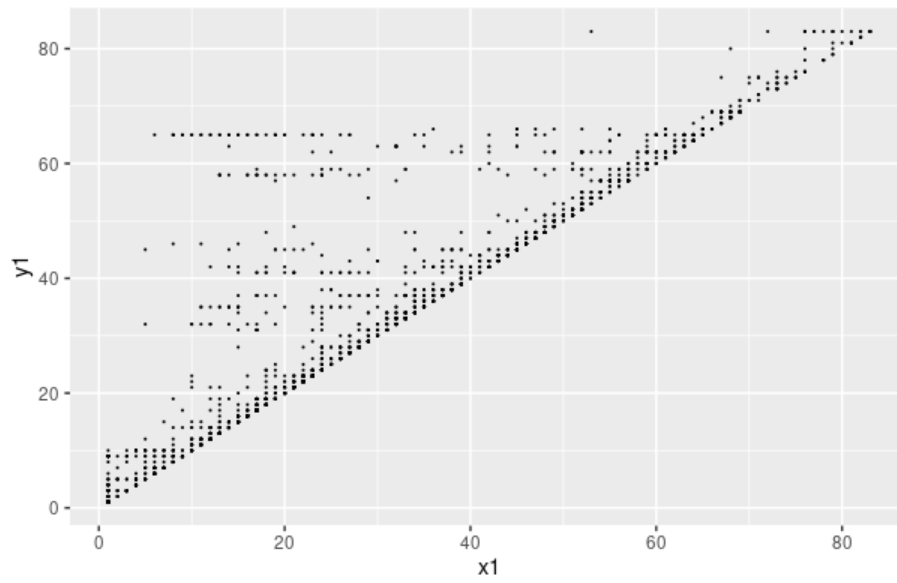


Figure A.89: Kylin

- Observation period: Approximately 80 months
- Total number of bugs: 2499
- Percentage of languages (%): Java68.8, JavaScript10.4, other6.9, scala4.2, C++3.4, HTML2.9, other3.4
- URL for the software overview: <https://kylin.apache.org/>

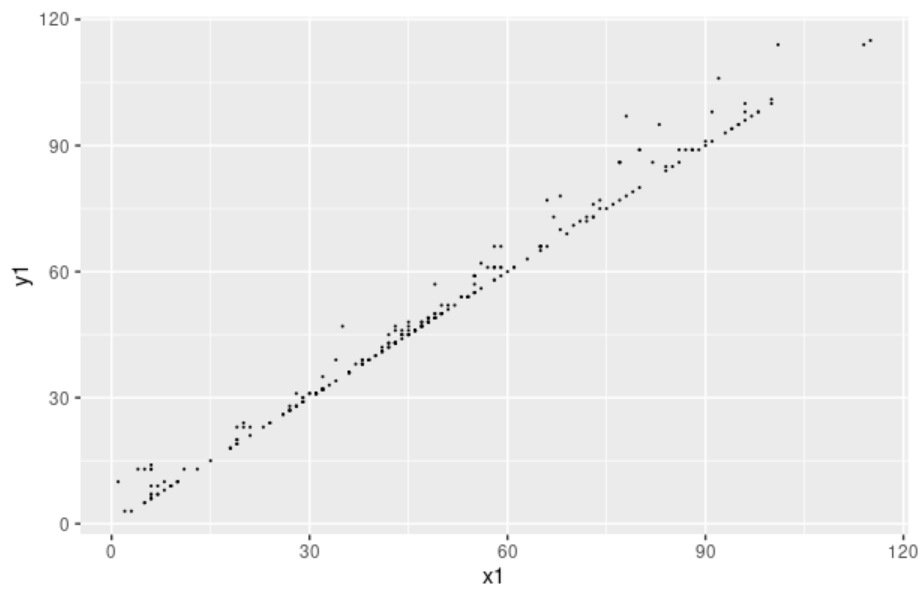


Figure A.90: Libcloud

- Observation period: Approximately 120 months
- Total number of bugs: 474
- Percentage of languages (%): python99.9, other0.1
- URL for the software overview: <https://libcloud.apache.org/>

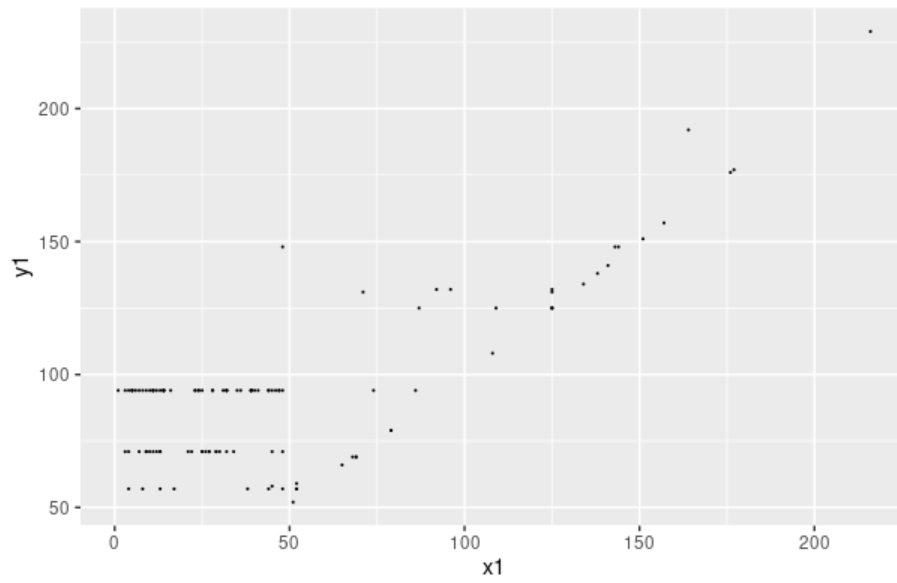


Figure A.91: Logging

- Observation period: Approximately 200 months
- Total number of bugs: 116
- Percentage of languages (%): Java79.7, shell20.3
- URL for the software overview: <https://logging.apache.org/>

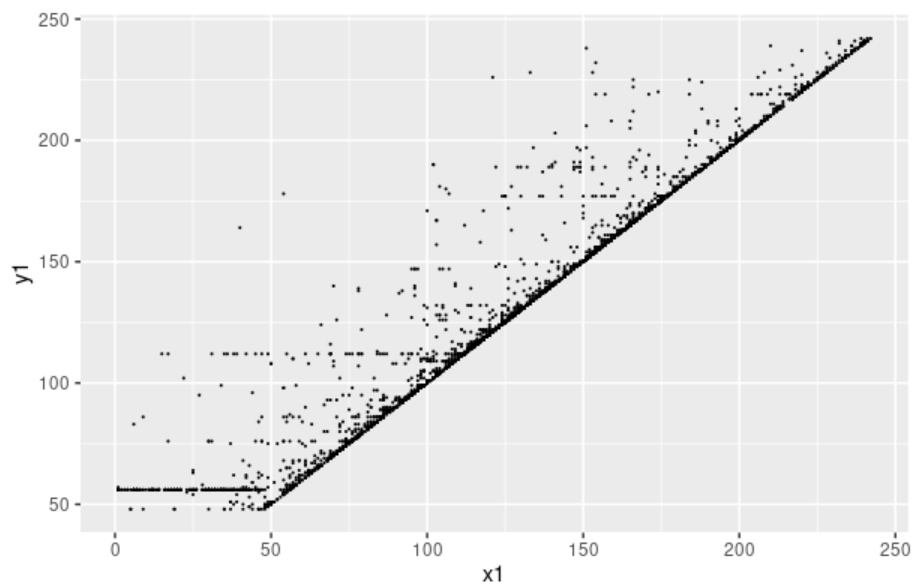


Figure A.92: Lucene

- Observation period: Approximately 250 months
- Total number of bugs: 4519
- Percentage of languages (%): Java97.6, HTML1.1, python0.8, other0.3, Perl0.1, shell0.1
- URL for the software overview: <https://lucene.apache.org/>

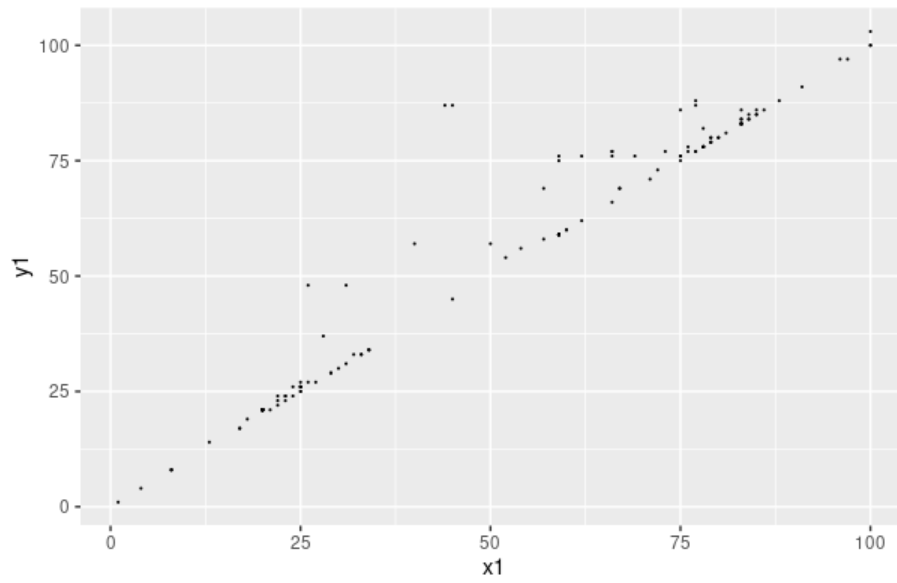


Figure A.93: Lucy

- Observation period: Approximately 100 months
- Total number of bugs: 154
- Percentage of languages (%): C79.7, Perl14.6, GO4.7, shell0.2, Java0.2, other0.6
- URL for the software overview: <https://github.com/apache/lucy>

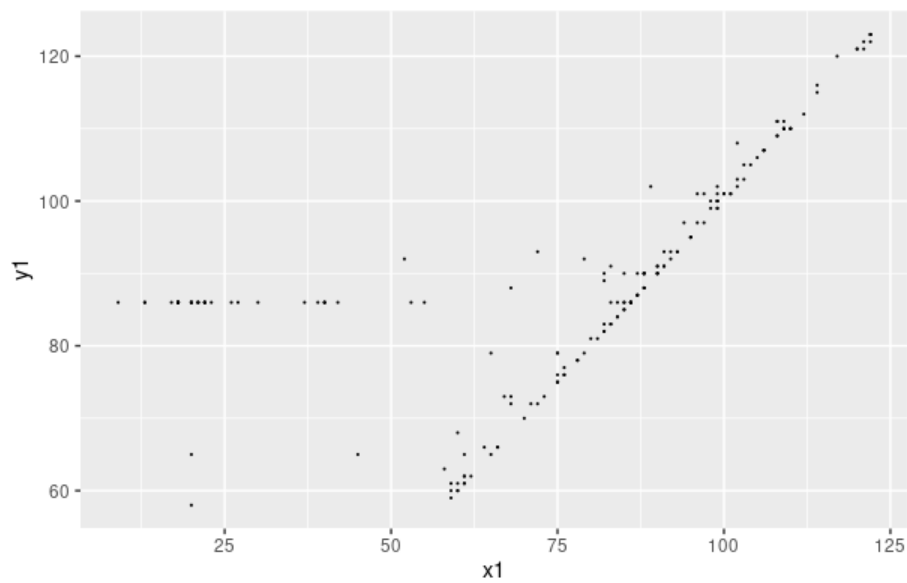


Figure A.94: MADlib

- Observation period: Approximately 125 months
- Total number of bugs: 656
- Percentage of languages (%): C++41.0, C38.9, python13.2, shell1.0, other5.9
- URL for the software overview: <https://madlib.apache.org/>

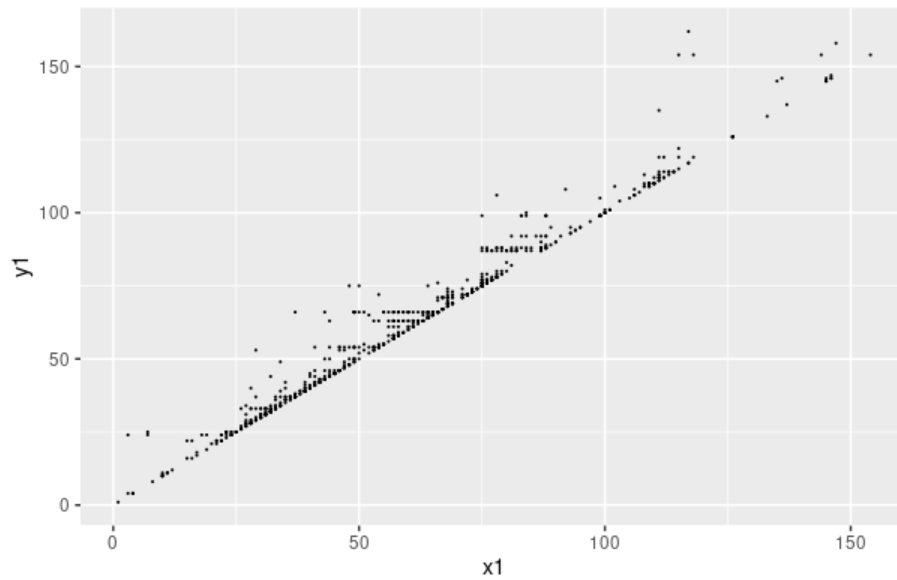


Figure A.95: Mahout

- Observation period: Approximately 150 months
- Total number of bugs: 826
- Percentage of languages (%): Java80.5, scala14.1, Perl1.9, other1.4, shell1.1, HTML0.3, other0.7
- URL for the software overview: <https://mahout.apache.org/>

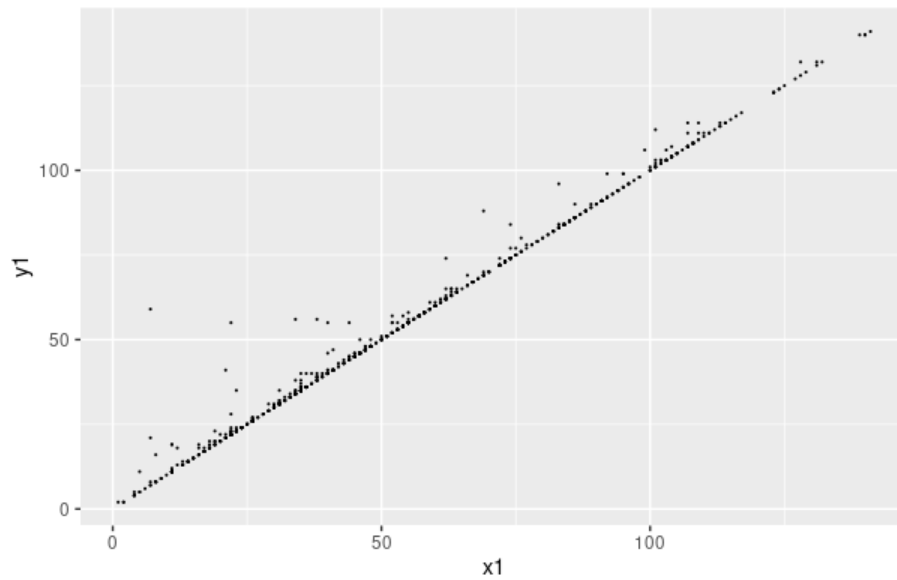


Figure A.96: ManifoldCF

- Observation period: Approximately 150 months
- Total number of bugs: 868
- Percentage of languages (%): Java91.2, HTML4.6, JavaScript1.7, python1.1, C0.3, other1.1
- URL for the software overview: <https://manifoldcf.apache.org/>

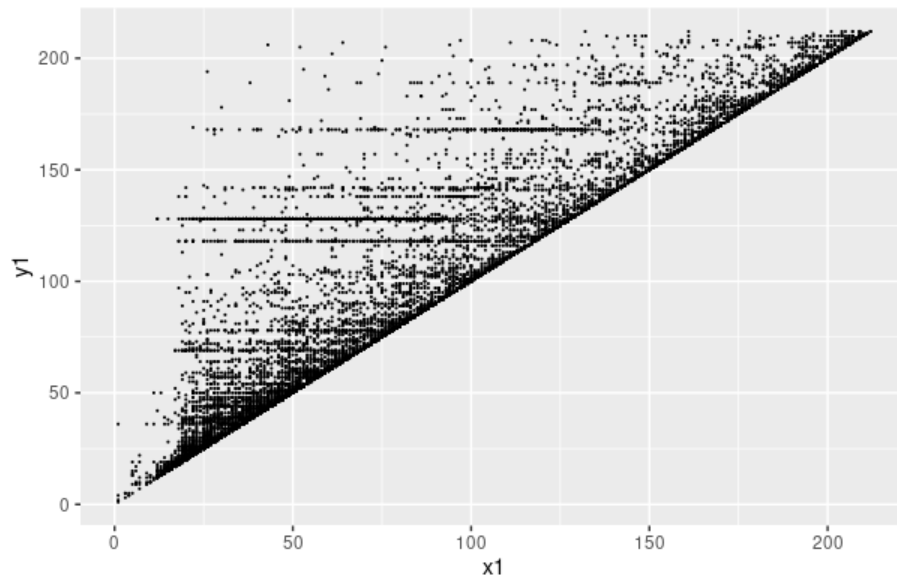


Figure A.97: Maven

- Observation period: Approximately 200 months
- Total number of bugs: 13917
- Percentage of languages (%): Java99.6, other0.4
- URL for the software overview: <https://maven.apache.org/>

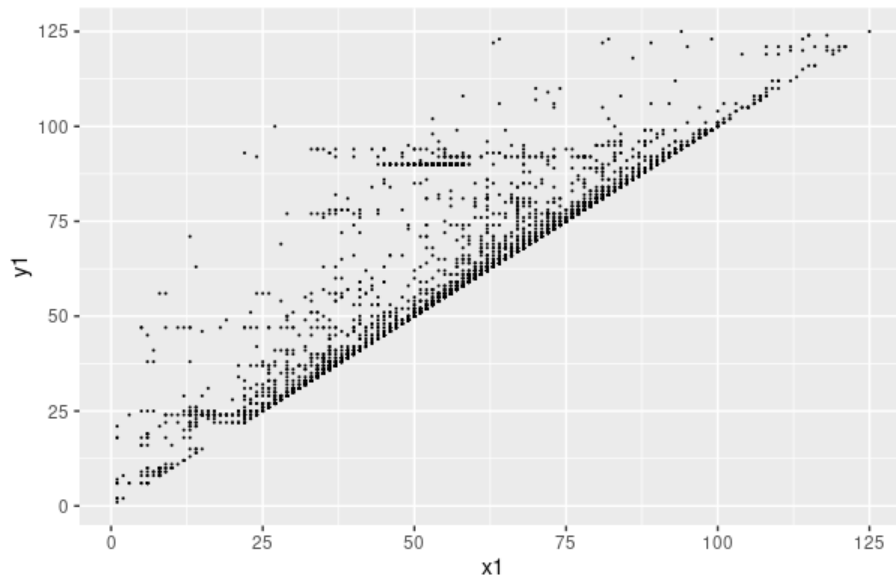


Figure A.98: Mesos

- Observation period: Approximately 125 months
- Total number of bugs: 4917
- Percentage of languages (%): C++92.1, python2.1, Java0.9, shell0.9, make-file0.7, other3.3
- URL for the software overview: <https://mesos.apache.org/>

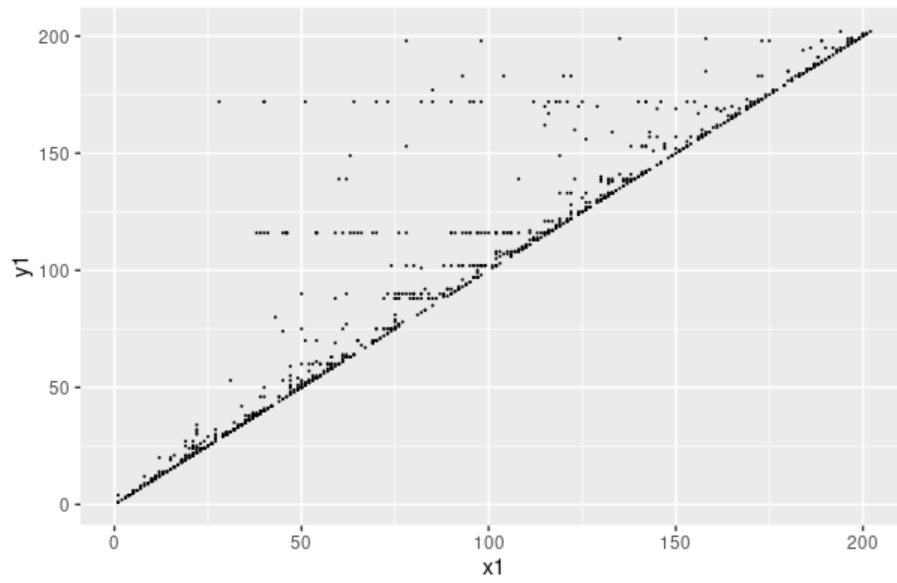


Figure A.99: MINA

- Observation period: Approximately 200 months
- Total number of bugs: 1132
- Percentage of languages (%): Java99.4, other0.6
- URL for the software overview: <https://mina.apache.org/>

