UNIVERSITY of INFORMATION TECHNOLOGY

Software Defect Prediction Analysis

By

Business Information System [Group 2]

Group Members

Hsu Wai Htet	4BIS - 909
Aye Myat Mon	4BIS - 935
Kaung Min Htet	4BIS - 948
Thet Eain San	4BIS - 955
Ingyin Khin	4BIS - 1002
Nang Thiri San	4BIS - 1009
Yoon Myat Noe	4BIS - 1010
Tun Zarni Aung	4BIS - 1155
Aye Myat Thu	4BIS - 1170
La Wunn Cho	4BIS - 1179

CST - 42315 : Data Analysis & Management

September, 2023

1. Abstract

The software defect management process plays a pivotal role in ensuring the quality and reliability of software products. An effective defect management process encompasses defect identification, documentation, prioritization, tracking, resolution, verification, and analysis. By employing a structured approach, software development teams can efficiently manage defects throughout the software development lifecycle, from initial coding to final release. It highlights the significance of defect management in maintaining software integrity and outlines the steps involved in the process. Moreover, it emphasizes the importance of defect analysis and reporting to enhance the development process continually. A well-implemented defect management process not only leads to reduced post-release issues but also fosters collaboration among development, testing, and project management teams. As software systems become increasingly complex, a robust defect management process becomes indispensable for delivering software products that meet user expectations and industry standards.

Software defect management, also known as bug tracking or issue tracking, is the process of identifying, documenting, prioritizing, tracking, and resolving software defects or issues that arise during the development and testing phases of a software project. Defects can include software bugs, errors, vulnerabilities, and other unexpected behaviours that impact the functionality, performance, or security of the software.

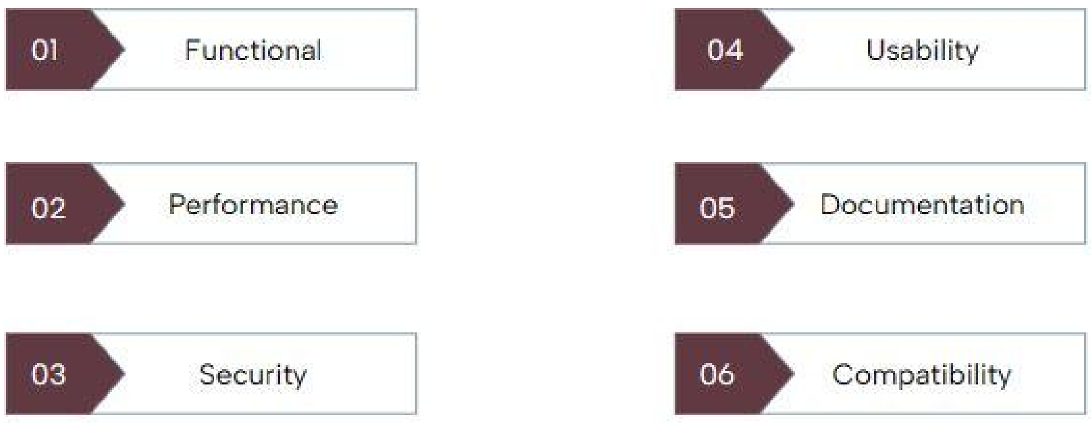


Figure : Types of software defects

Objective

The goal of defect management is to ensure that software is of high quality and functions as intended by minimizing the impact of defects on the end-users and maintaining the integrity of the software product. This process is crucial for delivering reliable and efficient software to users.

Reduced Down Time
Improved Software Quality
Enhanced Testing

Figure: Primary Objectives

2. Machine Learning

Machine learning is a subset of artificial intelligence (AI) that empowers computers to learn from data, recognize patterns, and make decisions or predictions without being explicitly programmed, enhancing their performance over time, focuses on developing algorithms and models that allow computers to learn from data and make predictions or decisions without being explicitly programmed for each specific task. Machine learning algorithms enable computers to identify patterns, make predictions, and take actions based on examples and data inputs. Machine learning has a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, autonomous vehicles, fraud detection, medical diagnosis, and much more. It has revolutionized industries by automating tasks and providing insights that were previously difficult or time-consuming to obtain. In the following, there will be detailed processes about machine learning.

- Data: the foundation of machine learning. It includes the information used to train, validate, and test machine learning models. Data can be structured (tabular data, databases) or unstructured (text, images, audio), and the quality and quantity of data play a significant role in the success of a machine learning project.
- Model: the algorithm or mathematical representation that is trained on the data to learn patterns and relationships. The model transforms inputs (features) into outputs

(predictions or classifications). Different types of models are used for various types of tasks, such as regression for predicting continuous values, classification for categorizing data into classes, and clustering for finding patterns in data.

- Training: During the training phase, the model learns from the provided data. It adjusts its internal parameters based on the patterns and relationships in the training data to make accurate predictions or classifications.
- Supervised Vs Unsupervised Learning: Supervised and unsupervised data mining are two fundamental approaches in the field of data mining, which is the process of discovering patterns, relationships, or valuable insights from large datasets. These techniques are used in various domains, such as machine learning, artificial intelligence, and business intelligence, to extract valuable information from data.
- Reinforcement Learning: In reinforcement learning, an agent learns to interact with an environment to maximize a reward signal. The agent takes actions and receives feedback in the form of rewards or penalties. Over time, it learns to take actions that lead to the highest cumulative reward.
- Evaluation and Testing: After training, the model is evaluated on new, unseen data to assess its performance. This helps determine how well the model generalizes to new examples and whether it's ready for real-world use.
- Deployment: Once a model is trained and validated, it can be deployed in real-world applications to make predictions or decisions based on new data.

To handle large volumes of data, the KDD process, "Knowledge Discovery Process in Databases," is a comprehensive and iterative framework used for extracting useful knowledge and insights from massive data. It encompasses various stages to transform raw data into valuable information, knowledge, and actionable insights. It often involves iteration and refinement of the various stages as new insights are gained or as project requirements evolve. The process emphasizes transforming raw data into actionable knowledge and insights, enabling informed decision-making and enhancing the value derived from data.