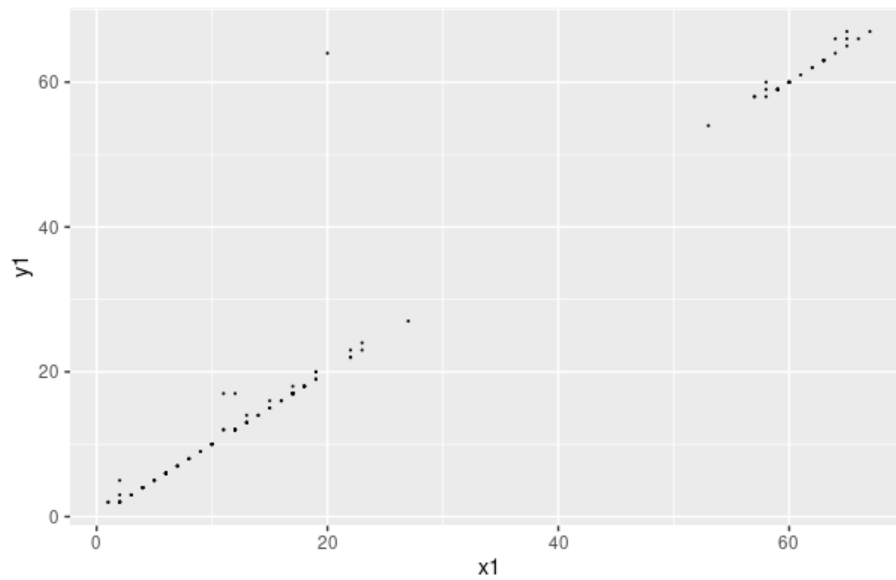


Figure A.100: Mnemonic

- Observation period: Approximately 60 months
- Total number of bugs: 132
- Percentage of languages (%): Java75.7, C16.0, scala4.4, other1.4, shell1.2, python0.9, dockerfile0.4
- URL for the software overview: <https://mnemonic.apache.org/>

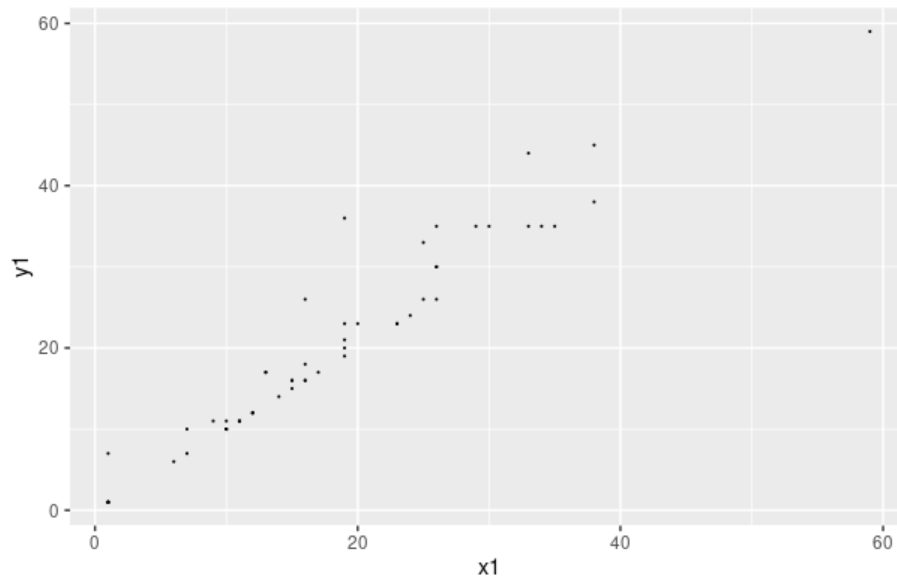


Figure A.101: MRUnit

- Observation period: Approximately 60 months
- Total number of bugs: 72
- Percentage of languages (%): Java92.9, HTML6.5, shell0.6
- URL for the software overview: <https://github.com/apache/mrunit>

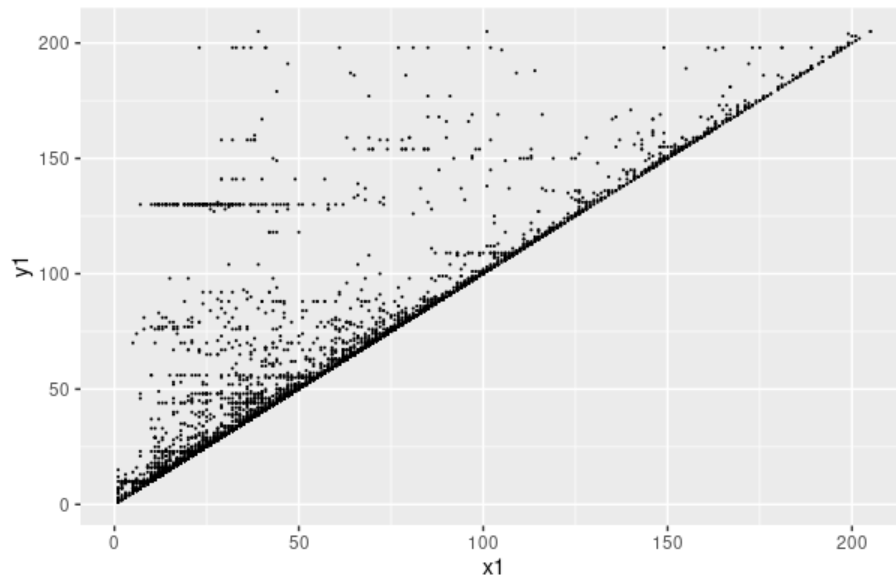


Figure A.102: MyFaces

- Observation period: Approximately 200 months
- Total number of bugs: 6729
- Percentage of languages (%): Java90.6, HTML6.2, JavaScript3.2
- URL for the software overview: <https://myfaces.apache.org/>

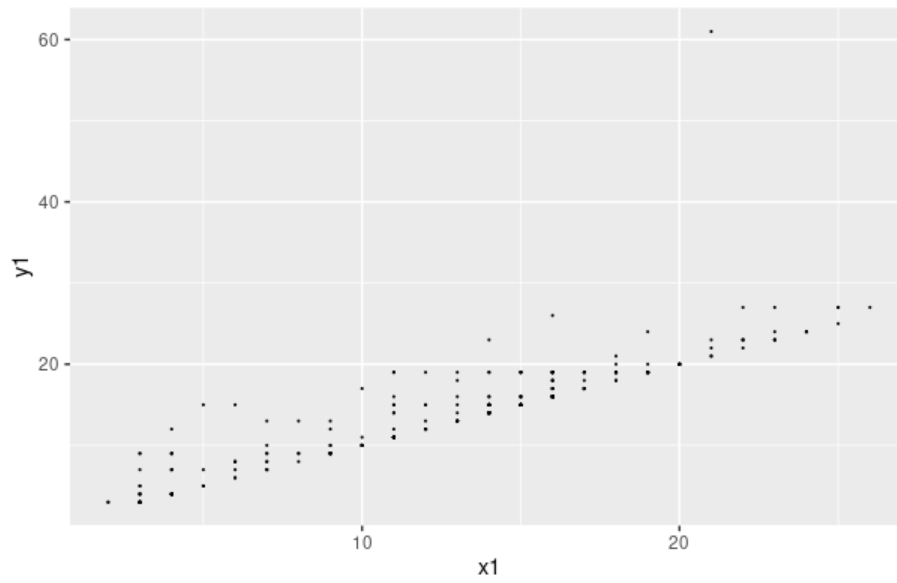


Figure A.103: Mynewt

- Observation period: Approximately 60 months
- Total number of bugs: 357
- Percentage of languages (%): C99.2, other0.8
- URL for the software overview: <https://mynewt.apache.org/>

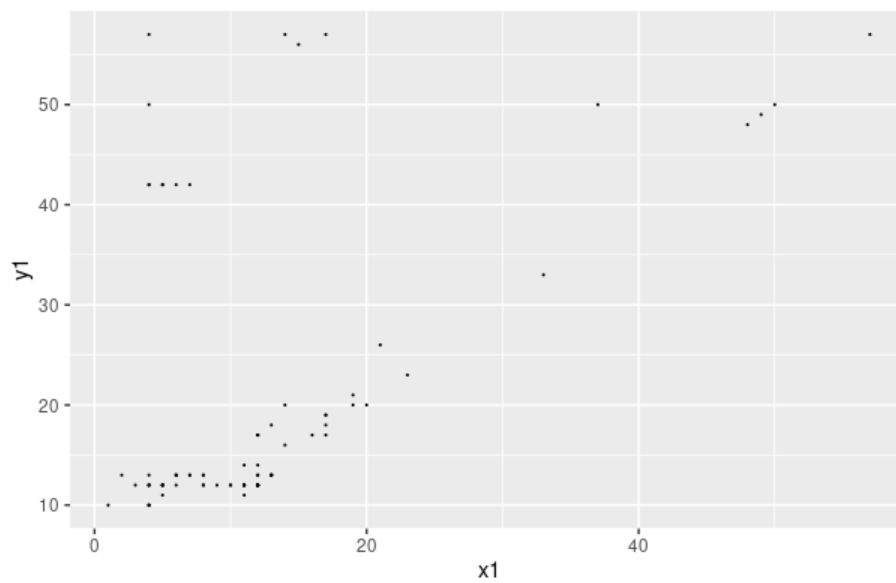


Figure A.104: Myriad

- Observation period: Approximately 60 months
- Total number of bugs: 160
- Percentage of languages (%): Java47.9, CSS30.3, JavaScript14.7, shell3.0, HTML2.1, python1.0, other1.0
- URL for the software overview: <https://github.com/apache/incubator-myriad>

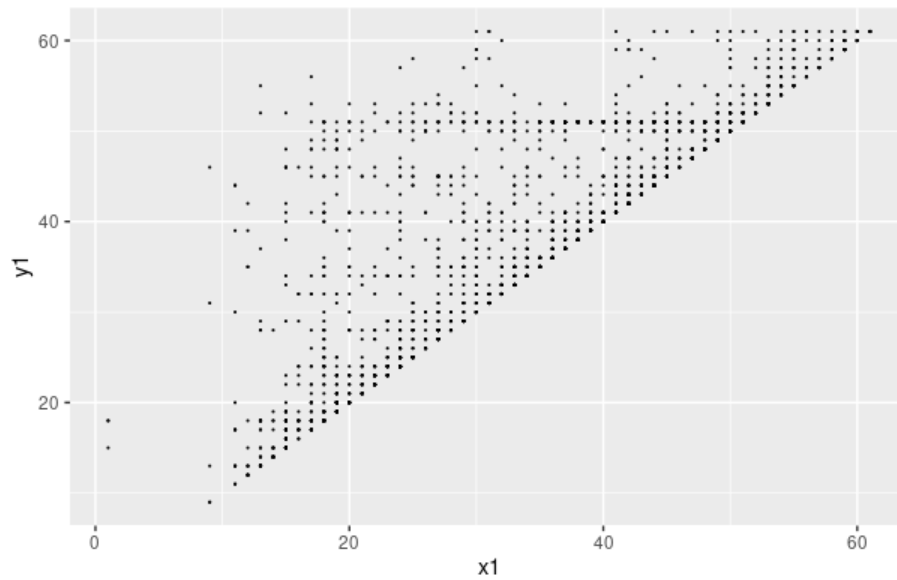


Figure A.105: NetBeans

- Observation period: Approximately 60 months
- Total number of bugs: 4547
- Percentage of languages (%): Java86.8, HTML1.9, php1.1 XSLT0.4, JavaScript0.4, other1.3
- URL for the software overview: <https://netbeans.apache.org/>

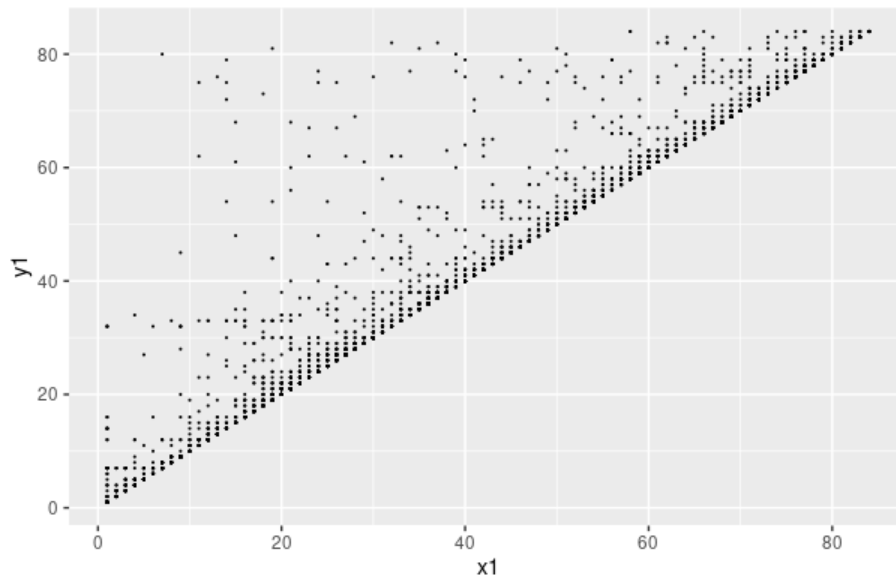


Figure A.106: NiFi

- Observation period: Approximately 80 months
- Total number of bugs: 4871
- Percentage of languages (%): Java86.7, JavaScript6.5, Groovy4.0, HTML1.7, CSS0.5, shell0.3, other0.3
- URL for the software overview: <https://nifi.apache.org/>

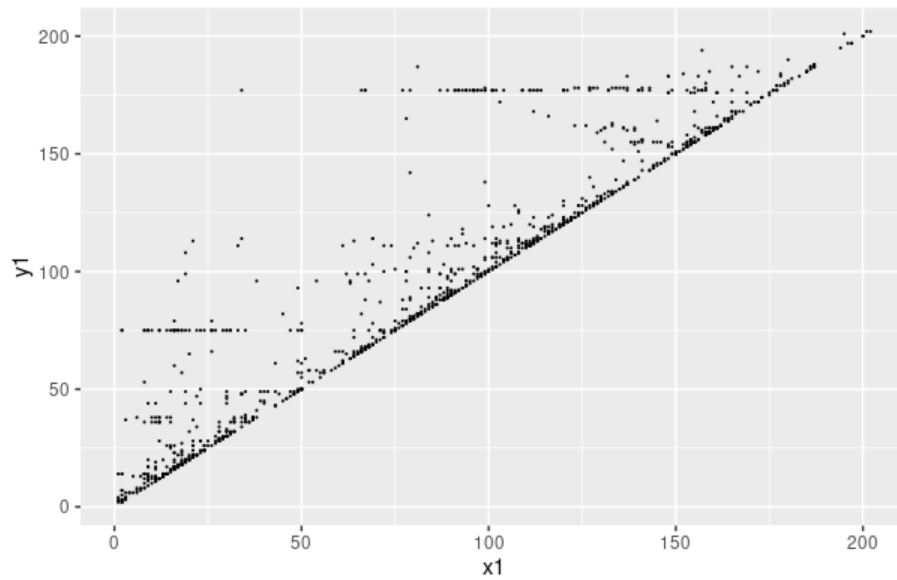


Figure A.107: Nutch

- Observation period: Approximately 200 months
- Total number of bugs: 1282
- Percentage of languages (%): Java93.6, HTML5.3, shell0.9, XSLT0.1, rich-textformat 0.1
- URL for the software overview: <https://nutch.apache.org/>

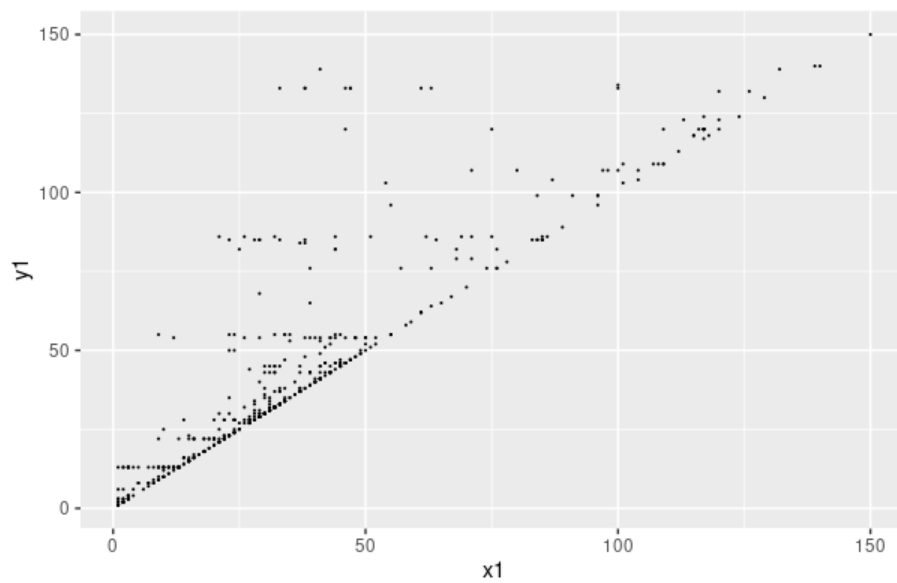


Figure A.108: ODE

- Observation period: Approximately 150 months
- Total number of bugs: 655
- Percentage of languages (%): Java92.1,JavaScript4.1,Ruby1.8,HTML0.7,CSS0.6,shell0.2,other0.5
- URL for the software overview: <https://github.com/apache/ode>

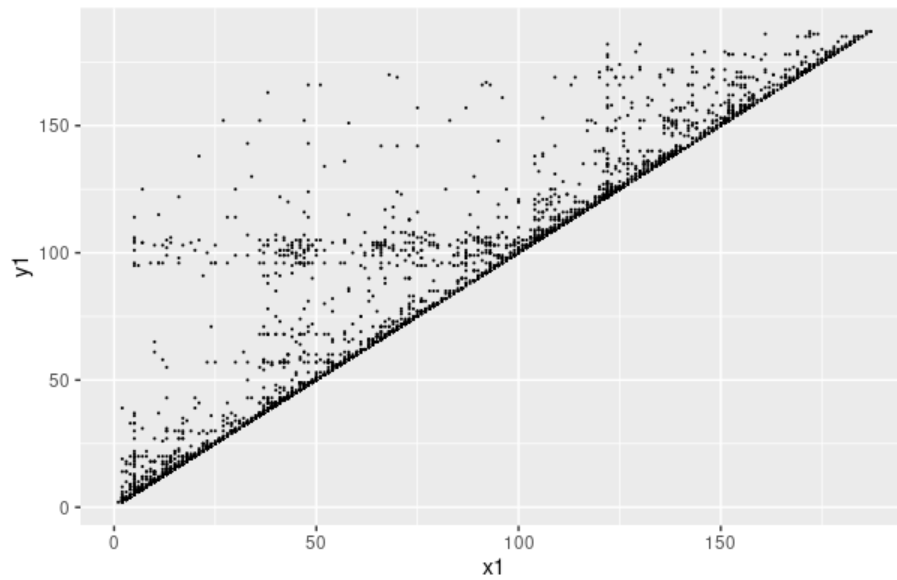


Figure A.109: OFBiz

- Observation period: Approximately 200 months
- Total number of bugs: 4663
- Percentage of languages (%): Java46.0, HTML26.6, freemarker10.8, JavaScript8.5, Groovy4.0, CSS3.9, other0.2
- URL for the software overview: <https://ofbiz.apache.org/>

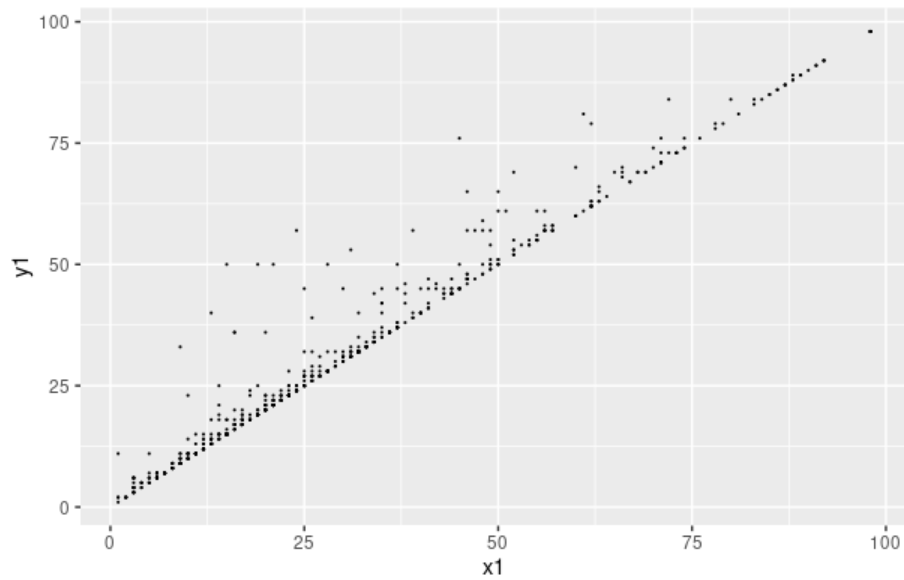


Figure A.110: Olingo

- Observation period: Approximately 100 months
- Total number of bugs: 831
- Percentage of languages (%): Java99.9, HTML0.1
- URL for the software overview: <https://olingo.apache.org/>

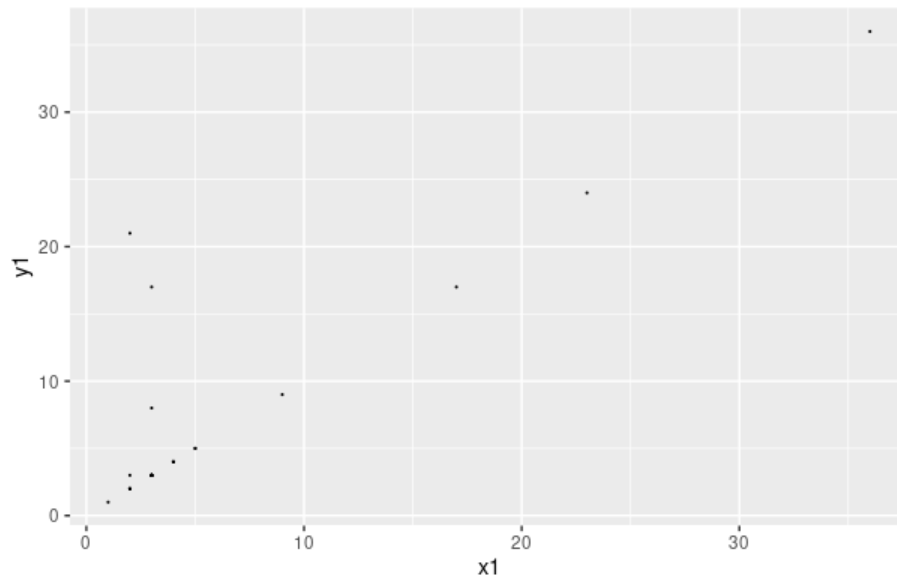


Figure A.111: Onami

- Observation period: Approximately 40 months
- Total number of bugs: 32
- Percentage of languages (%): Java99.7, shell0.3
- URL for the software overview: <https://github.com/apache/onami>

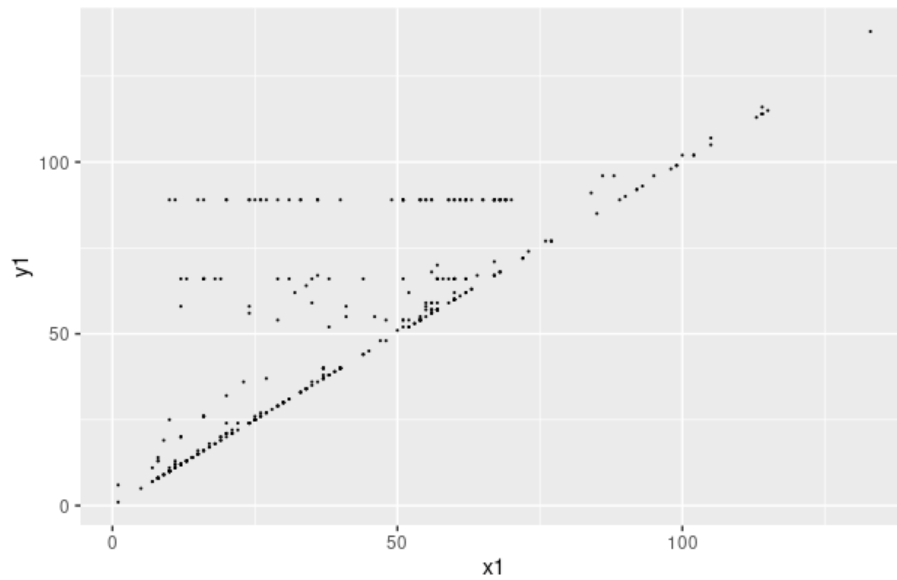


Figure A.112: OODT

- Observation period: Approximately 125 months
- Total number of bugs: 392
- Percentage of languages (%): Java89.0, python1.9, HTML1.8, shell1.6, CSS1.6, roff1.4, other2.7
- URL for the software overview: <https://oodt.apache.org/>

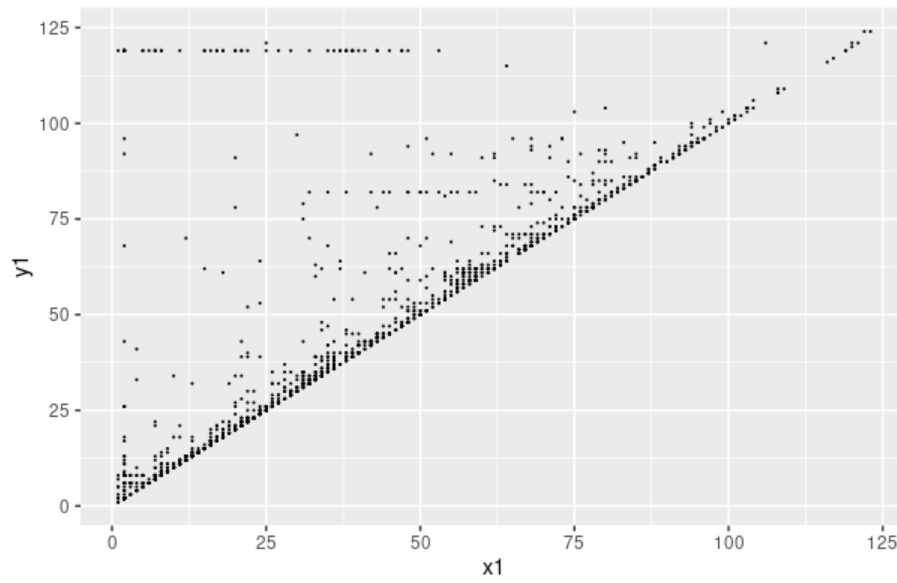


Figure A.113: Oozie

- Observation period: Approximately 125 months
- Total number of bugs: 2232
- Percentage of languages (%): Java97.5, JavaScript1.2, other1.3
- URL for the software overview: <https://oozie.apache.org/>

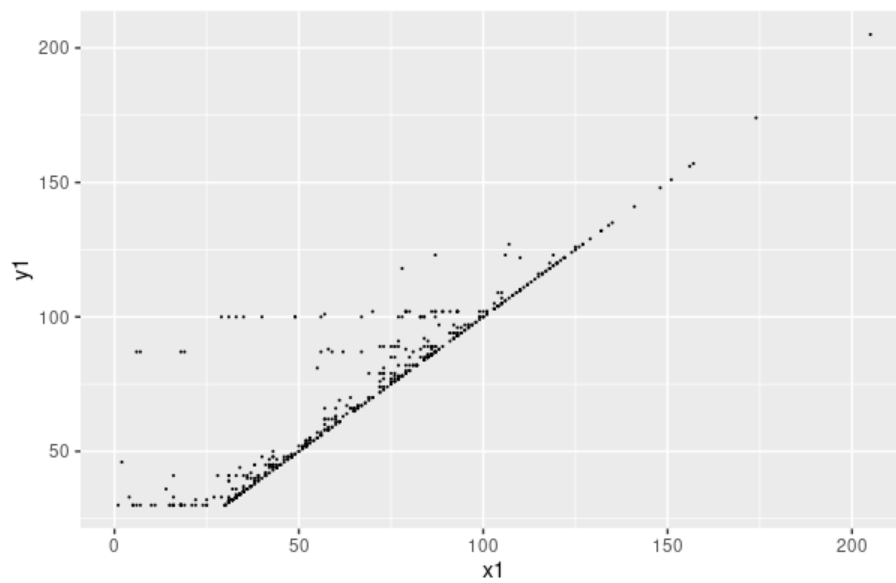


Figure A.114: OpenEJB

- Observation period: Approximately 200 months
- Total number of bugs: 710
- Percentage of languages (%): Java97.8, HTML1.9, other0.3
- URL for the software overview: <https://openejb.apache.org/>

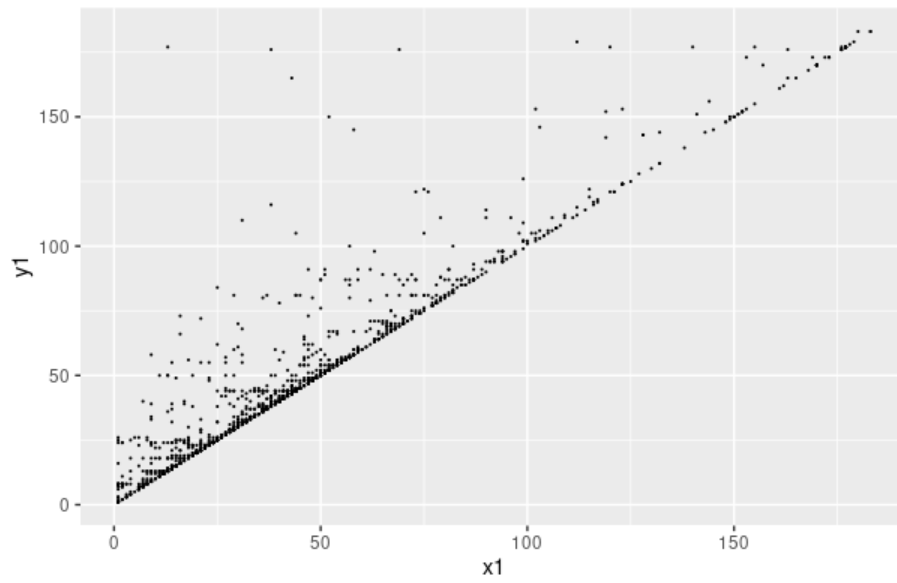


Figure A.115: OpenJPA

- Observation period: Approximately 175 months
- Total number of bugs: 1862
- Percentage of languages (%): Java99.2, HTML0.4, JavaScript0.2, CSS0.1, XSLT0.1
- URL for the software overview: <https://openjpa.apache.org/>

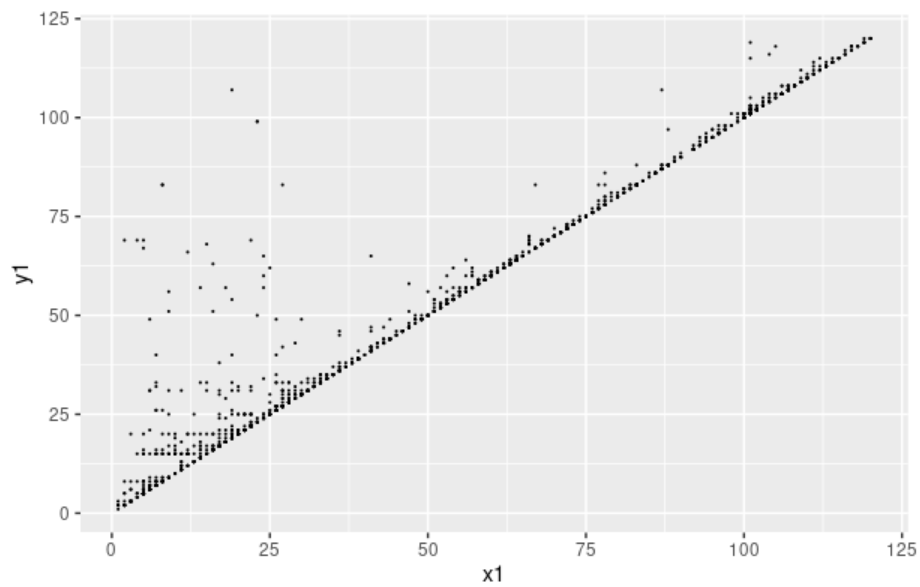


Figure A.116: Openmeetings

- Observation period: Approximately 125 months
- Total number of bugs: 1785
- Percentage of languages (%): Java82.7,HTML8.8,JavaScript6.0,CSS2.1,shell0.2,XSLT0.1,other0.1
- URL for the software overview: <https://openmeetings.apache.org/>

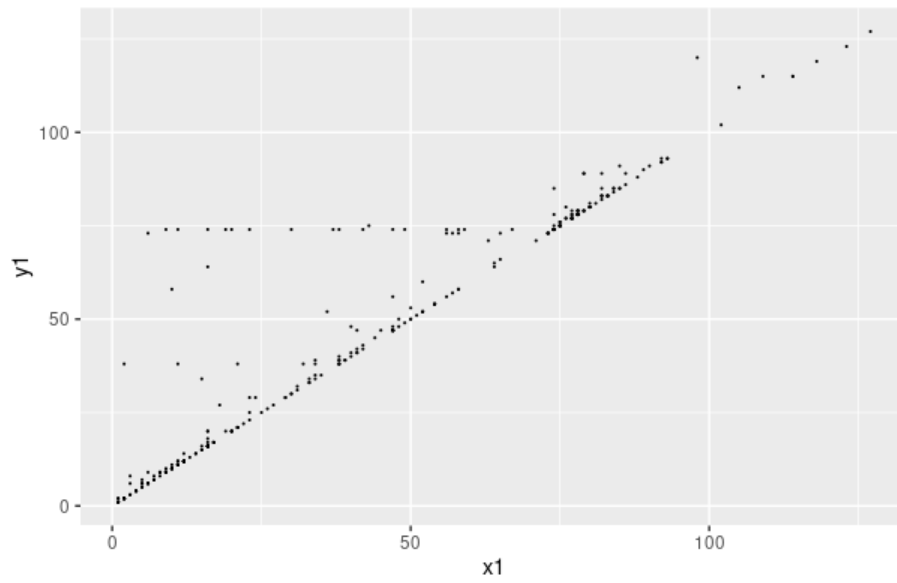


Figure A.117: OpenNLP

- Observation period: Approximately 150 months
- Total number of bugs: 357
- Percentage of languages (%): Java99.5, HTML0.2, batchfile0.2, shell0.1
- URL for the software overview: <https://opennlp.apache.org/>

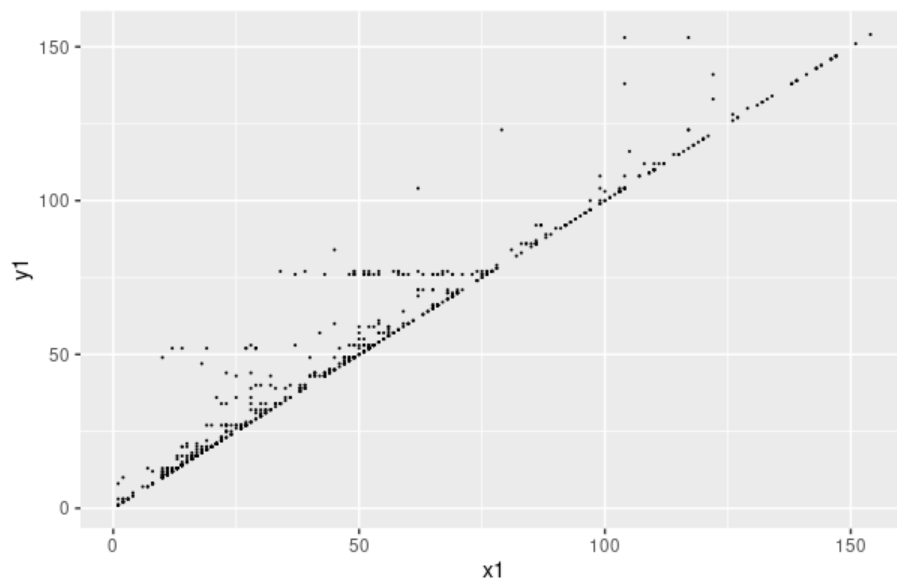


Figure A.118: OpenWebBeans

- Observation period: Approximately 150 months
- Total number of bugs: 813
- Percentage of languages (%): Java99.7, other0.3
- URL for the software overview: <https://openwebbeans.apache.org/>

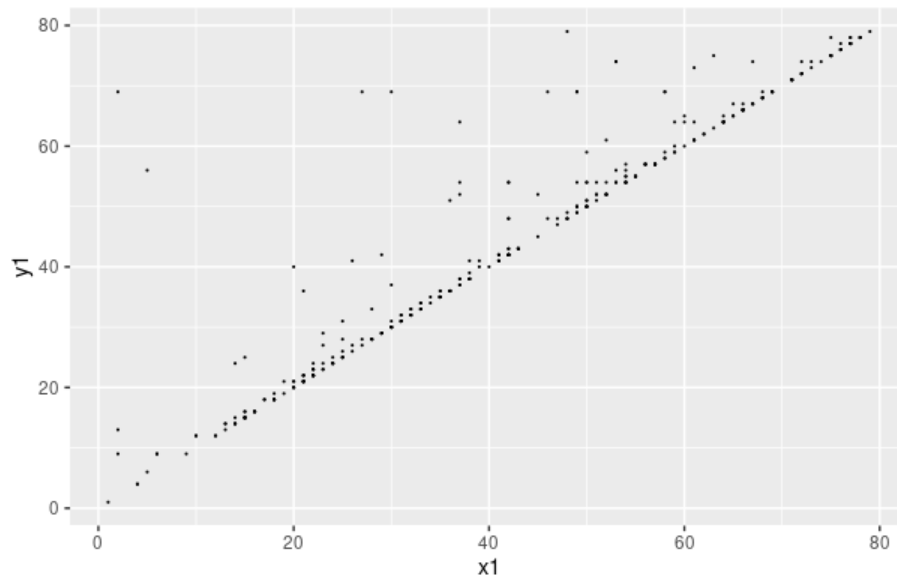


Figure A.119: Orc

- Observation period: Approximately 80 months
- Total number of bugs: 437
- Percentage of languages (%): HTML78.2, Java13.1, C++8.0 CSS0.3, other0.4
- URL for the software overview: <https://orc.apache.org/>

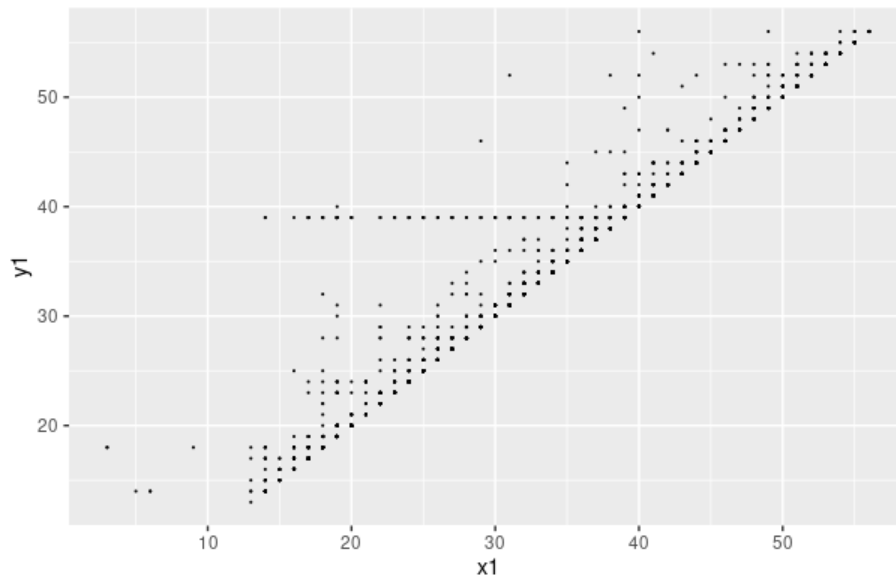


Figure A.120: Ozone

- Observation period: Approximately 60 months
- Total number of bugs: 2243
- Percentage of languages (%): Java97.0, shell0.7, TypeScript0.6, C++0.4, python0.4, HTML0.4, other0.5
- URL for the software overview: <https://ozone.apache.org/>

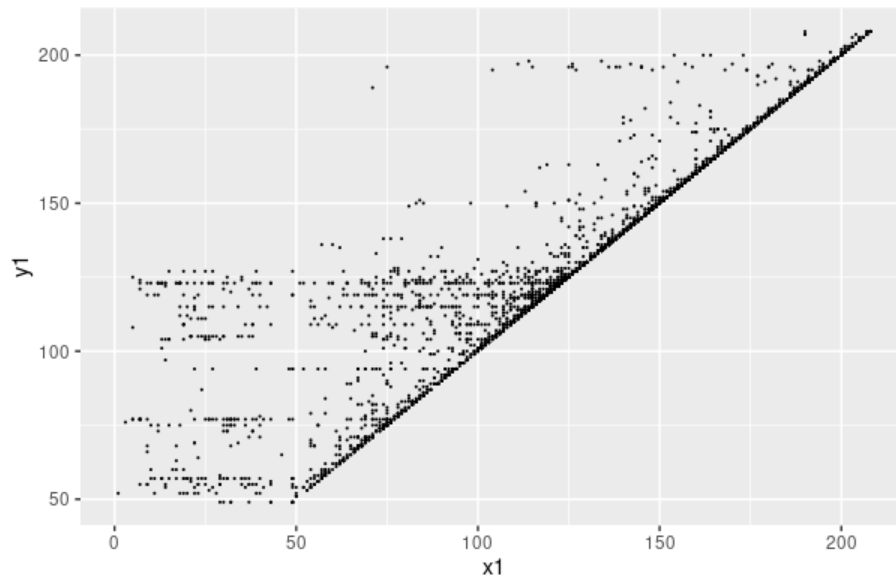


Figure A.121: PDFBox

- Observation period: Approximately 200 months
- Total number of bugs: 3797
- Percentage of languages (%): Java99.2, other0.8
- URL for the software overview: <https://pdfbox.apache.org/>