The KDD process is widely used in machine learning projects. There will be presented the key stages of the KDD process below.

- To understand the problem and data: In this initial phase, the problem domain and the data are thoroughly studied. Project objectives are defined, and data sources are identified and explored. This step involves getting a clear understanding of the business or research goals and the nature of the data available.

- Data Cleaning and preprocessing: Raw data often contain errors, missing values, outliers, and inconsistencies. This step involves cleaning the data to ensure its quality and reliability. Data preprocessing includes tasks like removing duplicates, handling missing values, and transforming data into a suitable format for analysis.
- Data Integration: When dealing with multiple data sources, data integration aims to combine data from various sources into a unified format. This process helps create a more comprehensive dataset for analysis.
- Data Selection & Transformation: Relevant features and attributes are selected for analysis while irrelevant or redundant ones are removed. Data transformation involves converting data into a suitable format and scale, such as normalization and standardization.
- Data Mining: This stage involves applying various data mining techniques, including machine learning algorithms, to extract patterns, trends, and relationships from the pre-processed data. This is where the core analysis takes place, and models are built to identify hidden insights.
- Pattern Evaluation: the discovered patterns or models are evaluated against the project's objectives. The quality and usefulness of the patterns are assessed to determine their value in addressing the initial problem.
- Knowledge Representation: The knowledge gained from data mining is typically represented in a more understandable and actionable form. This could include visualization, summaries, or reports that convey the insights to stakeholders.
- Interpretation & Evaluation: he results of the analysis are interpreted to gain insights and derive actionable conclusions. This involves understanding the implications of the discovered patterns in the context of the problem domain.
- Deployment & Application: If the insights are deemed valuable, they are deployed for real-world use. This might involve integrating the insights into decision-making processes, systems, or applications.
- Monitoring & Maintenance: Once deployed, the system is continuously monitored to ensure that the insights remain relevant and effective. Regular updates and maintenance might be necessary to adapt to changing data patterns or requirements.

Here, Supervised learning and unsupervised learning will be explained in detail.

Supervised learning is a machine learning approach that's defined by its use of labelled datasets. These datasets are designed to train or "supervise" algorithms into classifying data or predicting outcomes accurately. The goal of supervised learning is to learn a mapping function from the input variables to the output variables, so that given new, unseen data points, the algorithm can predict their corresponding output labels or values. The training process involves providing the algorithm with examples of input-output pairs, allowing it to learn the underlying patterns and relationships. The model's performance is then evaluated on a separate test dataset to measure its ability to generalize to new, unseen data accurately. The most common techniques are:

- Classification: Assigning inputs to predefined categories or classes. Accurately assign test data into specific categories, such as separating apples from oranges. Grouping the patients based on their records.
- Regression: Predicting continuous numerical values. Regression models are helpful for predicting numerical values based on different data points, identify distribution, relationship trends based on available data, such as sales revenue projections for a given business.
- Recommender systems: Providing personalized recommendations based on user preferences.

Unsupervised learning uses machine learning algorithms to analyse and cluster unlabelled data sets. These algorithms discover hidden patterns in data without the need for human intervention. An unlabelled dataset and tasked with finding patterns or structures within the data on its own. Unlike supervised learning, there are no predefined output labels or target values during the training process. The most common techniques are:

- Clustering: Grouping similar data points together based on their intrinsic characteristics or similarities.
- Association: Identifying interesting relationships or associations between different variables or items within the dataset. This is often used in market basket analysis to find frequently co-occurring items in transactions. Finding frequent Patterns "Customers Who Bought This Item Also Bought" recommendations.
- Customer segmentation: Grouping customers based on similar behaviors or attributes.

- Image segmentation: Dividing an image into meaningful segments or regions.
- Anomaly detection: Identifying unusual patterns or outliers in the data.

In Software Defect Prediction, it can be clear that a supervised learning task is applied. In supervised learning, the algorithm is trained on a labeled dataset, where each data point is associated with a known outcome. The algorithm learns to make predictions based on the input features and their corresponding labels. Supervised learning models commonly used for software defect prediction include logistic regression, decision trees, random forests, support vector machines, and neural networks. These models are trained on historical data with known outcomes, and then they can be applied to new data to predict the likelihood of defects.

In the context of software defect prediction:

- Input Features: These features are derived from various aspects of the software development process, codebase, historical defect data, and other relevant attributes. These could include code complexity metrics, code churn, developer expertise, historical defect reports, etc.
- Labels: The labels in this case represent whether a specific module of code was found to be defective or not (binary classification). The labels are determined based on past defect reports and testing outcomes.

The goal of the supervised learning model in software defect prediction is to learn the patterns and relationships between the input features and the labels so that it can accurately predict whether a new, unseen module of code is likely to contain defects.

Information Retrieval

Information retrieval in the context of software defect prediction involves gathering and analyzing relevant data to make informed decisions about the likelihood of defects in software projects.

- Define Objectives and Scope
- Data Collection
- Data Preprocessing
- Feature Selection/Extraction
- Data Splitting

- Model Building
- Model Evaluation
- Model Tuning
- Interpretation and Visualization
- Deployment and Monitoring
- Feedback Loop
- Documentation

The above steps provide a structured approach to information retrieval and modeling for software defect prediction, helping organizations identify and mitigate potential issues in their software projects.

3. Information Organisation

Defects in software, also known as bugs or software errors, are issues or flaws in a computer program's code or functionality. These defects can occur at various stages of the software development process and can have different consequences in the organization.

Bugs
Functionality
Issues
Errors
Flaws

Figure : Defects in software

3.1 Defect Prediction

Software defect prediction using big data involves leveraging large volumes of software-related data, such as code metrics, historical defect reports, developer activity logs, and more, to develop predictive models that can identify potential defects in software systems. Here's a high-level overview of the process and some key considerations:

- Data Collection
- Feature Extraction
- Data Preprocessing
- Training and Validation
- Feature Selection

3.2 Advantages of Defect Prediction

- Improved Software Quality
- Enhanced Testing
- Reduced Downtime and Customer Complaints

4. Theory Background

1. **Regression**

   Regression analysis is a statistical technique used in predictive modeling to analyze the relationship between a dependent variable (also called the target or response variable) and one or more independent variables (also called predictors or features). Its primary goal is to understand how changes in the independent variables are associated with changes in the dependent variable, and then use this understanding to make predictions.

2. **Time series Analysis**

   Time series analysis is a statistical technique used to analyze and interpret data that is collected or recorded over time. This type of data is referred to as a time series, and it can be used to make forecasts, identify patterns, and extract meaningful insights. Time series analysis is valuable in various fields, including finance, economics, epidemiology, environmental science, and more.

3. **Machine learning algorithm**
   Machine learning algorithms are computational techniques that enable computers to learn from and make predictions or decisions based on data. These algorithms play a central role in artificial intelligence (AI) and are used for a wide range of tasks, including classification, regression, clustering, recommendation, and more.

4. **Neural networks and deep learning**
   Neural networks are a fundamental component of deep learning, a subfield of machine learning. Deep learning models are designed to mimic the way the human brain works, with interconnected artificial neurons arranged in layers. These models excel at automatically learning representations from data, making them particularly well-suited for tasks like image recognition, natural language processing, and more.

5. **K-Nearest neighbor**
   K-Nearest Neighbors (KNN) is a simple and intuitive machine learning algorithm used for both classification and regression tasks. It's a non-parametric, instance-based algorithm that makes predictions based on the majority class or the average of the k-nearest data points in the feature space.

6. **Ensemble methods**
   Ensemble methods are machine learning techniques that combine multiple individual models (often referred to as base models or weak learners) to create a more powerful and robust model. The goal of ensemble methods is to improve predictive performance, reduce overfitting, and enhance model generalization.

7. **Survival analysis**
   Survival analysis, also known as time-to-event analysis or event history analysis, is a statistical technique used to analyze and model the time until an event of interest occurs. This technique is commonly used in medical research, social sciences, engineering, and various other fields to study events such as death, disease recurrence, equipment failure, customer churn, and more. Survival analysis takes into account

censoring, where some individuals do not experience the event during the study period or are lost to follow-up.

8. **Bayesian network**

   Bayesian regression and linear regression are two distinct approaches to modeling relationships between variables in statistics and machine learning. While both techniques are used for regression analysis, they have different foundations and assumptions.

   Bayesian Regression:

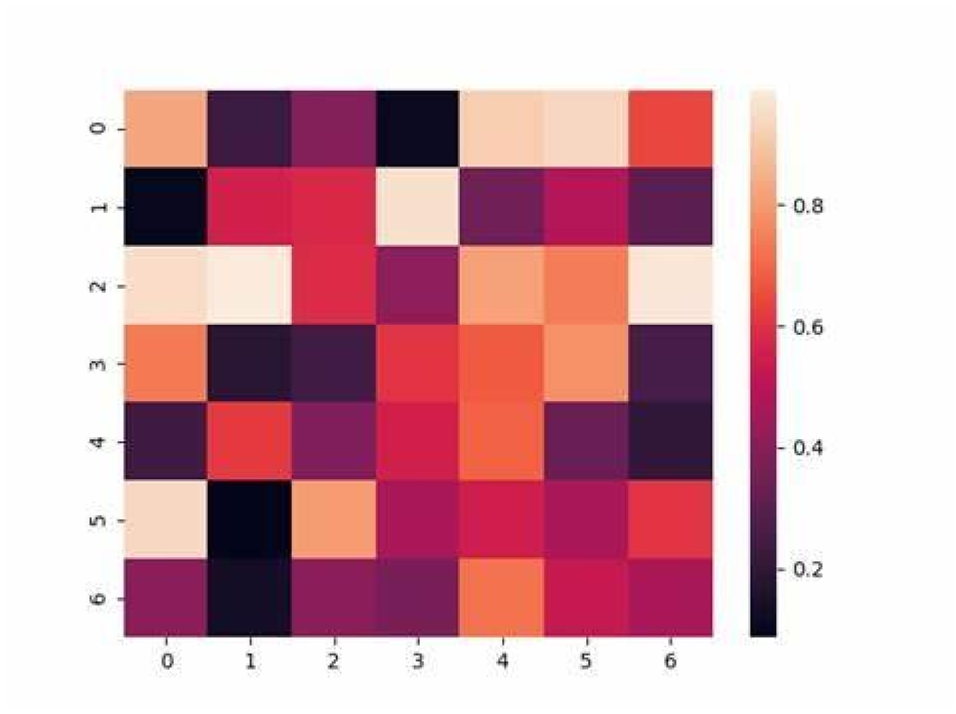
   Bayesian regression is a probabilistic approach to regression analysis that incorporates Bayesian principles. It models the uncertainty associated with regression coefficients and predictions by treating them as probability distributions rather than fixed values. Bayesian regression can be applied to linear models as well as non-linear models.

4.1 Visualization Techniques

1. **Heatmaps**

   A heatmap is a data visualization technique used to represent data in a tabular format as a grid of colored squares, where the color intensity of each square corresponds to the value of the data it represents. Heatmaps are particularly useful for visualizing the magnitude of values in a matrix or 2D dataset.

   To create a heatmap, you typically assign a color scale to the values in your dataset, with colors ranging from a minimum value to a maximum value. The color intensity in each square cell reflects the magnitude of the corresponding value, making it easier to identify patterns and trends in the data.