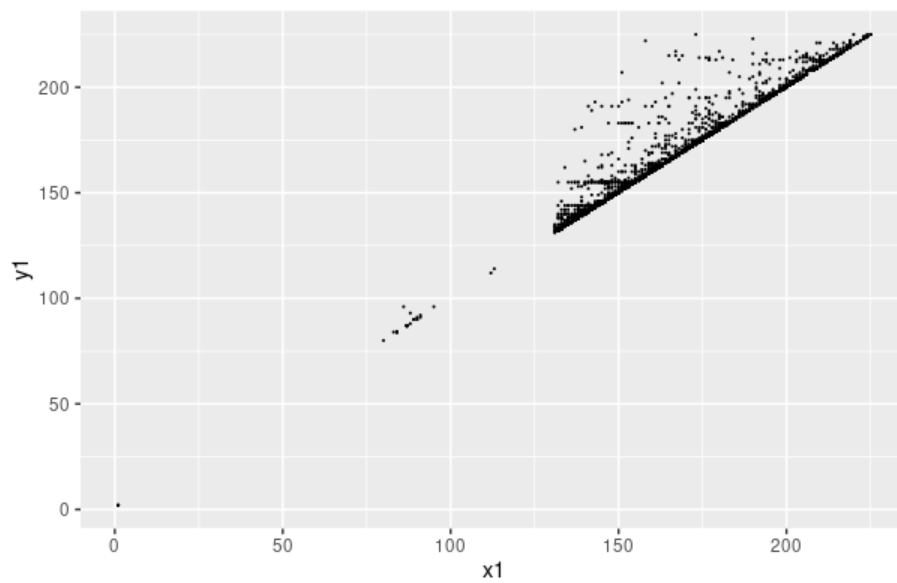


Figure A.122: Phoenix

- Observation period: Approximately 225 months
- Total number of bugs: 3729
- Percentage of languages (%): Java97.3, JavaScript1.0, python0.7, shell0.7, HTML0.1, other0.2
- URL for the software overview: <https://phoenix.apache.org/>

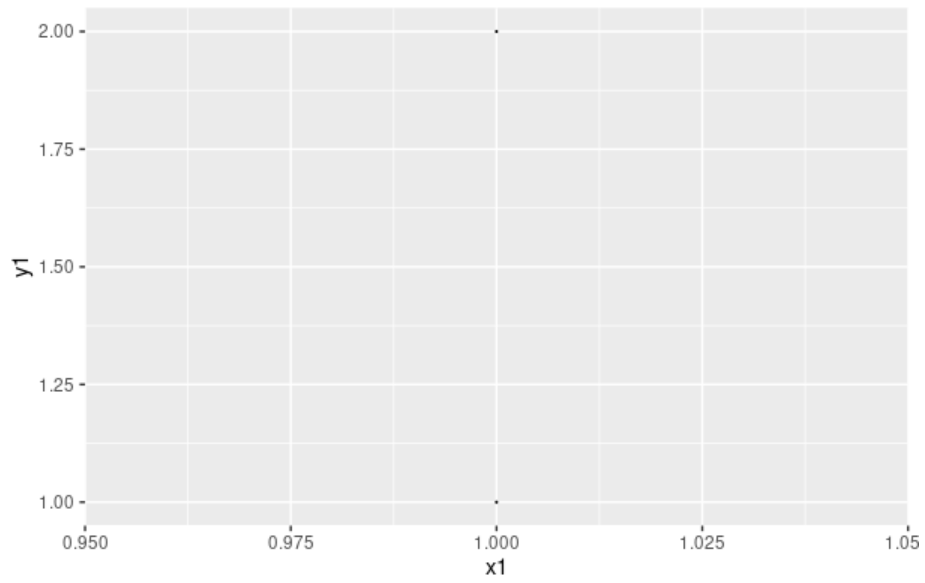


Figure A.123: Pinot

- Observation period: Approximately 2 months
- Total number of bugs: 10
- Percentage of languages (%): Java96.7, TypeScript2.2 shell0.4, scala0.4, HTML0.1, other0.2
- URL for the software overview: <https://pinot.apache.org/>

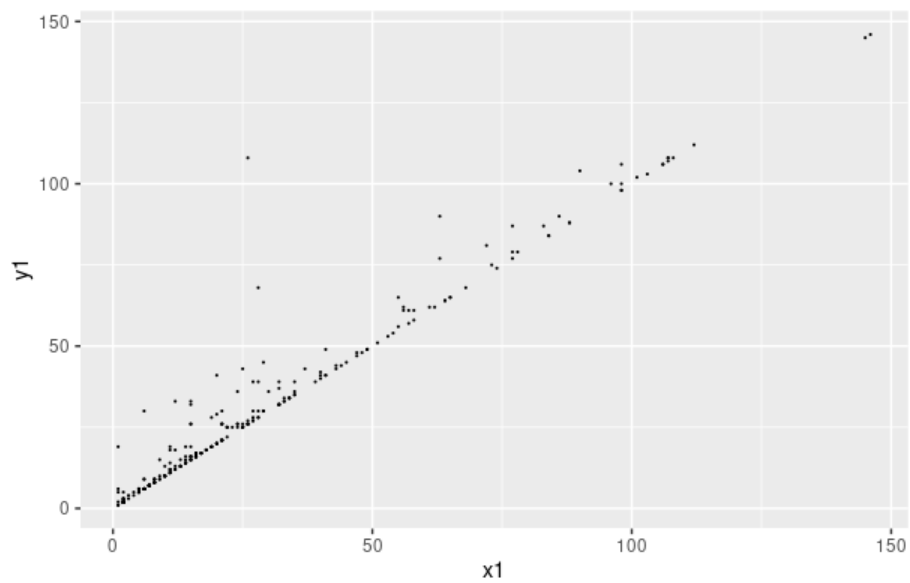


Figure A.124: Pivot

- Observation period: Approximately 150 months
- Total number of bugs: 335
- Percentage of languages (%): Java98.9, HTML0.6, JavaScript0.3, XSLT0.2
- URL for the software overview: <https://pivot.apache.org/>

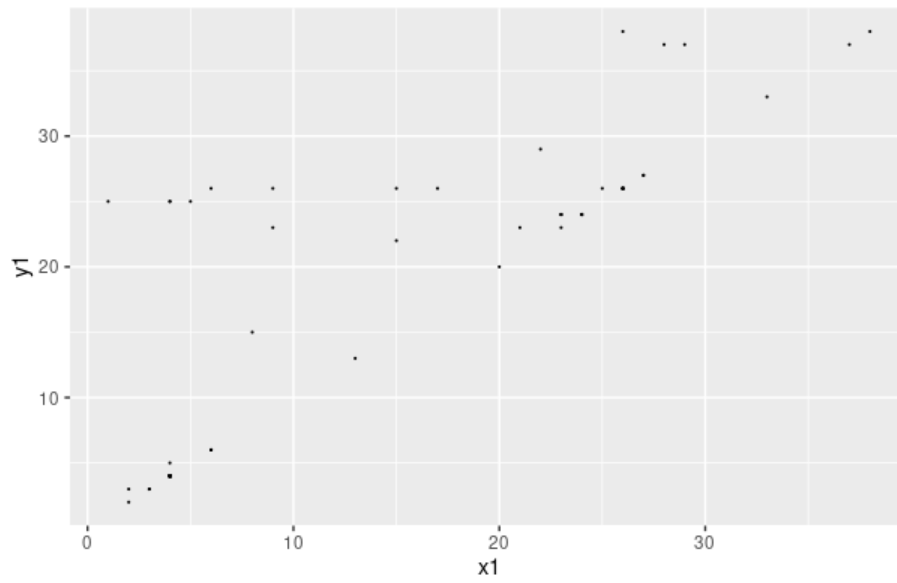


Figure A.125: Polygene

- Observation period: Approximately 40 months
- Total number of bugs: 79
- Percentage of languages (%): Java92.9, HTML2.8, Groovy1.9, JavaScript0.8
- URL for the software overview: <https://github.com/apache/polygene-Java>

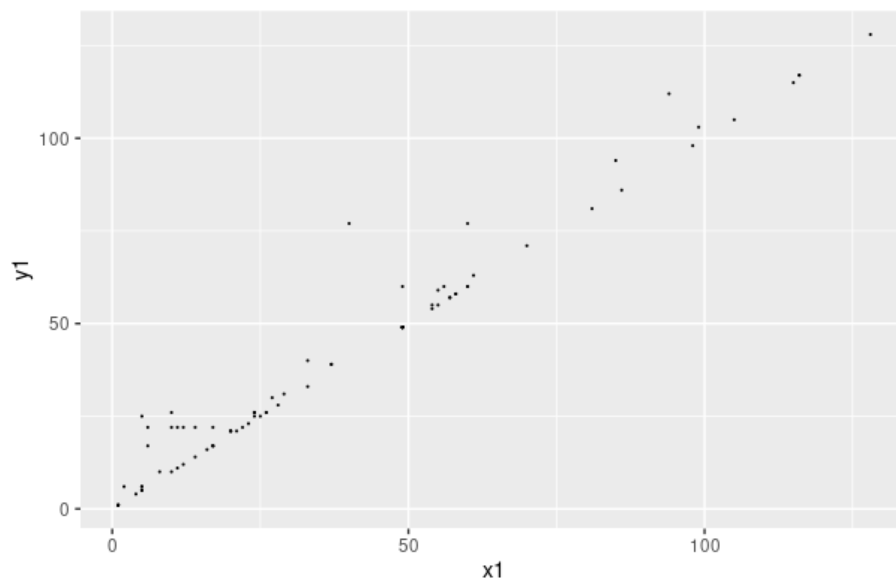


Figure A.126: Portals

- Observation period: Approximately 150 months
- Total number of bugs: 100
- Percentage of languages (%): Java95.9, JavaScript3.7, other0.4
- URL for the software overview: <https://portals.apache.org/>

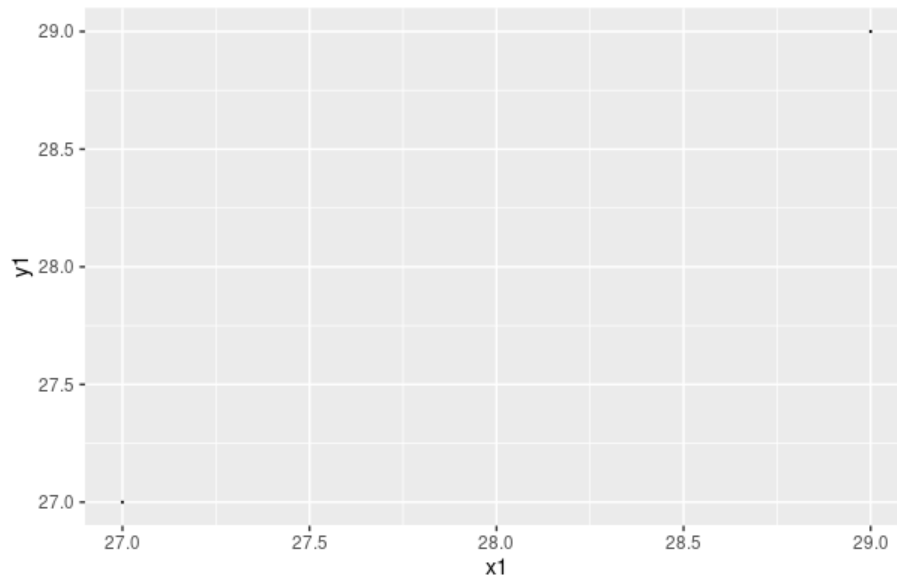


Figure A.127: Pulsar

- Observation period: Approximately 30 months
- Total number of bugs: 5
- Percentage of languages (%): Java89.6,C++6.4,python1.5,JavaScript0.9,shell0.7,GO0.4,other0.5
- URL for the software overview: <https://pulsar.apache.org/>

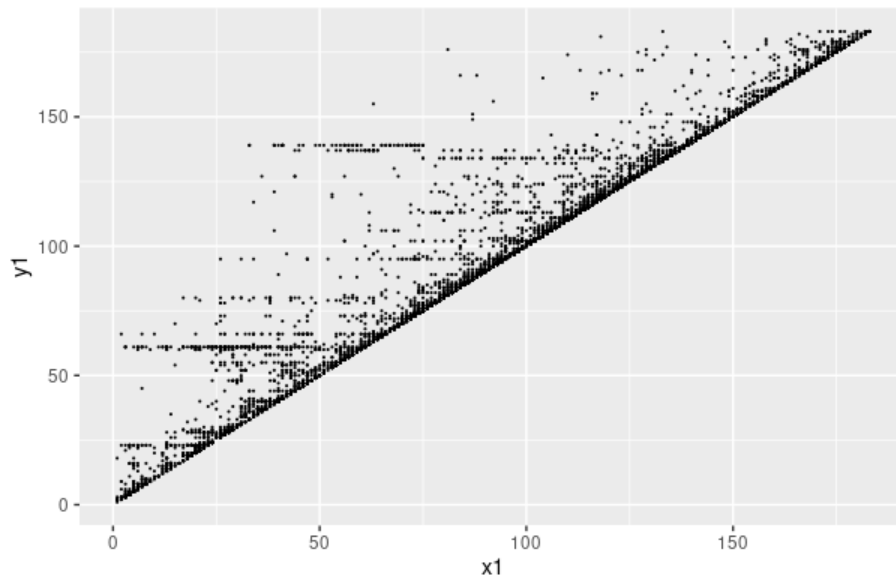


Figure A.128: Qpid

- Observation period: Approximately 175 months
- Total number of bugs: 8222
- Percentage of languages (%): Java64.1, JavaScript20.5, shell4.8, HTML3.3, CSS2.9, XSLT1.9, other2.5
- URL for the software overview: <https://qpid.apache.org/>

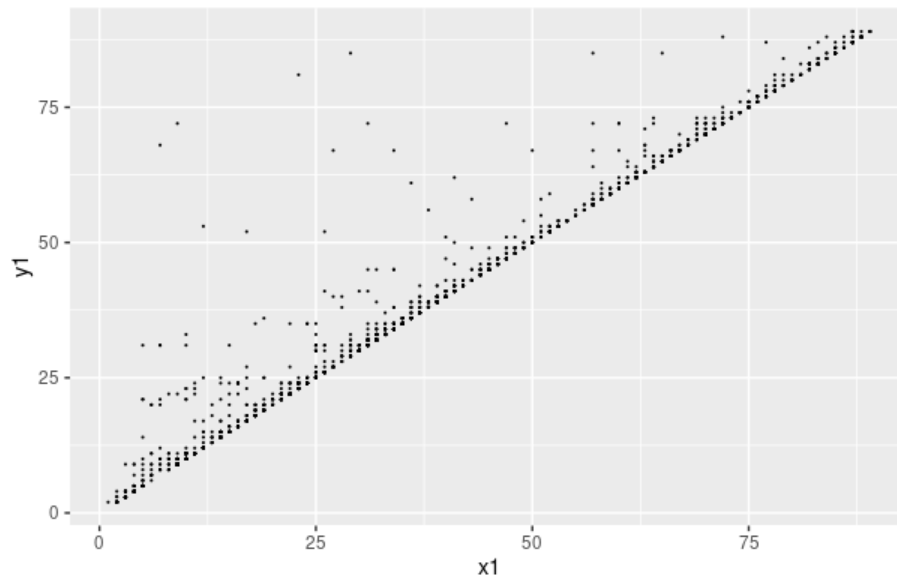


Figure A.129: Ranger

- Observation period: Approximately 75 months
- Total number of bugs: 2325
- Percentage of languages (%): Java68.7, JavaScript15.4, python4.7, shell2.5, HTML1.8, other6.9
- URL for the software overview: <https://ranger.apache.org/>

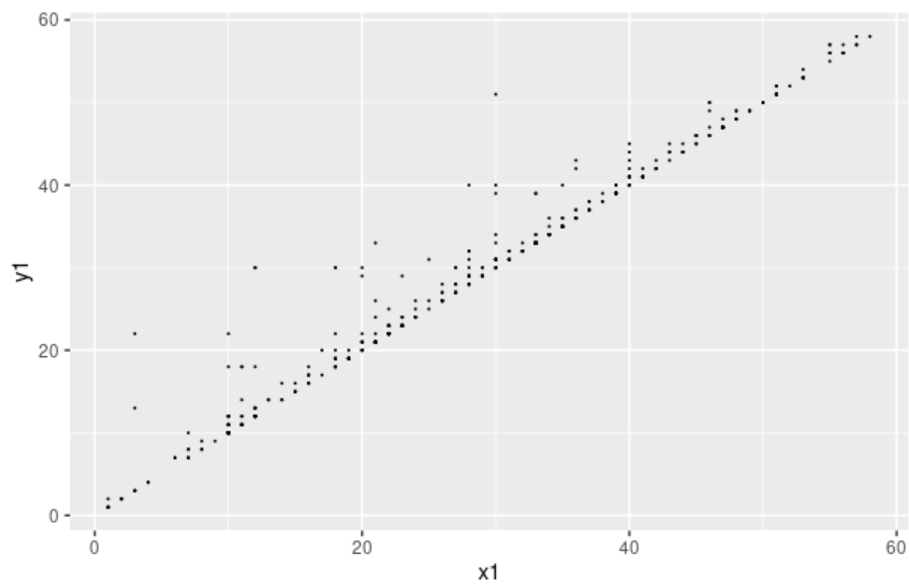
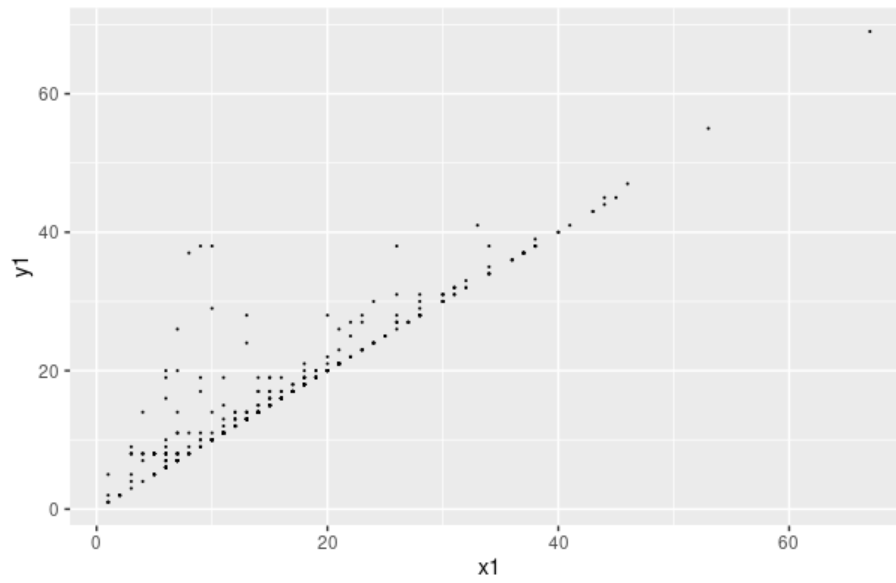


Figure A.130: Ratis

- Observation period: Approximately 60 months
- Total number of bugs: 552
- Percentage of languages (%): Java97.0, shell2.1, other0.9
- URL for the software overview: <https://ratis.apache.org/>

**Figure A.131:** REEF

- Observation period: Approximately 70 months
- Total number of bugs: 500
- Percentage of languages (%): Java52.9,C#44.8,C++1.5,shell0.2, docker-file0.2,python0.2,other0.2
- URL for the software overview: <https://reef.apache.org/>

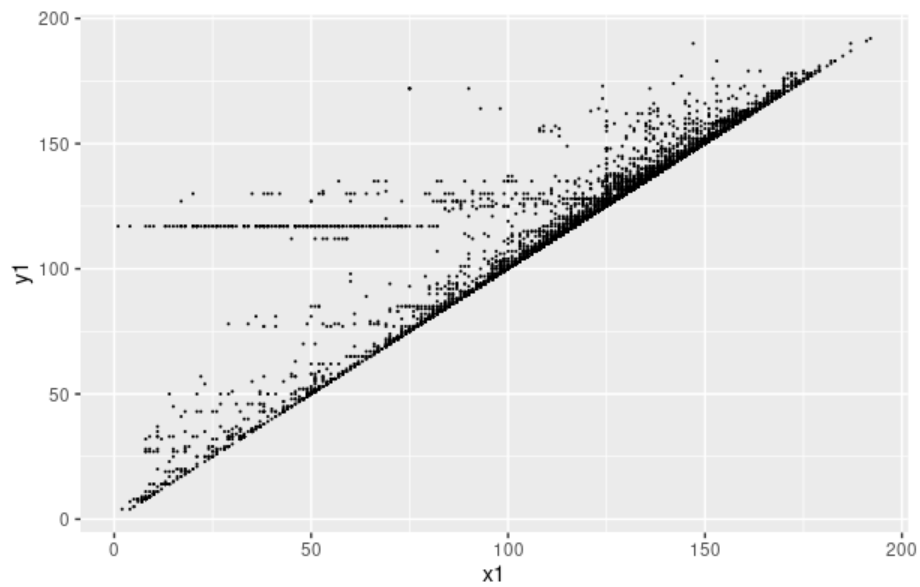


Figure A.132: Retired

- Observation period: Approximately 200 months
- Total number of bugs: 11681
- Percentage of languages (%): Java100
- URL for the software overview: <https://github.com/apache/incubator-retired-tamaya-site>