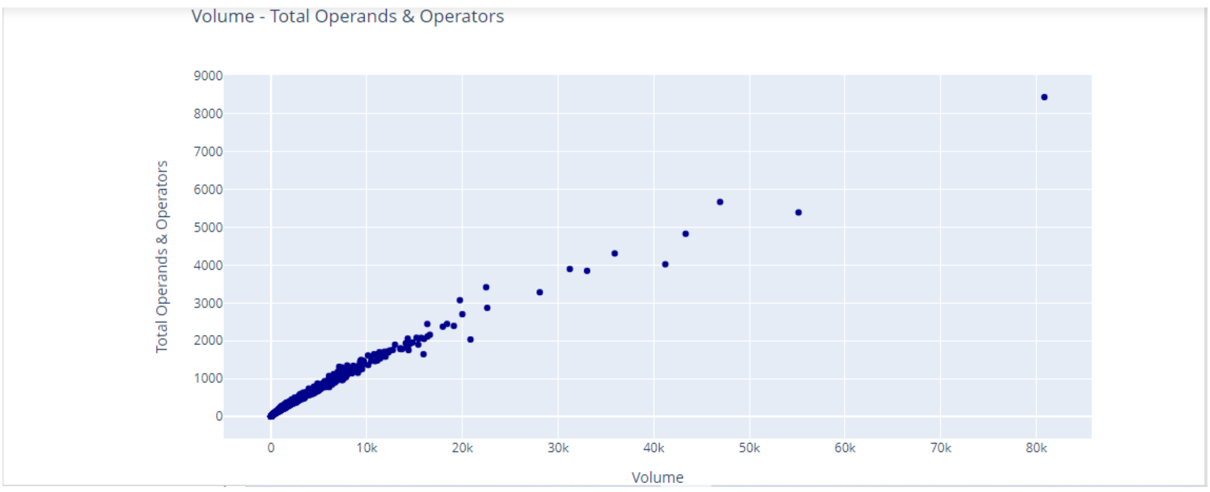


2. **Scatter plot**

   A scatter plot is a type of data visualization that displays individual data points as individual dots on a two-dimensional graph. Each dot represents a single observation, and the position of the dot on the graph is determined by the values of two variables, one plotted on the x-axis and the other on the y-axis. Scatter plots are widely used in data analysis and visualization to explore relationships between two continuous variables and to identify patterns, trends, or outliers in the data.

   If you are plotting the heights and weights of individuals, a scatter plot could reveal that there is a positive correlation, indicating that as height increases, weight tends to increase. Outliers in the plot might represent individuals who are exceptionally tall or heavy.

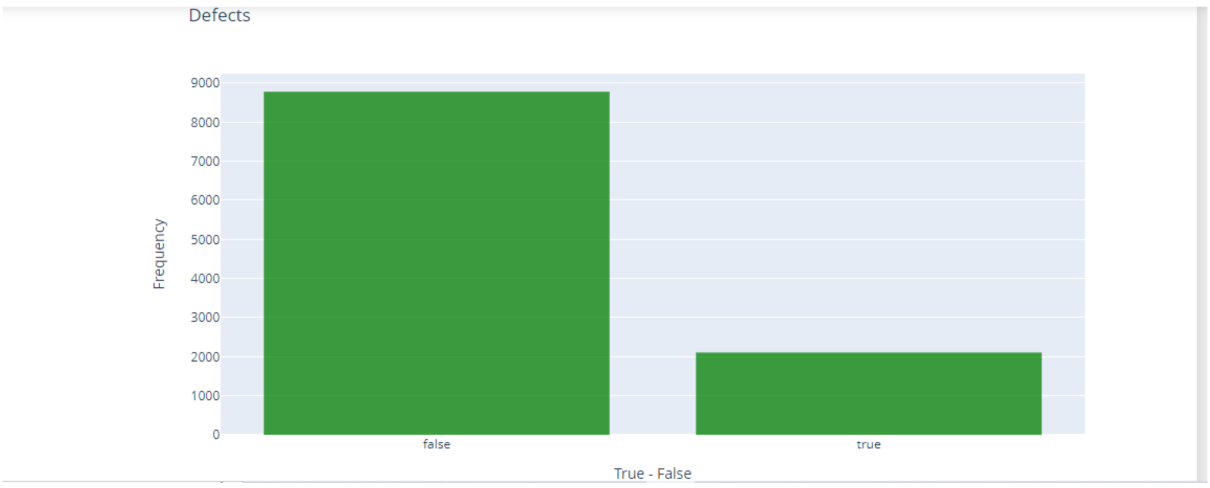
   Scatter plots are commonly created using data visualization libraries like Matplotlib (Python), ggplot2 (R), or Excel. They are a fundamental tool in data analysis, helping analysts and data scientists gain insights into their data and make informed decisions based on patterns and relationships observed in the plot.



3. Histogram

A histogram is a graphical representation of the distribution of a dataset. It provides a visual summary of the underlying frequency distribution of a set of continuous or discrete data. Histograms are commonly used in data analysis and statistics to understand the shape, central tendency, and spread of data.

Histograms are a fundamental tool in exploratory data analysis (EDA) and are often the first step in understanding the distribution of a dataset. They provide valuable insights into the nature of the data, helping researchers and analysts make informed decisions and identify potential areas for further investigation or analysis.

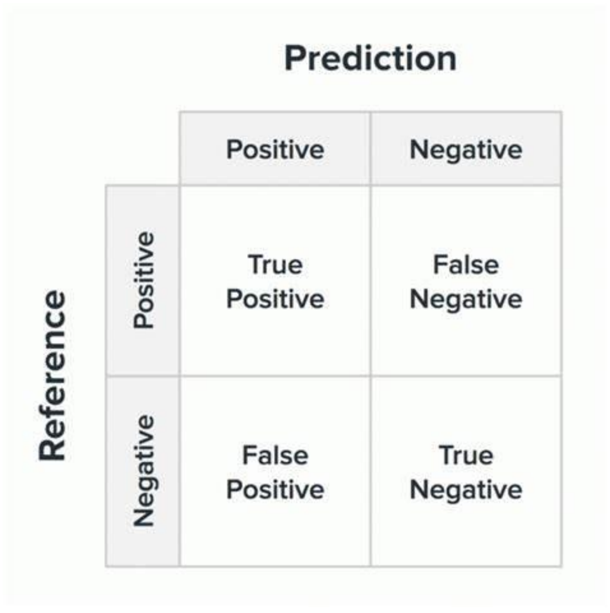


4. **Confusion matrix**

A confusion matrix is a table used in machine learning and classification tasks to evaluate the performance of a classification model, particularly in binary and multi-class classification problems. It provides a summary of how well a model's predictions align with the actual class labels in the dataset. A confusion matrix is also known as an error matrix.

These metrics provide a comprehensive view of the model's performance, especially in situations where false positives and false negatives have different consequences or costs.

In multi-class classification, a confusion matrix can be extended to represent the relationships between multiple classes, providing a more detailed assessment of the model's performance across all classes.

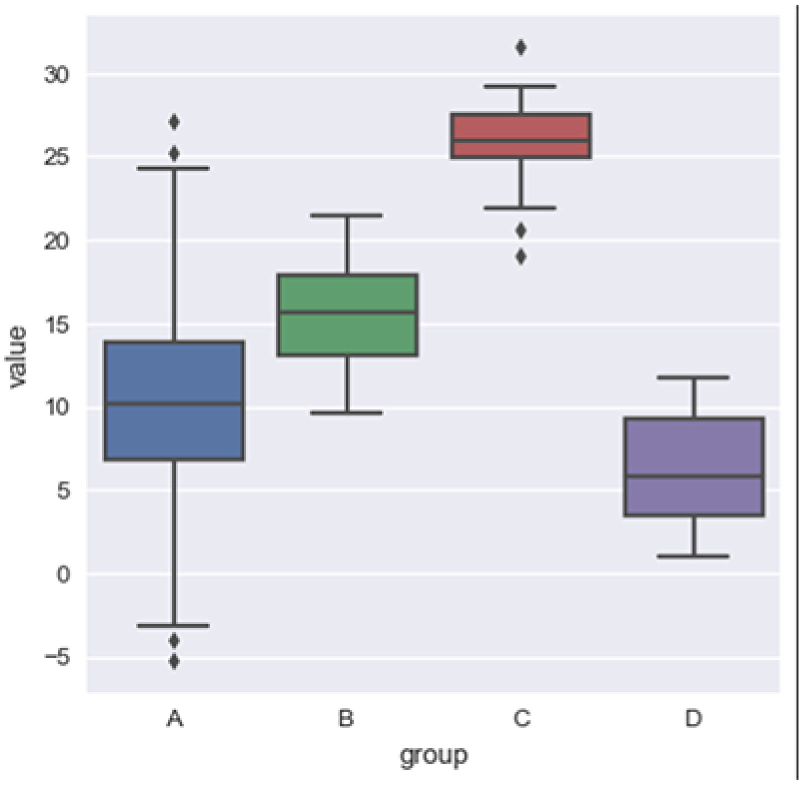


		Prediction	
		Positive	Negative
Reference	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

5. **Box plot**

A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset, particularly useful for visualizing the spread and central tendency of data. It displays the five-number summary of a dataset: the minimum, first quartile (Q1), median (second quartile or Q2), third quartile (Q3), and maximum. Box plots are particularly effective for identifying outliers and comparing the distributions of multiple datasets.

Box plots are a common tool in exploratory data analysis (EDA) and are frequently used in various fields, including statistics, data science, and data visualization, to gain insights into the distribution of data and to make informed decisions based on data characteristics.



4.2 Data Prepossessing

Data preprocessing is a crucial preparatory phase in data analysis and machine learning that involves a series of operations to enhance the quality and usability of raw data. This multifaceted process encompasses tasks such as handling missing values through imputation or removal, detecting and addressing duplicates, identifying and mitigating outliers, scaling or normalizing numerical features for uniformity, and encoding categorical variables for numerical compatibility. Furthermore, it may entail feature engineering, including creating new informative attributes or transforming existing ones, and data reduction techniques like dimensionality reduction to reduce complexity. Ultimately, data preprocessing ensures that the dataset is clean, well-structured, and ready for further analysis or model training, significantly impacting the accuracy and robustness of downstream tasks.

The quality of your preprocessing can significantly impact the accuracy and effectiveness of your data analysis or machine learning models. Here are key aspects of data preprocessing:

- Data Collection
- Data Cleaning
- Data Transformation
- Data Reduction
- Data Integration
- Data Splitting
- Data Scaling
- Data Visualization
- Data Quality Assurance
- Documentation

Basic steps in this projects:

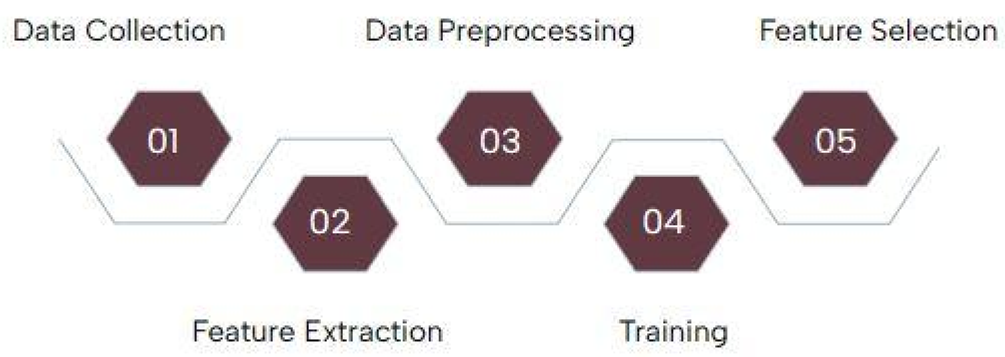


Figure: Major phases in the project

Data preprocessing is often an iterative process, and the specific steps and techniques used can vary depending on the nature of the data and the goals of the analysis or modeling task. Properly preprocessed data sets a strong foundation for robust and reliable data analysis and machine learning outcomes.

4.3 Prediction Steps

Data prediction, a fundamental component of machine learning and predictive analytics, entails utilizing trained models to make informed forecasts or classifications based on input data. These forecasts can span various domains, from predicting future values in time series data to categorizing items, diagnosing diseases, recommending products, optimizing processes, or detecting anomalies. The prediction process typically involves selecting an appropriate predictive model, preprocessing input data to align with model requirements, feeding the data into the model to generate predictions, evaluating prediction performance using relevant metrics, and potentially deploying the model in real-world applications to drive decision-making. The overarching aim of data prediction is to leverage historical patterns and relationships within data to make informed, data-driven decisions, automate tasks, and gain actionable insights that drive improved outcomes in various fields.