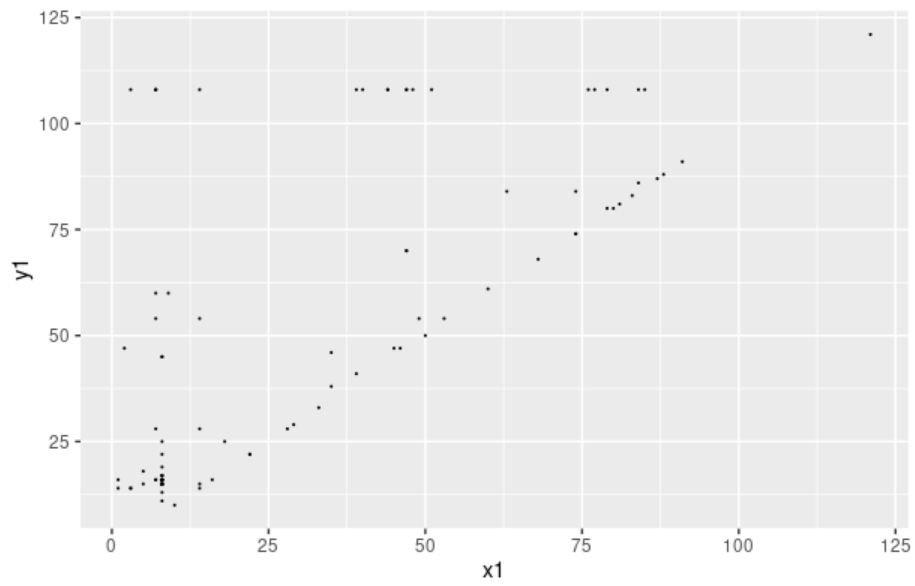


Figure A.133: River

- Observation period: Approximately 125 months
- Total number of bugs: 172
- Percentage of languages (%): Java87.4, XSLT5.8, HTML5.4, CSS1.2, other0.2
- URL for the software overview: <https://river.apache.org/>

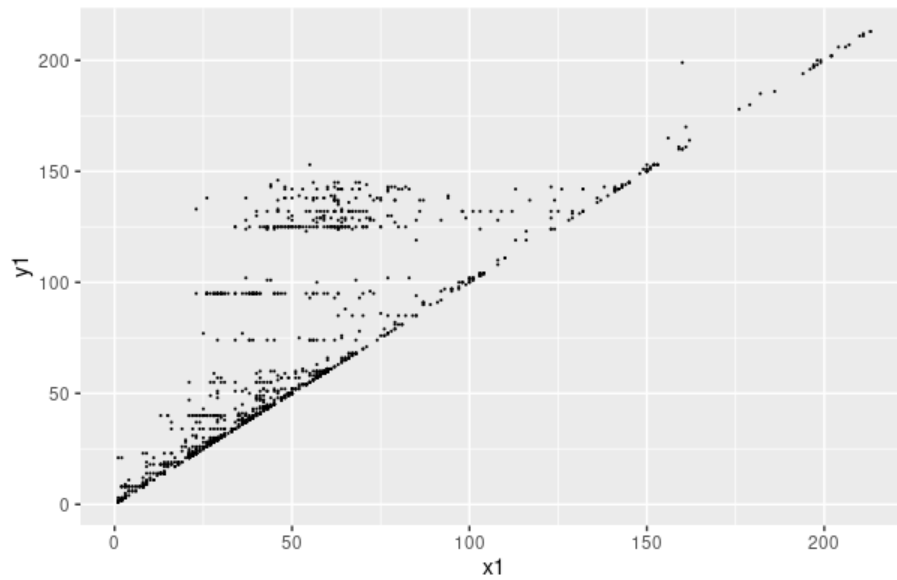


Figure A.134: Roller

- Observation period: Approximately 200 months
- Total number of bugs: 1357
- Percentage of languages (%): Java96.6,CSS1.9,JavaScript0.7,XSLT0.3,shell0.2,python0.1,other0.2
- URL for the software overview: <https://roller.apache.org/>

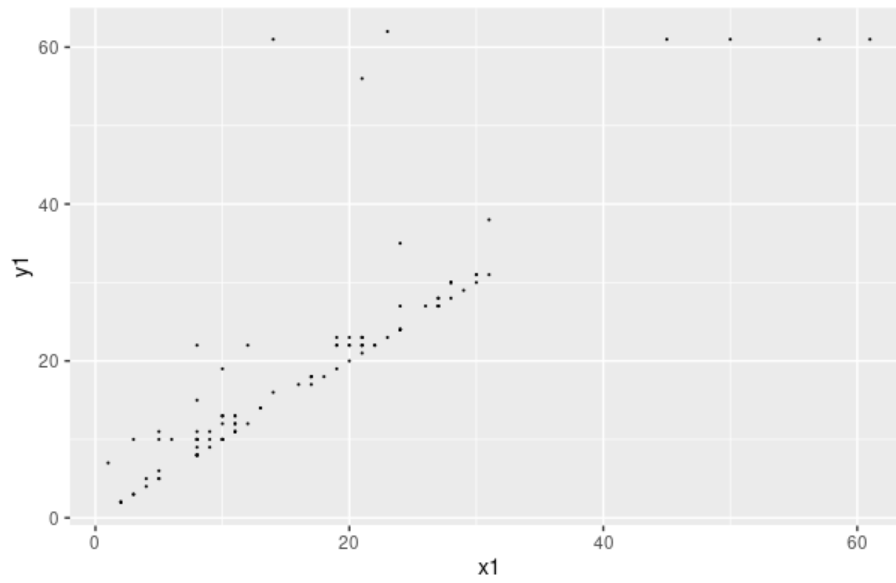


Figure A.135: Rya

- Observation period: Approximately 60 months
- Total number of bugs: 171
- Percentage of languages (%): Java99.6, other0.4
- URL for the software overview: <https://rya.apache.org/>

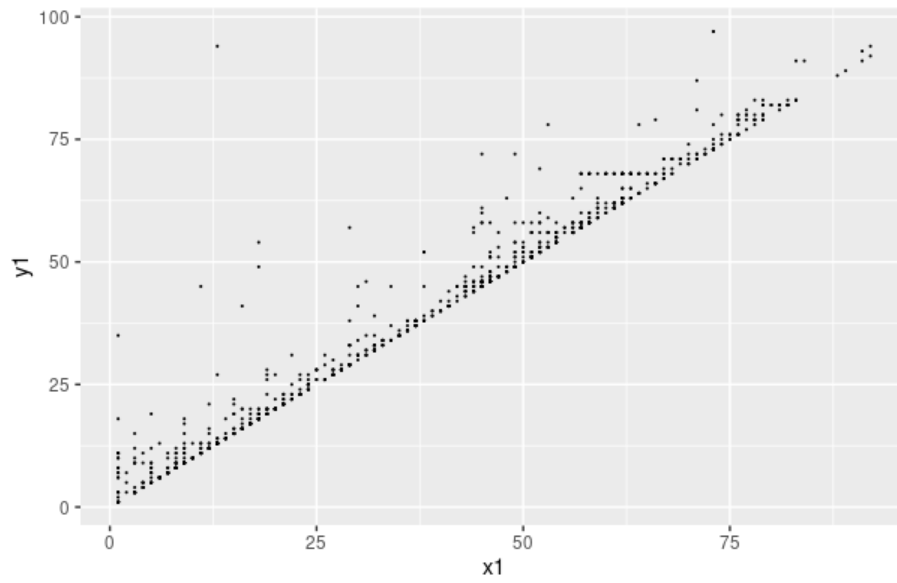


Figure A.136: Samza

- Observation period: Approximately 100 months
- Total number of bugs: 1460
- Percentage of languages (%): Java87.0, scala11.4, python0.9, shell0.5, scala0.1, XSLT0.1
- URL for the software overview: <https://samza.apache.org/>

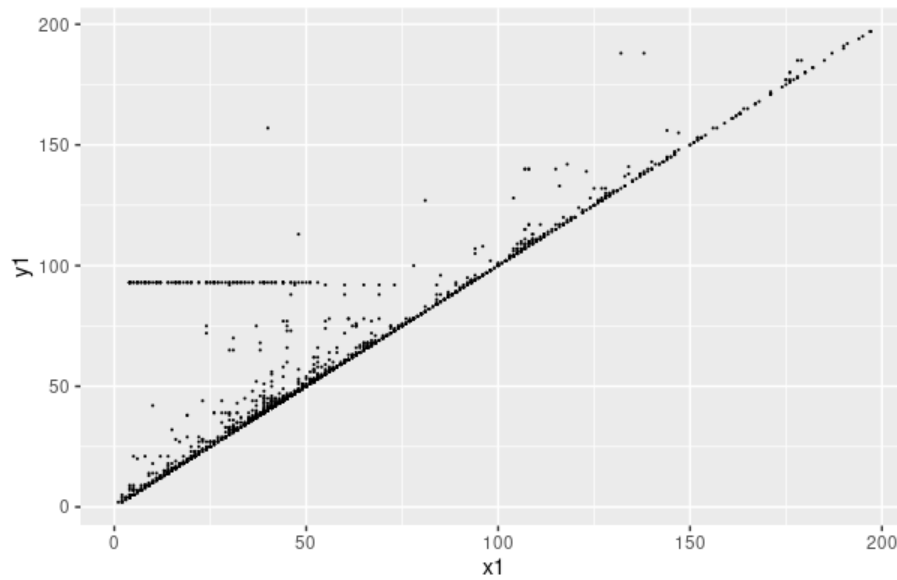


Figure A.137: ServiceMix

- Observation period: Approximately 200 months
- Total number of bugs: 2440
- Percentage of languages (%): Java97.9, HTML1.1, Groovy1.0
- URL for the software overview: <https://servicemix.apache.org/>

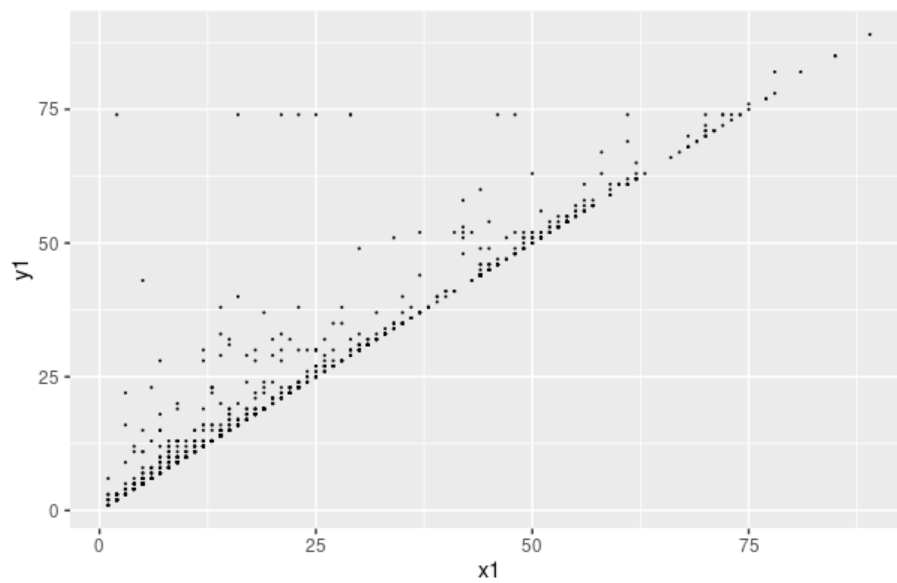


Figure A.138: shindig

- Observation period: Approximately 100 months
- Total number of bugs: 1032
- Percentage of languages (%): Java62.2, php19.0, JavaScript16.1, HTML2.3, shell2.3, shell0.1,XSLT0.1, other0.1
- URL for the software overview: <https://github.com/apache/shindig>

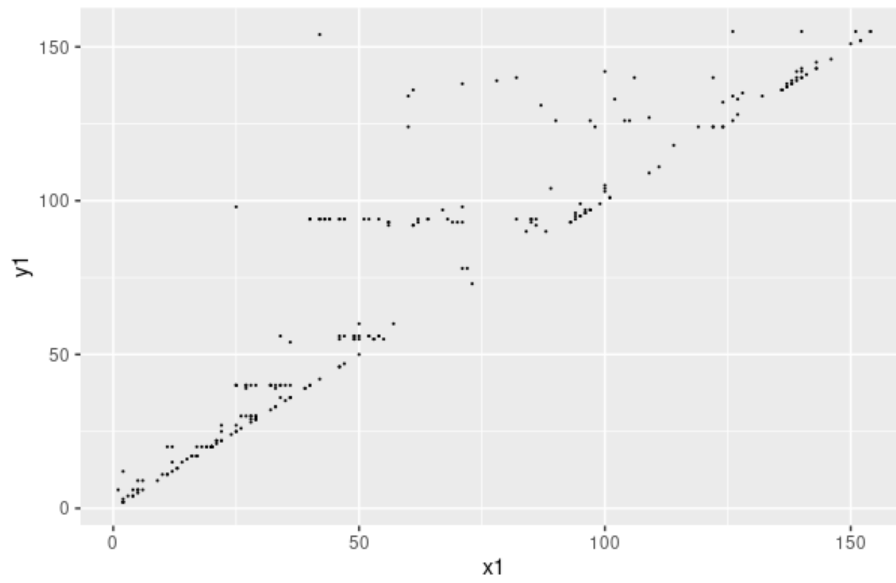


Figure A.139: shiro

- Observation period: Approximately 150 months
- Total number of bugs: 330
- Percentage of languages (%): Java89.6, Groovy10.2, other0.2
- URL for the software overview: <https://shiro.apache.org/>

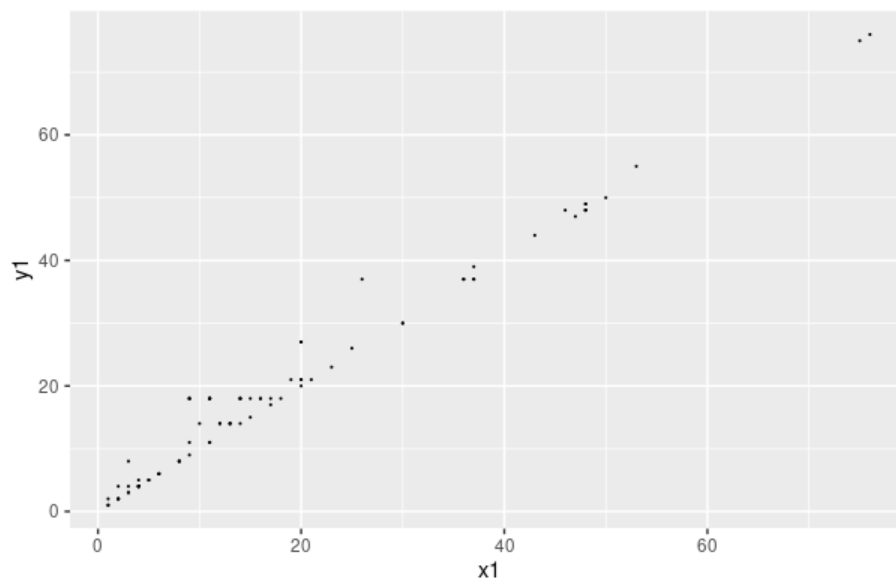
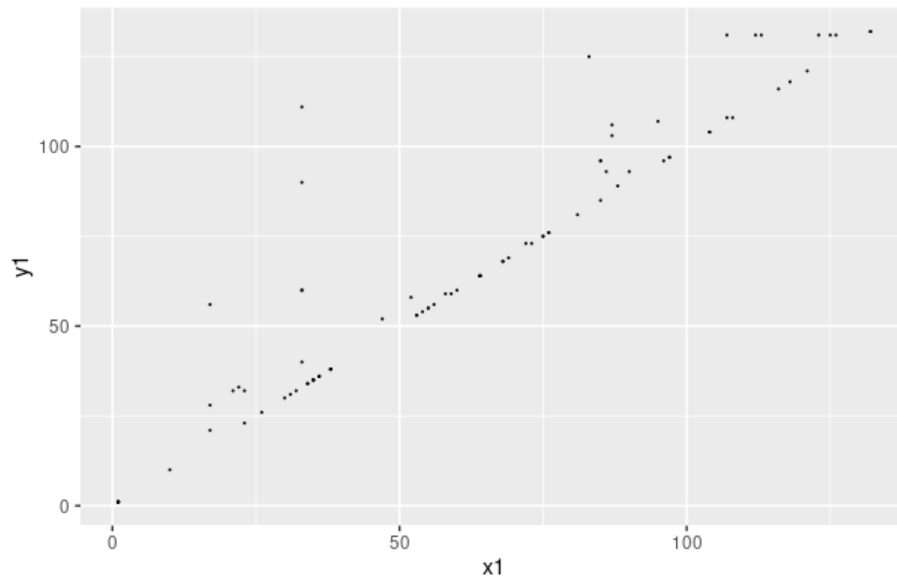


Figure A.140: Singa

- Observation period: Approximately 80 months
- Total number of bugs: 130
- Percentage of languages (%): C++70.0, python21.0, C4.7, other4.3
- URL for the software overview: <https://github.com/apache/singa>

**Figure A.141:** SIS

- Observation period: Approximately 150 months
- Total number of bugs: 103
- Percentage of languages (%): Java99.7, other0.3
- URL for the software overview: <https://sis.apache.org/>

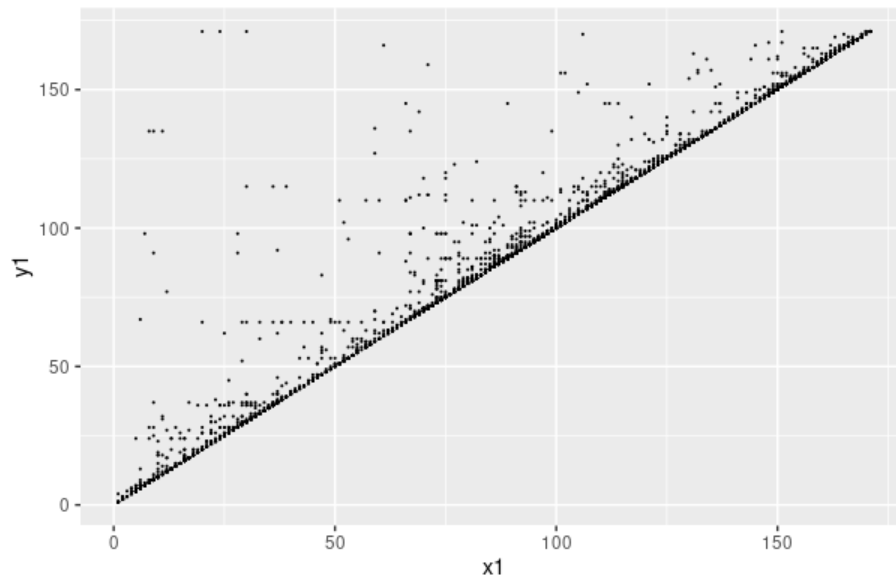
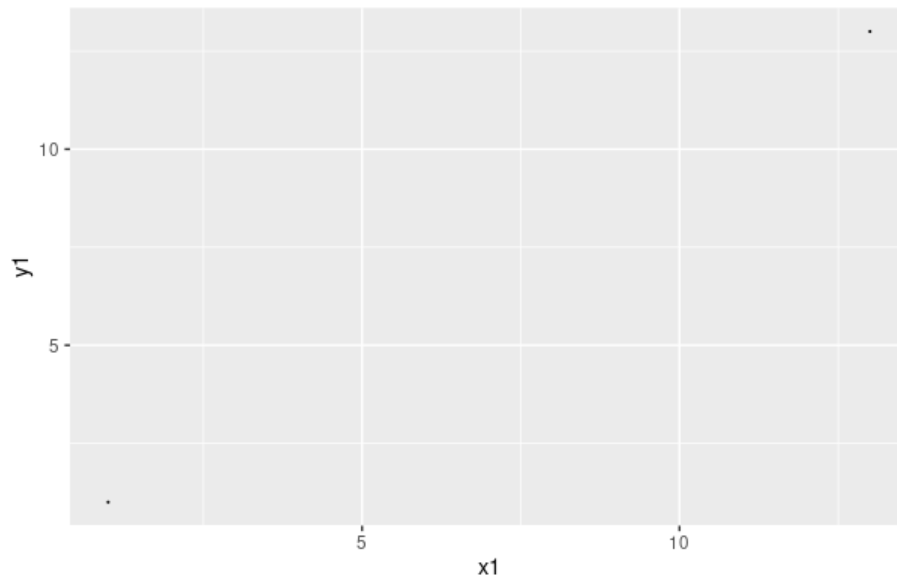