1. **Define the goals**

Data prediction encompasses a range of objectives, including forecasting future trends, classifying data into categories, estimating numeric values, detecting anomalies, making recommendations, optimizing processes, segmenting customers, recognizing patterns, assessing risks, diagnosing issues, ensuring quality, forecasting demand, retaining customers, conserving resources, and more. These goals are achieved by leveraging historical data patterns and relationships to inform decision-making, automate tasks, and gain valuable insights across diverse domains and applications.

In a software defect management system, the primary goals of data prediction are to proactively identify and address software defects by forecasting potential issues, improving software quality through efficient resource allocation, assessing risks associated with releases, enhancing process efficiency, prioritizing critical defects, analyzing historical trends for preventive measures, identifying root causes, leveraging user feedback for issue prediction, and automating aspects of defect management. These objectives collectively aim to minimize defects, reduce development costs, enhance user satisfaction, and optimize software quality and reliability.

2. Data collection

Data collection in a software defect management system involves systematically gathering information related to software defects and issues throughout the software development lifecycle. This process includes recording details such as defect descriptions, severity, affected components, timestamps, and the individuals involved in reporting and addressing issues. Data sources typically include bug tracking systems, version control repositories, continuous integration logs, user feedback channels, and automated testing tools. Additionally, user-reported defects and feedback are invaluable sources of data, providing insights into real-world usage and user experiences. The collected data serves as the foundation for defect analysis, prediction, and informed decision-making, helping organizations improve software quality and reliability.

3. Data exploration

Data exploration in a software defect management system involves a systematic examination of the collected defect data to uncover patterns, insights, and trends. This process includes descriptive statistics, visualization, and data summarization techniques to understand the distribution of defects, common sources of issues, and their impact on software quality. Exploratory data analysis helps identify recurring patterns, such as frequently occurring defects in specific code modules or at certain stages of development, aiding in informed decision-making. Additionally, it assists in identifying outliers, potential root causes, and areas requiring focused attention, ultimately contributing to more effective defect prioritization, resource allocation, and process improvements in software development and maintenance.

4. Data preparation

Data preparation in a software defect management system involves several key steps to ensure that defect data is ready for analysis and decision-making. It includes data cleaning, which entails handling missing or inconsistent values, removing duplicates, and standardizing data formats. Data transformation involves encoding categorical variables, scaling numerical features, and potentially creating new derived variables. Additionally, data splitting is often performed to separate data into training and testing sets, ensuring unbiased model evaluation. The goal of data preparation is to have a clean, structured dataset that can be used for various analyses, including defect prediction, trend analysis, and resource allocation, ultimately supporting efforts to improve software quality and reliability.

5. Data Analysis & model building

In a software defect management system, data analysis and model building are critical phases aimed at leveraging historical defect data to improve software quality and reliability. Data analysis involves using statistical techniques and visualization to uncover patterns, trends, and insights within the defect dataset, helping identify key factors contributing to defects. Model building entails the creation of predictive models, such as machine learning algorithms, to forecast potential defects, prioritize critical issues, and allocate resources effectively. These models are trained on historical defect data and utilize features like defect descriptions, code metrics, and development phase information to make informed predictions about future defects. The ultimate goal is to proactively manage defects, enhance software quality, and streamline development processes by harnessing the power of data-driven insights and predictive models.

6. Training the model

Training the model in a software defect management system is a crucial step that involves using historical defect data to build predictive algorithms. This process typically employs machine learning techniques, where the model learns patterns and relationships between various data features, such as defect descriptions, code attributes, and development phase information. The dataset is divided into training and validation subsets, allowing the model to learn from past defects while evaluating its performance on unseen data. Various algorithms, such as decision trees, support vector machines, or neural networks, can be used to create predictive models tailored to the specific objectives of defect management. The goal is to develop a reliable model that can accurately predict potential defects, assess their severity, and aid in resource allocation, ultimately contributing to enhanced software quality and more efficient development processes.

7. Making prediction

Making predictions in a software defect management system involves deploying the trained predictive model to assess and forecast potential defects in newly developed software or code changes. The system feeds relevant input data, such as code attributes, descriptions, and development phase information, into the model. The model, having learned from historical data during training, then generates predictions regarding the likelihood of defects, their severity, or the areas most susceptible to issues. These predictions assist software development teams and managers in proactively identifying and prioritizing defects, enabling more efficient resource allocation and quality improvement efforts. By making informed predictions, the software defect management system enhances the software development process, ultimately leading to higher software quality and reliability.

8. Interpretation & decision

Interpretation and decision-making in a software defect management system involve analyzing the predictions and insights generated by the system to inform actions and strategies. After obtaining defect predictions and severity assessments, software development teams and managers interpret the results, considering factors like business priorities and resource availability. They decide how to prioritize defect fixes, allocate resources effectively, and schedule development efforts accordingly. These decisions impact the software's quality, release schedules, and resource utilization. Additionally, interpretation involves understanding the root causes of defects, which can inform process improvements and preventive measures. By effectively interpreting and acting upon the system's outputs, organizations can enhance software quality, reduce development costs, and improve the overall software development process.

9. Monitoring & adjustment

Monitoring and adjustment in a software defect management system involve the ongoing evaluation of the system's performance and the implementation of improvements. This process includes tracking key performance metrics, such as defect prediction accuracy and the effectiveness of defect prioritization strategies, to ensure that the system continues to deliver value. If the system's predictions or recommendations show room for improvement, adjustments may be made, such as refining the predictive models, updating data sources, or redefining defect severity criteria. Regular monitoring and adjustment are critical to maintaining the system's effectiveness in optimizing software quality, resource allocation, and defect management processes, ensuring that it remains aligned with evolving development needs and objectives.

5. Software Defect Prediction Analysis

5.1 .Naive Bayes algorithm

The Naive Bayes algorithm is a probabilistic machine learning and classification method based on Bayes' theorem with the "naive" assumption of feature independence.

The Naive Bayes algorithm is a simple yet powerful machine learning algorithm that is used for classification tasks.

.The Naive Bayes algorithm is a probabilistic classifier, which means that it assigns a probability to each class for a given set of features. The class with the highest probability is the class that the algorithm predicts the data belongs to.

Here is the definition of the Naive Bayes algorithm in mathematical terms:

$P(c|x) = P(x|c)P(c) / P(x)$

where:

- c is the class label
- x is the feature vector
- $P(c|x)$ is the posterior probability of class c given feature vector x
- $P(x|c)$ is the likelihood of feature vector x given class c
- $P(c)$ is the prior probability of class c
- $P(x)$ is the prior probability of feature vector x

Some of the reasons why we use Naive Bayes algorithm in software defect detection analysis:

- Simple and efficient: The Naive Bayes algorithm is a simple and efficient algorithm that can be easily implemented. This makes it a good choice for software defect detection analysis, which is often a time-consuming and resource-intensive task.
- Robust to noise: The Naive Bayes algorithm is relatively robust to noise, which is common in software defect data. This means that the algorithm can still produce accurate results even if the data is not perfect.
- Applicable to different types of data: The Naive Bayes algorithm can be applied to different types of data, including code metrics, historical defect data, and expert opinions. This makes it a versatile tool that can be used in a variety of software defect detection scenarios.

5.2 .Linear Regression algorithm

Linear regression is a supervised machine learning algorithm that is used to predict a continuous value (known as the dependent variable) based on one or more independent variables. The independent variables are also known as the features or predictors.It is a statistical method or supervised learning technique used in data science and machine learning for predictive analysis.

The linear regression algorithm fits a line to the data points in such a way that the sum of the squared residuals (the distance between the line and the data points) is minimised. The line that minimises the sum of the squared residuals is called the regression line.

The equation of the regression line is:

$$y = mx + b$$

where:

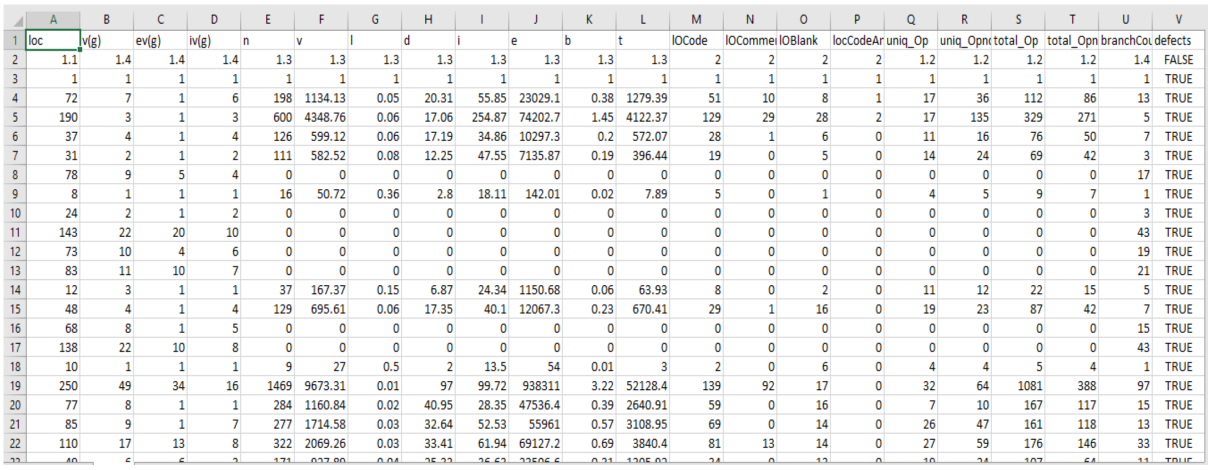
- y is the dependent variable
- m is the slope of the line
- b is the y-intercept

Here is a simpler explanation of why we use linear regression algorithm in software defect detection analysis:

- Linear regression is a simple algorithm that can be used to find a relationship between the number of defects in a software module and a set of features.
- This relationship can be used to predict the number of defects in a new module with similar features.
- Linear regression is a good choice for software defect detection analysis because it is simple, requires less data, and is robust to noise.
- However, linear regression is not perfect and can be sensitive to the choice of features.