Figure A.142: Sling

- Observation period: Approximately 175 months
- Total number of bugs: 4222
- Percentage of languages (%): Java100
- URL for the software overview: <https://sling.apache.org/>

**Figure A.143:** Steve

- Observation period: Approximately 15 months
- Total number of bugs: 18
- Percentage of languages (%): python47.7, JavaScript33.6, HTML14.5, CSS3.9, dockerfile0.3
- URL for the software overview: <https://steve.apache.org/>

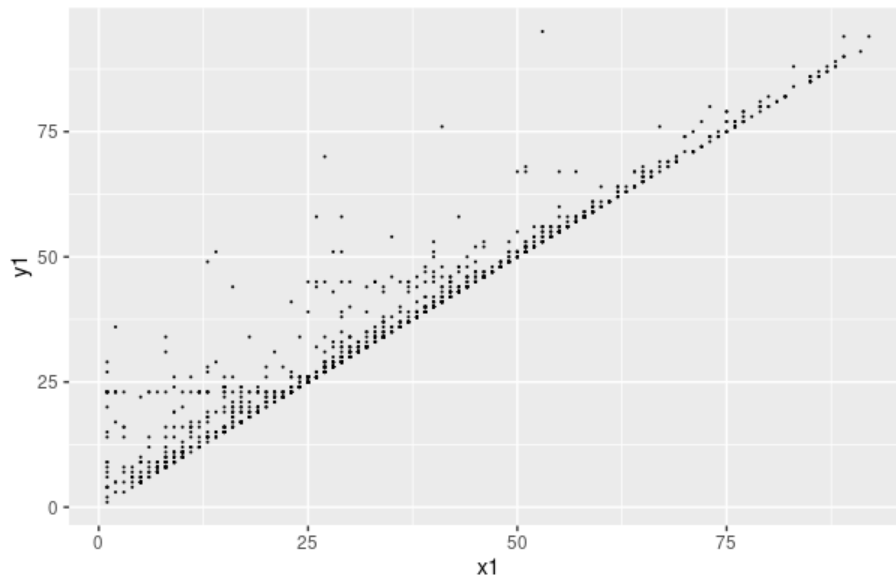


Figure A.144: Storm

- Observation period: Approximately 90 months
- Total number of bugs: 1760
- Percentage of languages (%): Java83.6, python7.3, HTML4.3, JavaScript0.5, other2.5
- URL for the software overview: <https://storm.apache.org/>

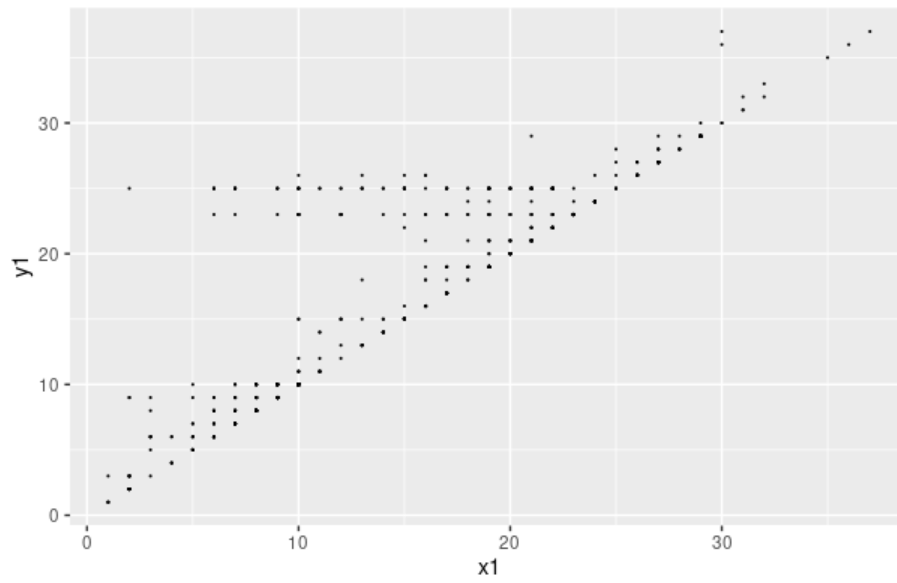


Figure A.145: Stratos

- Observation period: Approximately 40 months
- Total number of bugs: 768
- Percentage of languages (%): Java56.6, JavaScript34.2, python5.6, HTML1.3, shell1.1, CSS0.9, other0.3
- URL for the software overview: <https://github.com/apache/stratos>

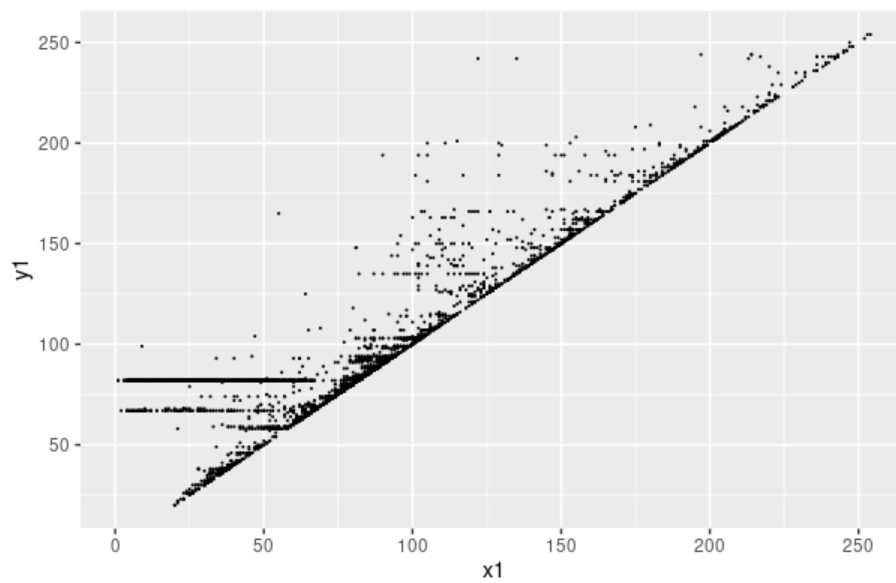


Figure A.146: StrutsFrameWork

- Observation period: Approximately 250 months
- Total number of bugs: 4875
- Percentage of languages (%): Java87.3, HTML9.8, freemarker 2.3, other0.6
- URL for the software overview: <https://github.com/apache/struts>

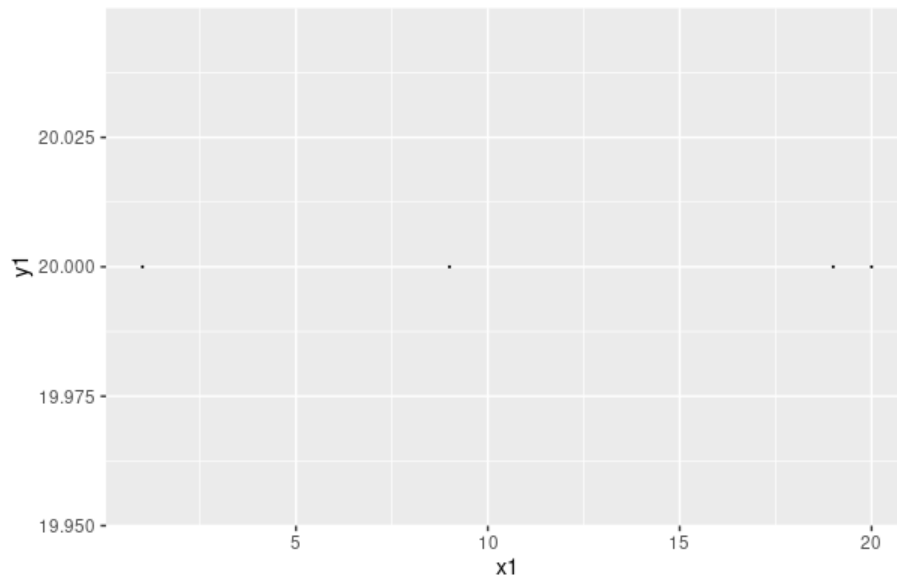


Figure A.147: Superset

- Observation period: Approximately 20 months
- Total number of bugs: 9
- Percentage of languages (%): python49.8, TypeScript29.4, JavaScript17.1, HTML1.3, shell1.1, other1.3
- URL for the software overview: <https://superset.apache.org/>

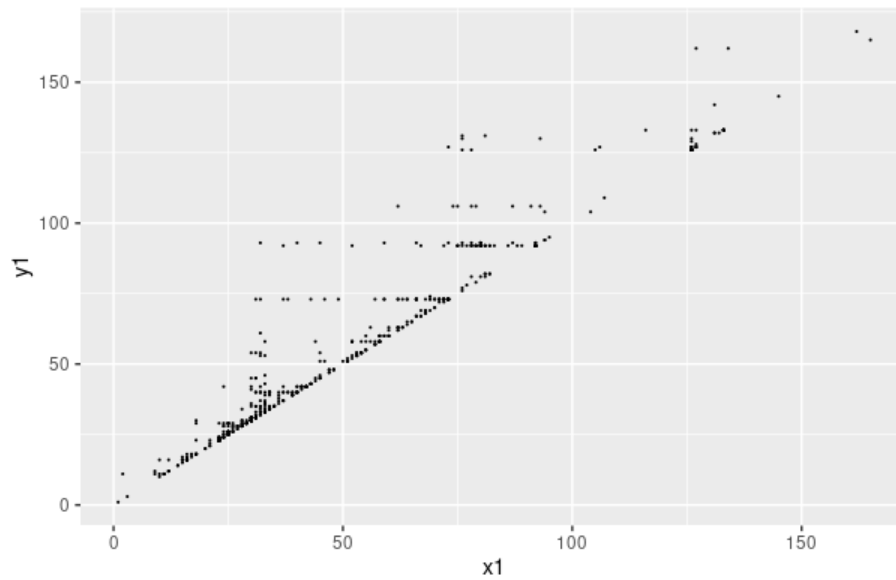


Figure A.148: Synapse

- Observation period: Approximately 150 months
- Total number of bugs: 644
- Percentage of languages (%): Java99.0, shell0.5, XSLT0.3, batchfile0.2
- URL for the software overview: <https://synapse.apache.org/>

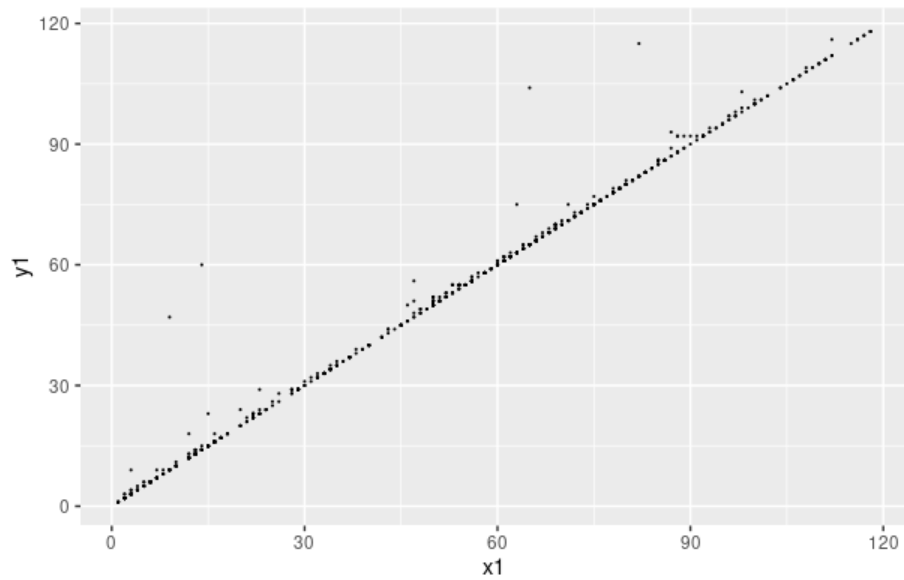


Figure A.149: Syncope

- Observation period: Approximately 120 months
- Total number of bugs: 842
- Percentage of languages (%): Java94.2, HTML3.8, Groovy0.5, CSS0.3 JavaScript0.2, other0.9
- URL for the software overview: <https://syncope.apache.org/>

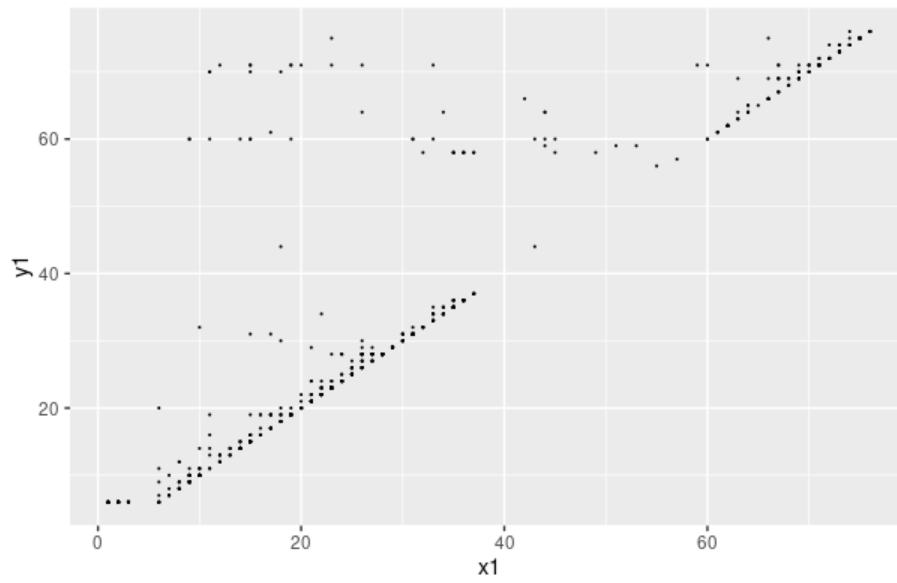


Figure A.150: SystemDS

- Observation period: Approximately 80 months
- Total number of bugs: 748
- Percentage of languages (%): Java88.0, R5.1, python3.5, shell1.4, C++0.4, other1.6
- URL for the software overview: <https://systemds.apache.org/>

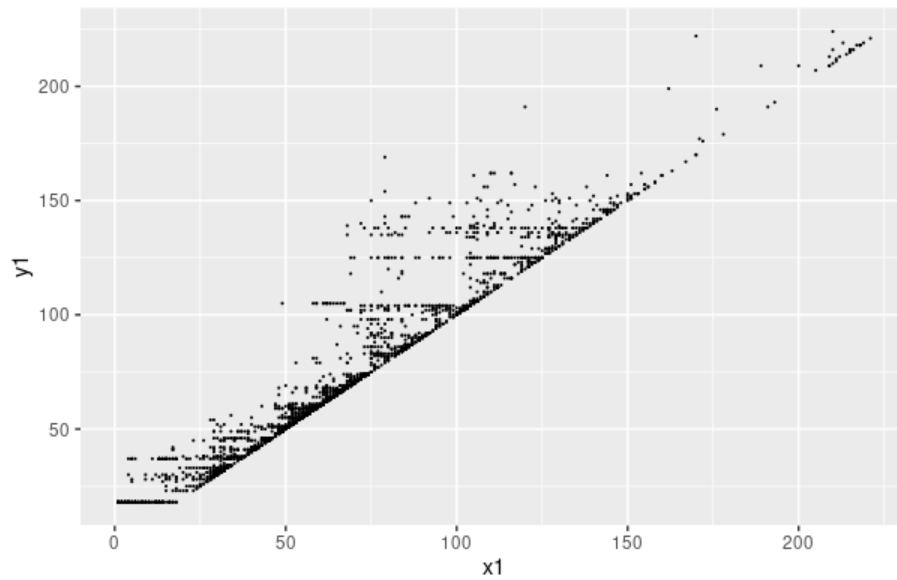


Figure A.151: Tapestry

- Observation period: Approximately 200 months
- Total number of bugs: 2935
- Percentage of languages (%): Java50.8, JavaScript39.1, roff5.2, Groovy3.7, HTML0.2, other1.0
- URL for the software overview: <https://tapestry.apache.org/>

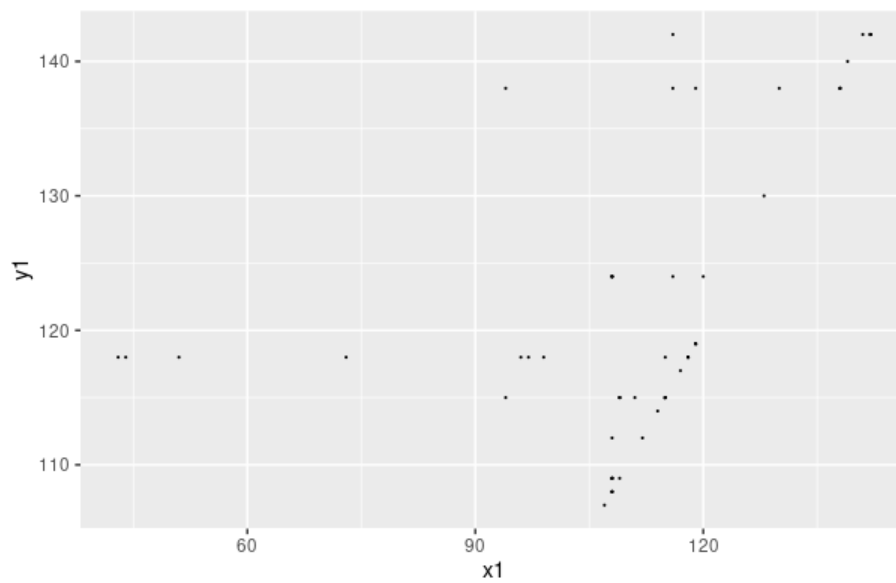


Figure A.152: Taverna

- Observation period: Approximately 140 months
- Total number of bugs: 681
- Percentage of languages (%): Java95.9, JavaScript3.6, other0.5
- URL for the software overview: <https://github.com/apache/incubator-taverna-common-activities>

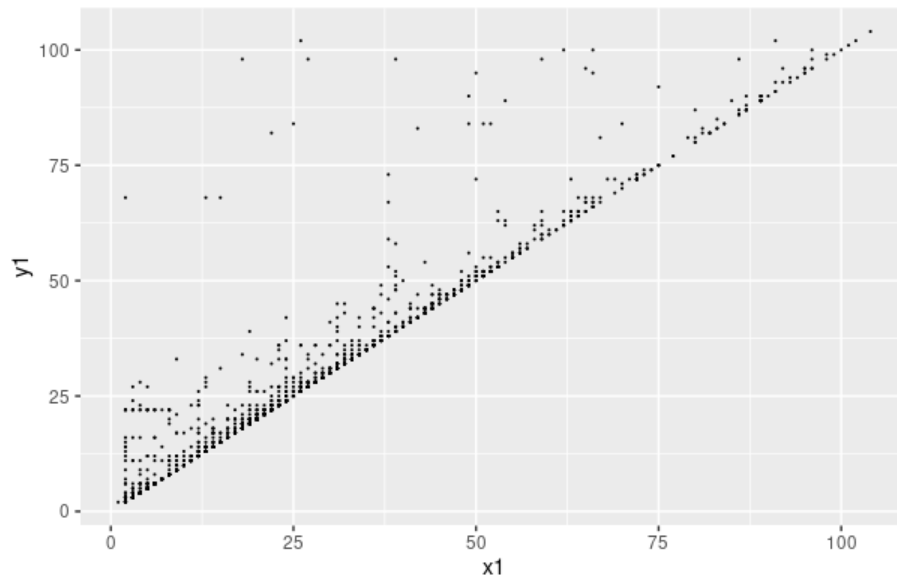


Figure A.153: Tez

- Observation period: Approximately 100 months
- Total number of bugs: 2371
- Percentage of languages (%): Java90.0, JavaScript7.8, shell0.3, python0.2, other1.7
- URL for the software overview: <https://tez.apache.org/>

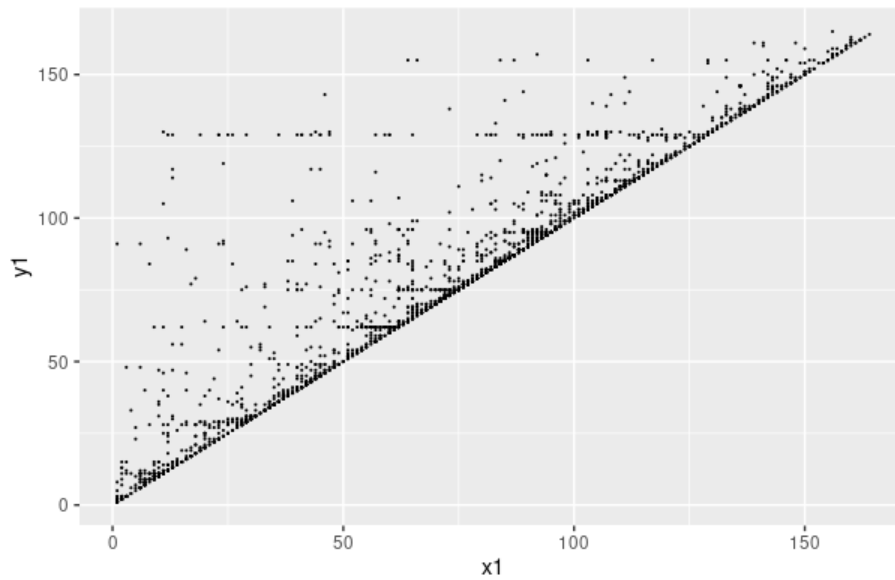


Figure A.154: Thrift

- Observation period: Approximately 175 months
- Total number of bugs: 3030
- Percentage of languages (%): C++33.0, Java9.4, C7.6, GO4.7, other45.3
- URL for the software overview: <https://thrift.apache.org/>

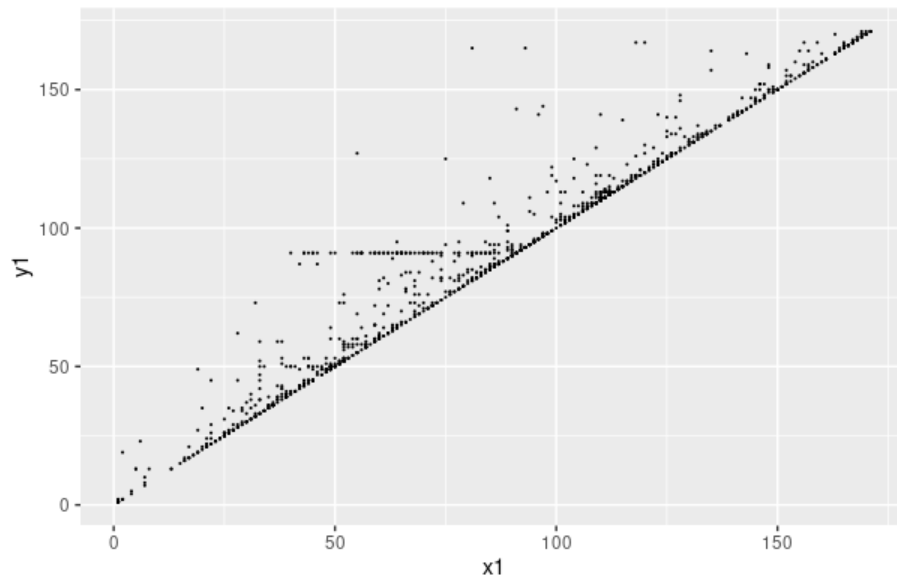


Figure A.155: Tika

- Observation period: Approximately 175 months
- Total number of bugs: 1710
- Percentage of languages (%): Java77.0, richtextformat19.8, HTML2.1, python0.6, shell0.3, other0.2
- URL for the software overview: <https://tika.apache.org/>

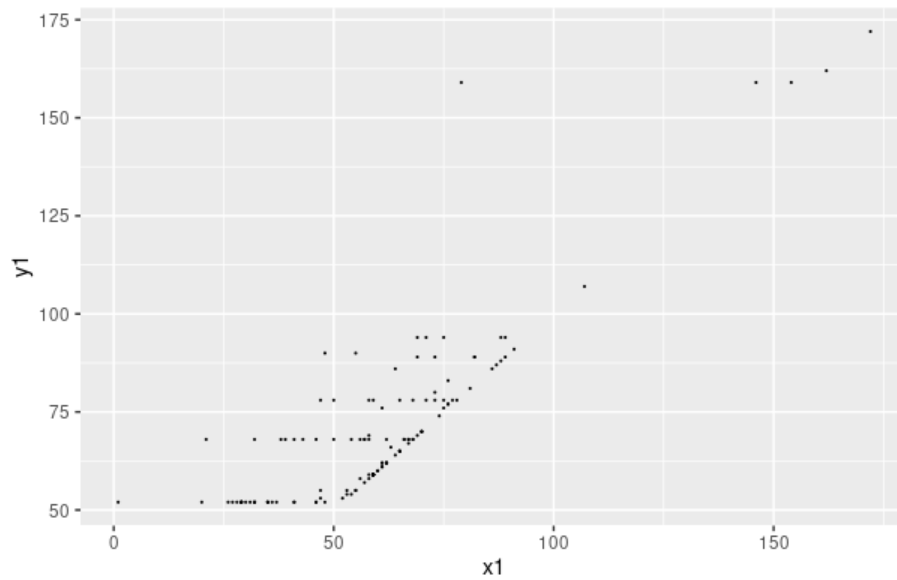


Figure A.156: Tomcat

- Observation period: Approximately 175 months
- Total number of bugs: 177
- Percentage of languages (%): Java97.6, HTML1.4, shell0.3, batchfile0.2, XSLT0.1, other0.4
- URL for the software overview: <https://tomcat.apache.org/>

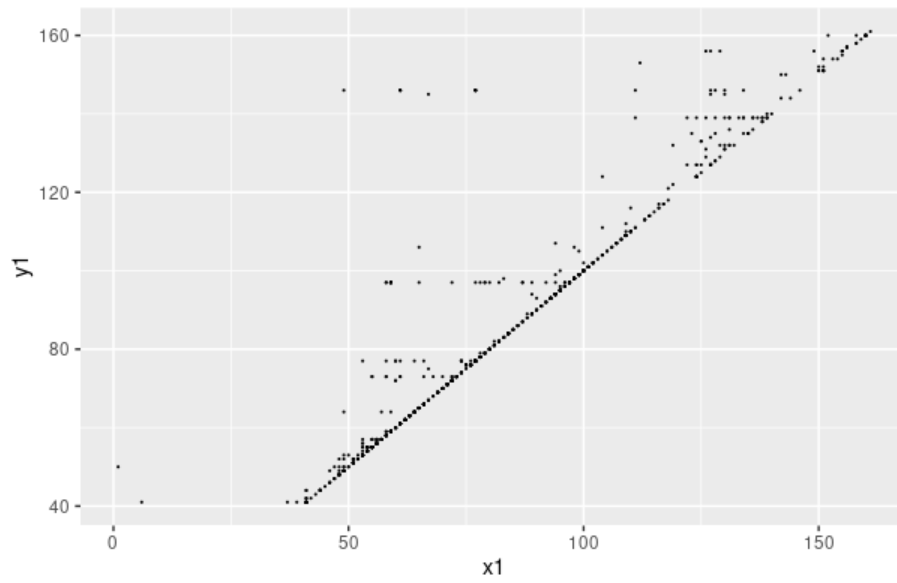


Figure A.157: TomEE

- Observation period: Approximately 160 months
- Total number of bugs: 1112
- Percentage of languages (%): Java99.0, JavaScript0.3, Groovy0.3, HTML0.1, batchfile0.1, shell0.1, other0.1
- URL for the software overview: <https://tomee.apache.org/>

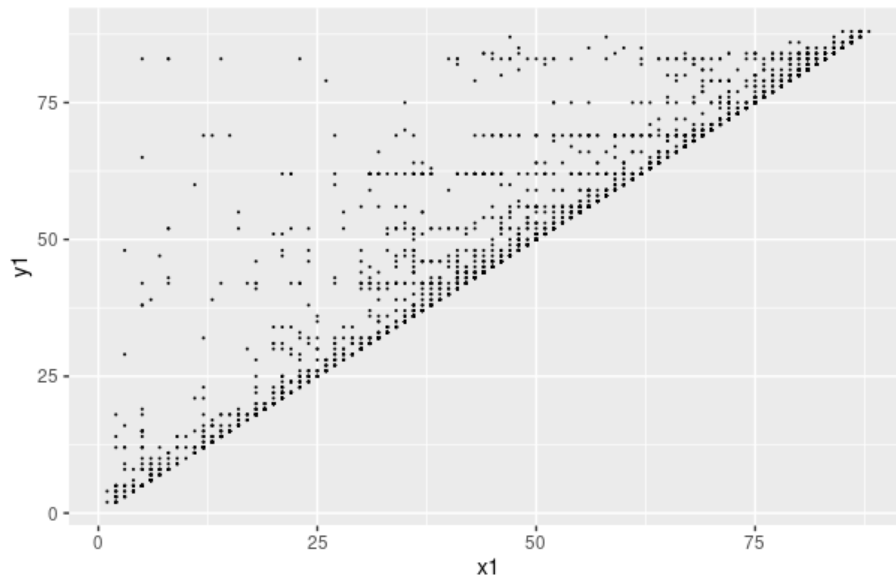


Figure A.158: Trafficserver

- Observation period: Approximately 90 months
- Total number of bugs: 3026
- Percentage of languages (%): C++80.7, C7.7, python7.1, makefile1.2, m41.0, shell0.9, other1.4
- URL for the software overview: <https://trafficserver.apache.org/>

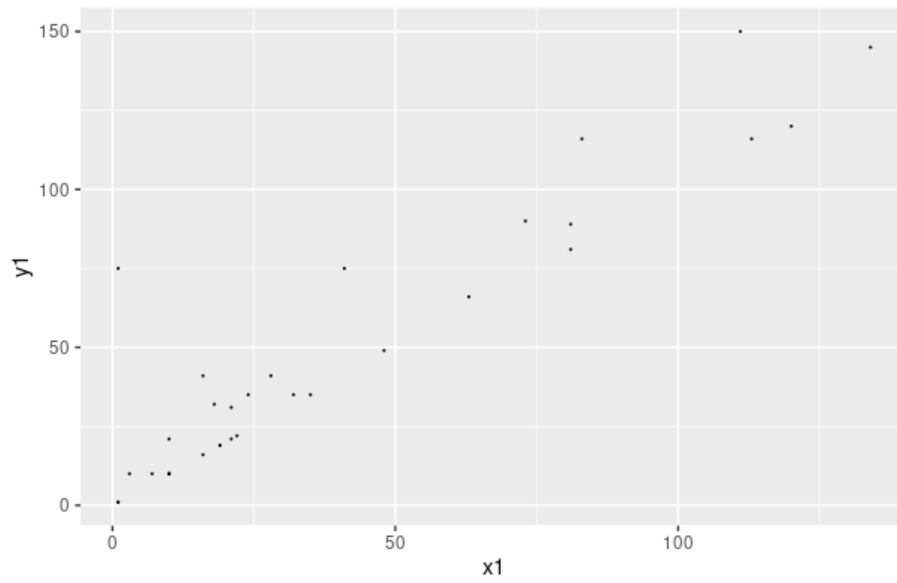


Figure A.159: Turbine

- Observation period: Approximately 150 months
- Total number of bugs: 40
- Percentage of languages (%): Java97.9, HTML2.0, other0.1
- URL for the software overview: <https://turbine.apache.org/>

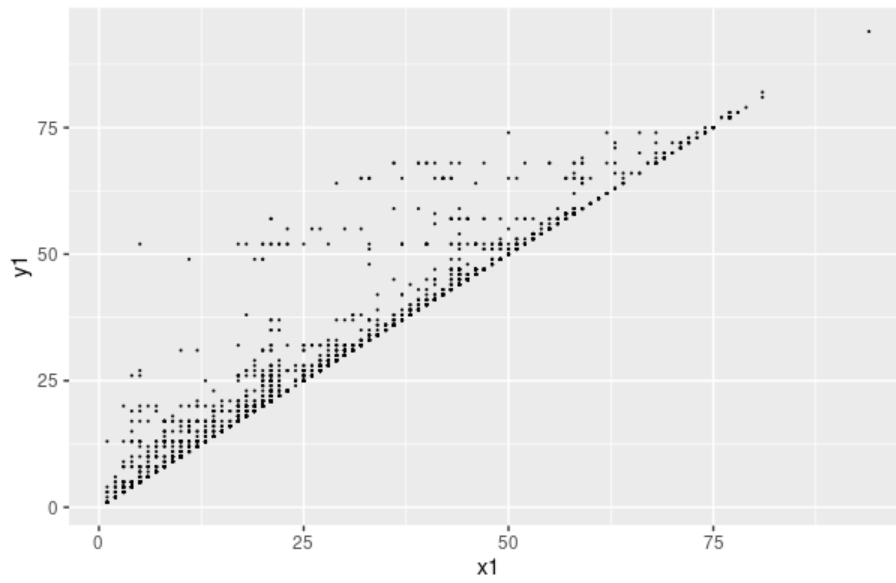


Figure A.160: Tuscany

- Observation period: Approximately 100 months
- Total number of bugs: 2826
- Percentage of languages (%): Java99.3, other0.7
- URL for the software overview: <https://github.com/apache/tuscany-das>

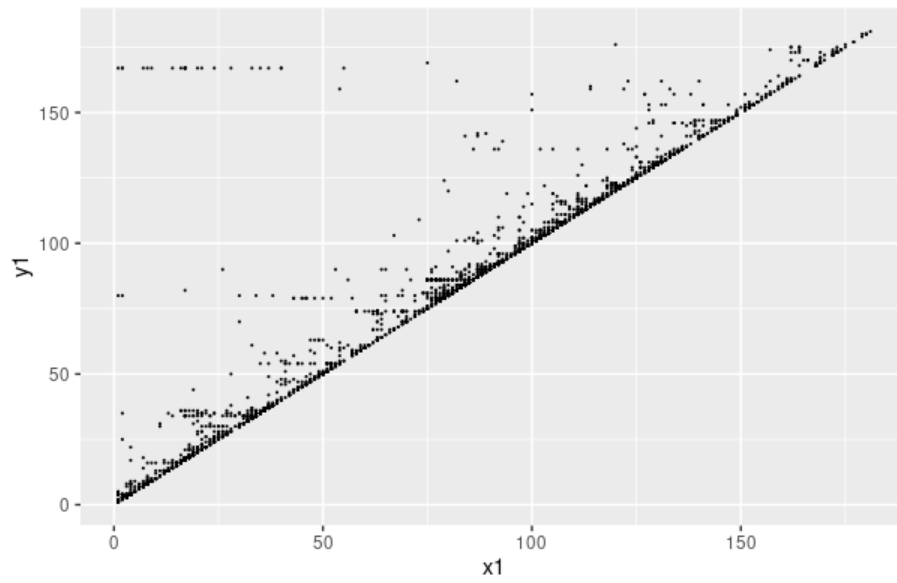


Figure A.161: UIMA

- Observation period: Approximately 175 months
- Total number of bugs: 3491
- Percentage of languages (%): Java98.5, HTML0.7, TypeScript0.3, batch-file0.2, shell0.2, XSLT0.1
- URL for the software overview: <https://uima.apache.org/>

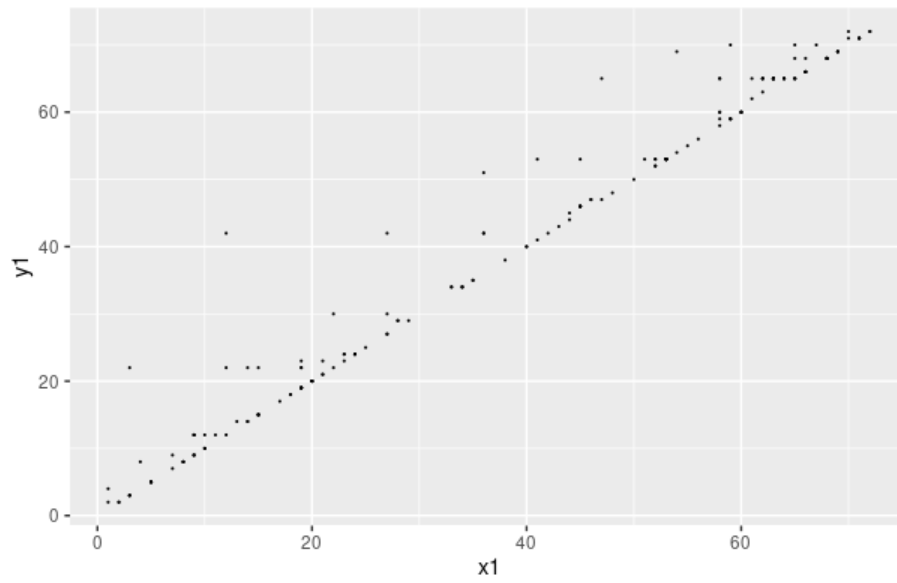


Figure A.162: Unomi

- Observation period: Approximately 80 months
- Total number of bugs: 219
- Percentage of languages (%): Java91.0, CSS6.8, JavaScript0.9, shell0.6, HTML0.5, Groovy0.1, other0.1
- URL for the software overview: <https://unomi.apache.org/>

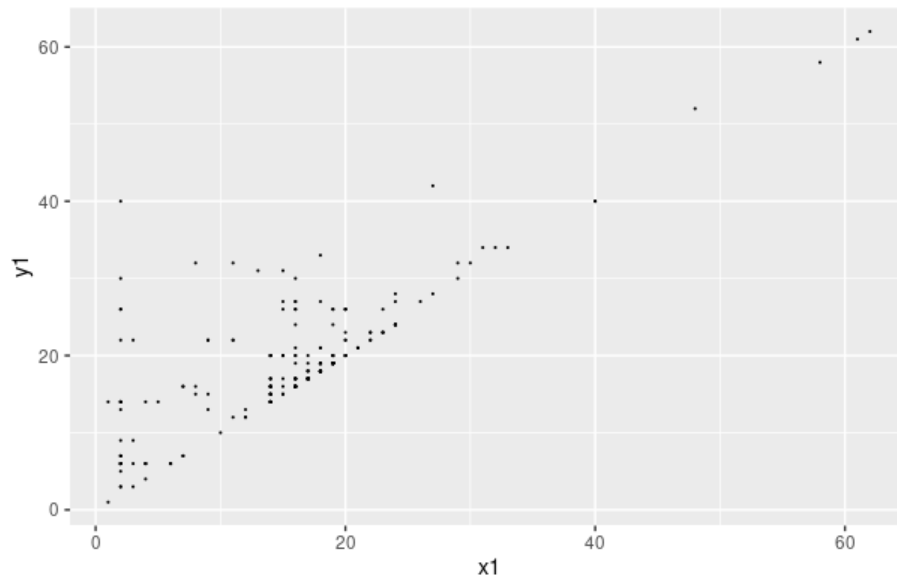


Figure A.163: Usergrid

- Observation period: Approximately 60 months
- Total number of bugs: 350
- Percentage of languages (%): Java66.3, HTML17.5, JavaScript3.5, php2.6, python2.4, other7.7
- URL for the software overview: <https://github.com/apache/usergrid>

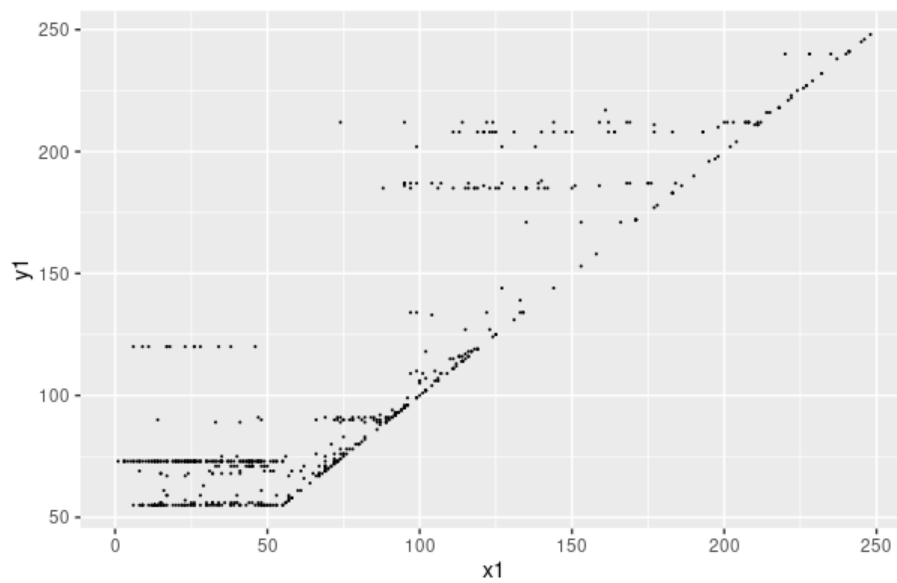


Figure A.164: Velocity

- Observation period: Approximately 250 months
- Total number of bugs: 676
- Percentage of languages (%): Java99.6, shell0.4
- URL for the software overview: <https://velocity.apache.org/>

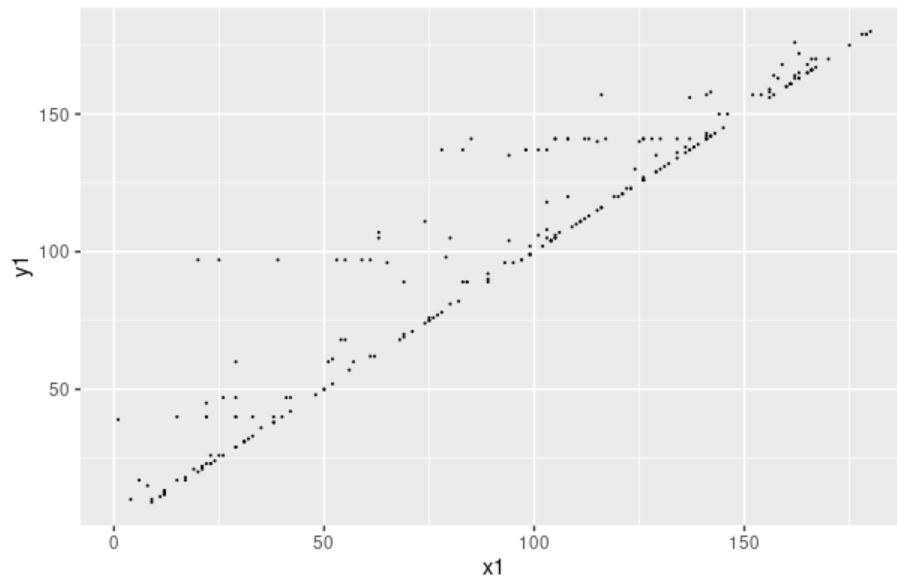


Figure A.165: WebService

- Observation period: Approximately 175 months
- Total number of bugs: 307
- Percentage of languages (%): Java100
- URL for the software overview: <https://ws.apache.org/>

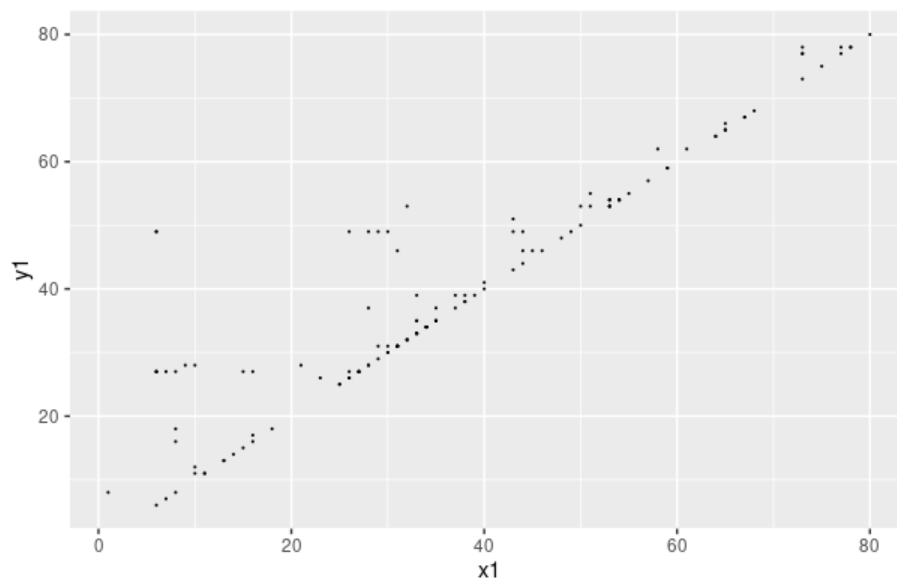
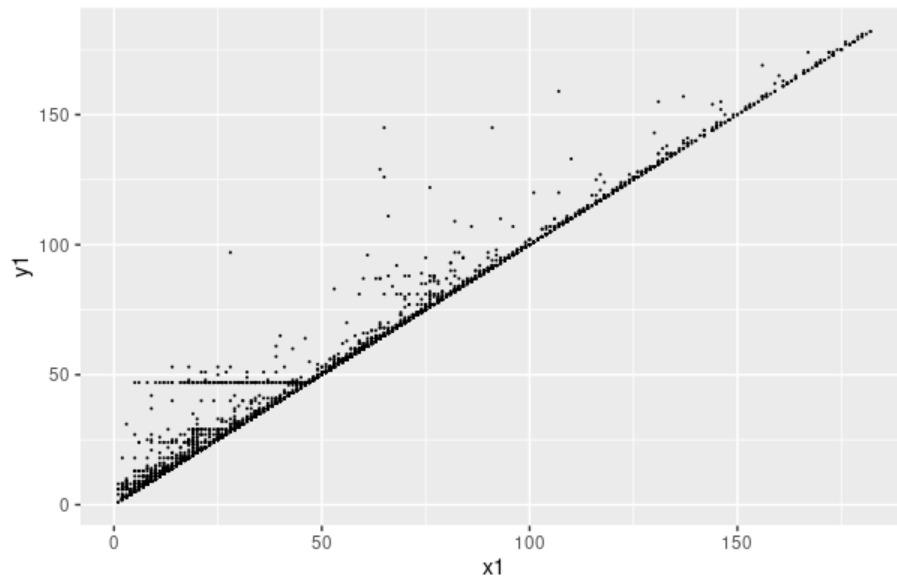


Figure A.166: Whimsy

- Observation period: Approximately 80 months
- Total number of bugs: 190
- Percentage of languages (%): Ruby91.9,JavaScript4.4,HTML2.2,CSS0.7,python0.5,dockerfile0.2,shell0.1
- URL for the software overview: <https://whimsical.apache.org/>

**Figure A.167**

- Observation period: Approximately 200 months
- Total number of bugs: 4170
- Percentage of languages (%): Java88.4, HTML6.5, JavaScript3.7, CSS1.1, shell0.2, other0.1
- URL for the software overview: <https://wicket.apache.org/>

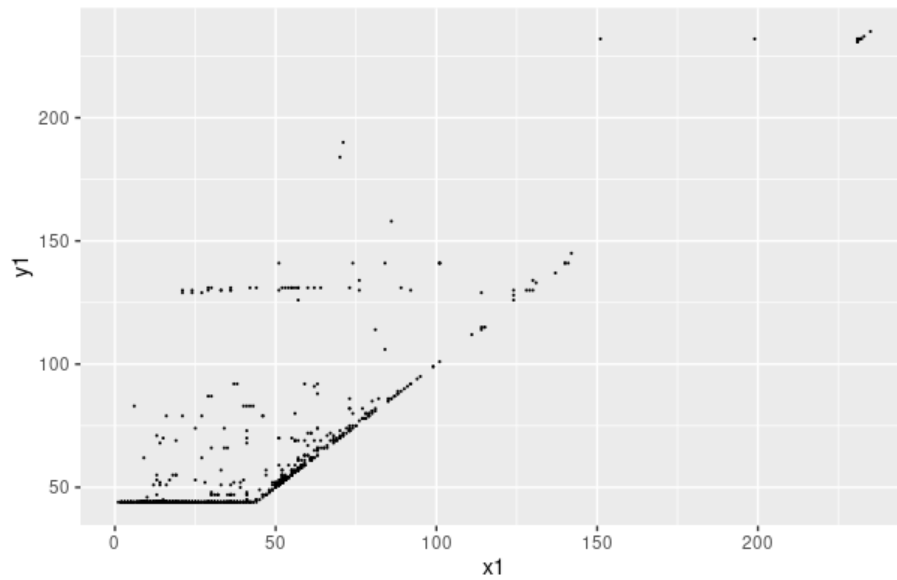


Figure A.168: Xalan

- Observation period: Approximately 250 months
- Total number of bugs: 1605
- Percentage of languages (%): C++98.1, Cmake1.3, other0.6
- URL for the software overview: <https://xalan.apache.org/>

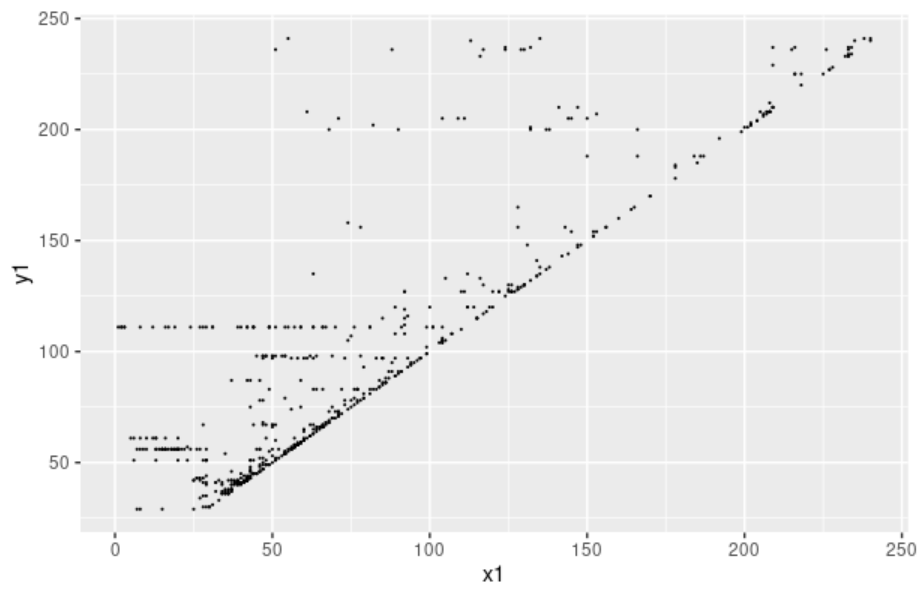


Figure A.169: XML

- Observation period: Approximately 250 months
- Total number of bugs: 807
- Percentage of languages (%): Java100
- URL for the software overview: <https://xerces.apache.org/xml-commons/>

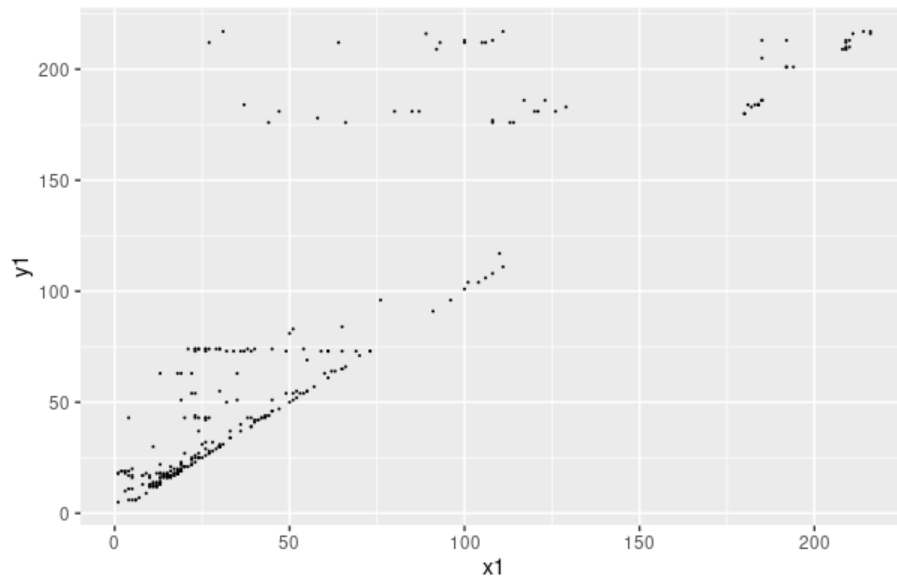


Figure A.170: XMLBeans

- Observation period: Approximately 250 months
- Total number of bugs: 422
- Percentage of languages (%): Java97.5, XSLT1.1, batchfile0.8, shell0.5, other0.1
- URL for the software overview: <https://github.com/apache/xmlbeans>

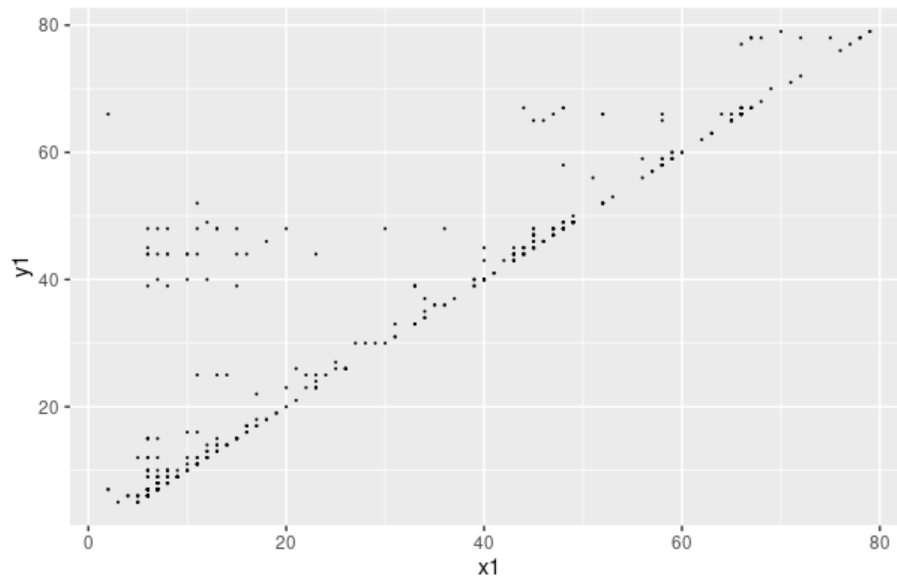


Figure A.171: Yetus

- Observation period: Approximately 80 months
- Total number of bugs: 501
- Percentage of languages (%): shell79.5, python9.9, Java4.9, dockerfile2.2, Ruby1.7, HTML1.7, other0.1
- URL for the software overview: <https://yetus.apache.org/>

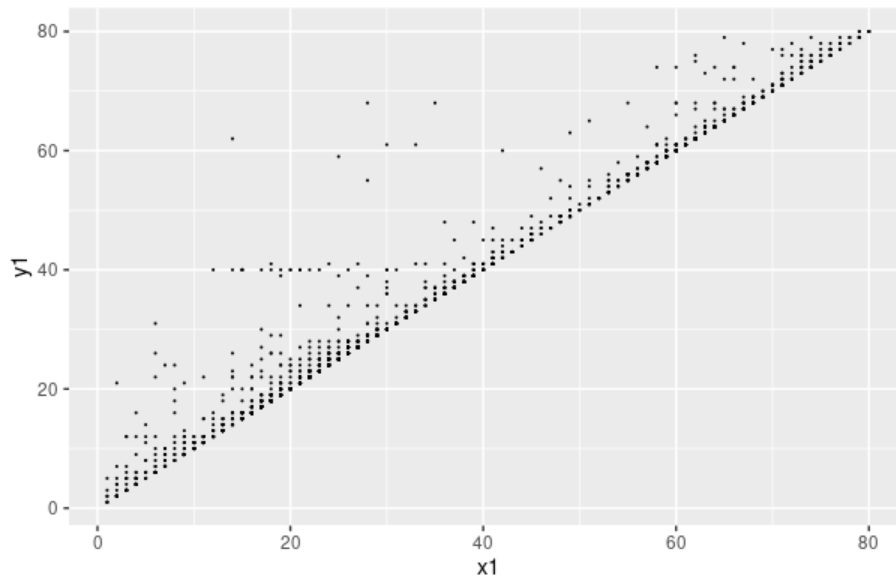


Figure A.172: Zeppelin

- Observation period: Approximately 80 months
- Total number of bugs: 2856
- Percentage of languages (%): Java60.9, Jupyter notebook14.0, JavaScript10.1, TypeScript4.0,HTML3.4, scala3.0, other4.6
- URL for the software overview: <https://zeppelin.apache.org/>

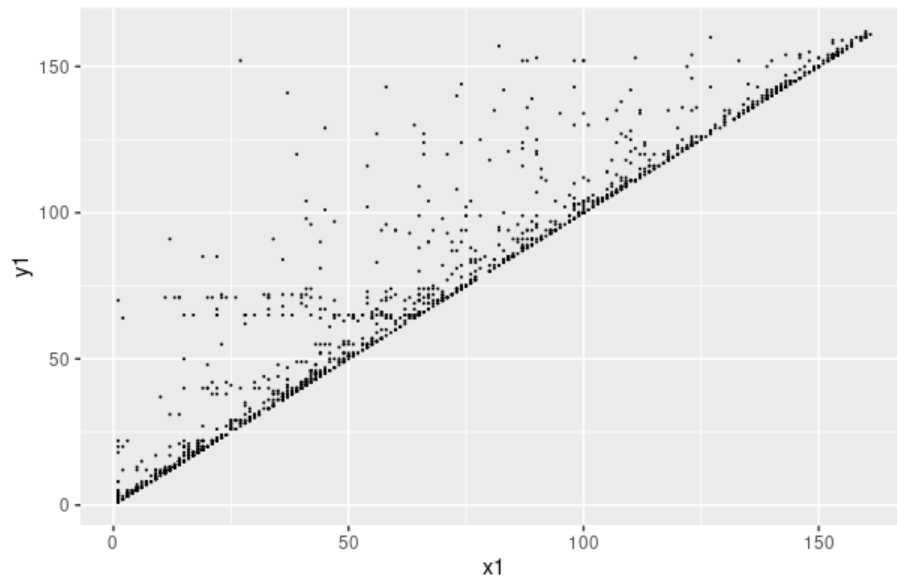


Figure A.173: ZooKeeper

- Observation period: Approximately 150 months
- Total number of bugs: 2233
- Percentage of languages (%): Java75.1, C++7.8, C6.9, JavaScript2.7, python1.8, shell1.2, other4.5
- URL for the software overview: <https://zookeeper.apache.org/>

Appendix B

Programming language

Here are 30 languages that constitute open-source software.

1. Java
2. JavaScript
3. shell
4. freemarker
5. Thrift
6. C++
7. python
8. scala
9. CSS
10. C
11. TypeScript
12. ELANG
13. HTML
14. Ruby
15. dockerfile
16. Groovy
17. XSLT
18. RUST
19. C#
20. GO
21. perl
22. batchfile

- 23. makefile
- 24. richformattext
- 25. roff
- 26. php
- 27. R
- 28. Jupyter notebook
- 29. Cmake
- 30. other

Appendix C

Description texts of OSS

Here, a list of stopwords commonly removed during natural language processing is presented.

1. “i”
2. “me”
3. “my”
4. “myself”
5. “we”
6. “our”
7. “ours”
8. “ourselves”
9. “you”
10. “your”
11. “yours”
12. “yourself”
13. “yourselves”
14. “he”
15. “him”
16. “his”
17. “himself”
18. “she”
19. “her”
20. “hers”
21. “herself”

22. “it”
23. “its”
24. “itself”
25. “they”
26. “them”
27. “their”
28. “theirs”
29. “themselves”
30. “what”
31. “which”
32. “who”
33. “whom”
34. “this”
35. “that”
36. “these”
37. “those”
38. “am”
39. “is”
40. “are”
41. “was”
42. “were”
43. “be”
44. “been”
45. “being”
46. “have”
47. “has”
48. “had”
49. “having”
50. “do”
51. “does”
52. “did”
53. “doing”
54. “would”

55. “should”
56. “could”
57. “ought”
58. “i’m”
59. “you’re”
60. “he’s”
61. “she’s”
62. “it’s”
63. “we’re”
64. “they’re”
65. “i’ve”
66. “you’ve”
67. “we’ve”
68. “they’ve”
69. “i’d”
70. “you’d”
71. “he’d”
72. “she’d”
73. “we’d”
74. “they’d”
75. “i’ll”
76. “you’ll”
77. “he’ll”
78. “she’ll”
79. “we’ll”
80. “they’ll”
81. “isn’t”
82. “aren’t”
83. “wasn’t”
84. “weren’t”
85. “hasn’t”
86. “haven’t”
87. “hadn’t”

88. “doesn’t”
89. “don’t”
90. “didn’t”
91. “won’t”
92. “wouldn’t”
93. “shan’t”
94. “shouldn’t”
95. “can’t”
96. “cannot”
97. “couldn’t”
98. “mustn’t”
99. “let’s”
100. “that’s”
101. “who’s”
102. “what’s”
103. “here’s”
104. “there’s”
105. “when’s”
106. “where’s”
107. “why’s”
108. “how’s”
109. “a”
110. “an”
111. “the”
112. “and”
113. “but”
114. “if”
115. “or”
116. “because”
117. “as”
118. “until”
119. “while”
120. “of”

121. “at”
122. “by”
123. “for”
124. “with”
125. “about”
126. “against”
127. “between”
128. “into”
129. “through”
130. “during”
131. “before”
132. “after”
133. “above”
134. “below”
135. “to”
136. “from”
137. “up”
138. “down”
139. “in”
140. “out”
141. “on”
142. “off”
143. “over”
144. “under”
145. “again”
146. “further”
147. “then”
148. “once”
149. “here”
150. “there”
151. “when”
152. “where”
153. “why”

- 154. “how”
- 155. “all”
- 156. “any”
- 157. “both”
- 158. “each”
- 159. “few”
- 160. “more”
- 161. “most”
- 162. “other”
- 163. “some”
- 164. “such”
- 165. “no”
- 166. “nor”
- 167. “not”
- 168. “only”
- 169. “own”
- 170. “same”
- 171. “so”
- 172. “than”
- 173. “too”
- 174. “very”

Bibliography

- [1] H. Okamura and T. Dohi, A generalized bivariate modeling framework of fault detection and correction processes, IEEE 28th Intl. Symp. on Soft. Reliab. Eng. (ISSRE 2017), 35–45, 2017.
- [2] G. Kusano, K. Fukumizu, and Y. Hiraoka, "Persistence weighted Gaussian kernel for topological data analysis," in Intl. Conf. on Machine Learning, 2004–2013, 2016.
- [3] The APACHE SOFTWARE FOUNDATION <https://www.apache.org/>
- [4] Kikuchi, Ueda, "Investigation of Activity Information for Early Success/-Failure Prediction in OSS Projects," IEICE Transactions on Information and Systems. KBSE2022-69, 2023.
- [5] Higashimoto, Kuramoto, Saito, Imura, Kondo, Kamei, Ubayashi, "Application of Survival Time Analysis to Reduce Adoption Risks in Open-Source Software (OSS)," IEICE Transactions on Information and Systems. KBSE2022-61, 2023.