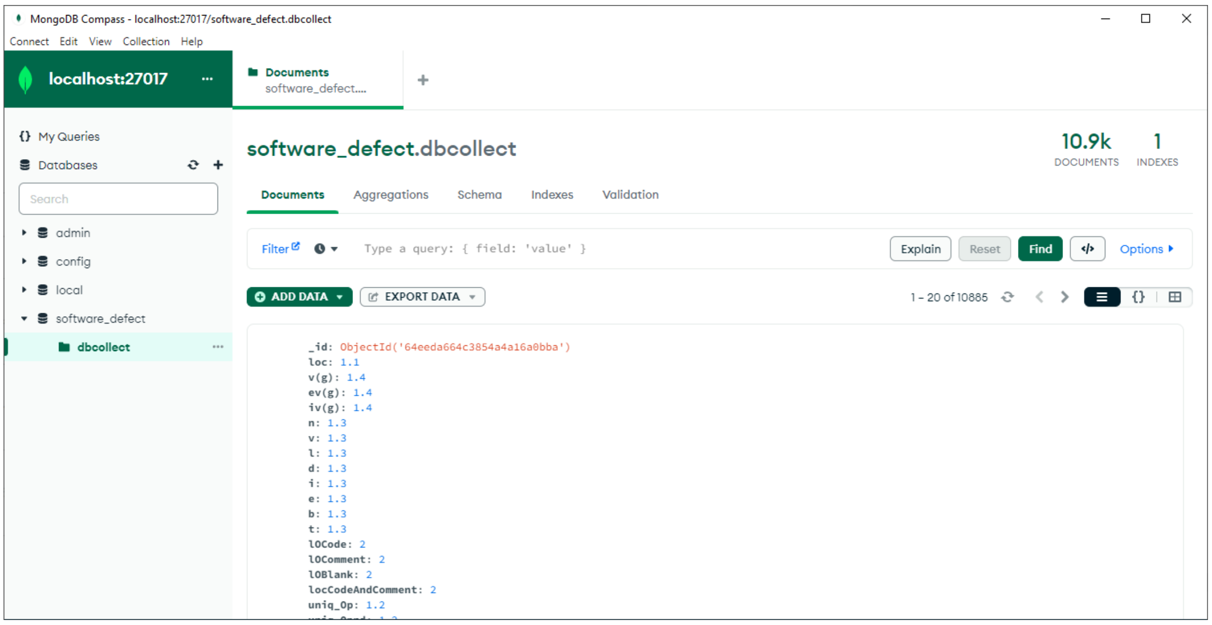
5.3 Training Data and Set

Training Data set in CSV format:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOCommei	lOBlank	locCodeAr	uniq_Op	uniq_Opn	total_Op	total_Opn	branchCou	defects
2	1.1	1.4	1.4	1.4	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2	2	2	2	1.2	1.2	1.2	1.2	1.4	FALSE
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	TRUE
4	72	7	1	6	198	1134.13	0.05	20.31	55.85	23029.1	0.38	1279.39	51	10	8	1	17	36	112	86	13	TRUE
5	190	3	1	3	600	4348.76	0.06	17.06	254.87	74202.7	1.45	4122.37	129	29	28	2	17	135	329	271	5	TRUE
6	37	4	1	4	126	599.12	0.06	17.19	34.86	10297.3	0.2	572.07	28	1	6	0	11	16	76	50	7	TRUE
7	31	2	1	2	111	582.52	0.08	12.25	47.55	7135.87	0.19	396.44	19	0	5	0	14	24	69	42	3	TRUE
8	78	9	5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	TRUE
9	8	1	1	1	16	50.72	0.36	2.8	18.11	142.01	0.02	7.89	5	0	1	0	4	5	9	7	1	TRUE
10	24	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	TRUE
11	143	22	20	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	TRUE
12	73	10	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	TRUE
13	83	11	10	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	TRUE
14	12	3	1	1	37	167.37	0.15	6.87	24.34	1150.68	0.06	63.93	8	0	2	0	11	12	22	15	5	TRUE
15	48	4	1	4	129	695.61	0.06	17.35	40.1	12067.3	0.23	670.41	29	1	16	0	19	23	87	42	7	TRUE
16	68	8	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	TRUE
17	138	22	10	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	TRUE
18	10	1	1	1	9	27	0.5	2	13.5	54	0.01	3	2	0	6	0	4	4	5	4	1	TRUE
19	250	49	34	16	1469	9673.31	0.01	97	99.72	938311	3.22	52128.4	139	92	17	0	32	64	1081	388	97	TRUE
20	77	8	1	1	284	1160.84	0.02	40.95	28.35	47536.4	0.39	2640.91	59	0	16	0	7	10	167	117	15	TRUE
21	85	9	1	7	277	1714.58	0.03	32.64	52.53	55961	0.57	3108.95	69	0	14	0	26	47	161	118	13	TRUE
22	110	17	13	8	322	2069.26	0.03	33.41	61.94	69127.2	0.69	3840.4	81	13	14	0	27	59	176	146	33	TRUE

Actual big data training set(Mongodb data format) for software defect prediction:


MongoDB Compass - localhost:27017/software_defect.dbcollect
Connect Edit View Collection Help
localhost:27017
Documents
software_defect....
My Queries
Databases
Search
admin
config
local
software_defect
dbcollect
software_defect.dbcollect
10.9k DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Filter
Type a query: { field: 'value' }
Explain Reset Find Options
ADD DATA
EXPORT DATA
1 – 20 of 10885

```
_id: ObjectId('64eeda664c3854a4a16a0bba')
loc: 1.1
v(g): 1.4
ev(g): 1.4
iv(g): 1.4
n: 1.3
v: 1.3
l: 1.3
d: 1.3
i: 1.3
e: 1.3
b: 1.3
t: 1.3
lOCode: 2
lOComment: 2
lOBlank: 2
locCodeAndComment: 2
uniq_Op: 1.2
```

Data attributes are as followed to predict:

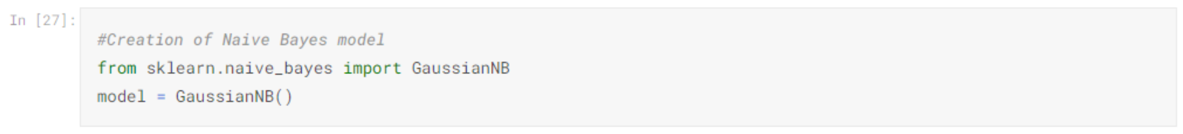
1. loc : numeric % McCabe's line count of code
2. v(g) : numeric % McCabe "cyclomatic complexity"
3. ev(g) : numeric % McCabe "essential complexity"
4. iv(g) : numeric % McCabe "design complexity"
5. n : numeric % Halstead total operators + operands
6. v : numeric % Halstead "volume"
7. l : numeric % Halstead "program length"

8. d : numeric % Halstead "difficulty"
9. i : numeric % Halstead "intelligence"
10. e : numeric % Halstead "effort"
11. b : numeric % Halstead
12. t : numeric % Halstead's time estimator
13. lOCode : numeric % Halstead's line count
14. lOComment : numeric % Halstead's count of lines of comments
15. lOBlank : numeric % Halstead's count of blank lines
16. lOCodeAndComment : numeric
17. uniq_Op : numeric % unique operators
18. uniq_Opnd : numeric % unique operands
19. total_Op : numeric % total operators
20. total_Opnd : numeric % total operands
21. branchCount : numeric % of the flow graph
22. defects : {false,true} % module has/has not one or more reported defects

5.4 Method

Method that we use as the main method for Naive Bayes algorithm is **Gaussian Naive Bayes**.

Gaussian Naive Bayes is a type of Naive Bayes classifier that is used for classification problems where the features are continuous. The Gaussian Naive Bayes classifier assumes that the features are distributed according to a Gaussian distribution, also known as a normal distribution. This means that each feature is normally distributed, and the mean and variance of each feature are independent of the means and variances of the other features.



```
In [27]:
#Creation of Naive Bayes model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

The Gaussian Naive Bayes algorithm can be used to predict the probability of a software module being defective by calculating the posterior probability of the module being defective given the values of its features. The posterior probability is calculated using Bayes' theorem:

P(defect|features) = P(features|defect)P(defect) / P(features)

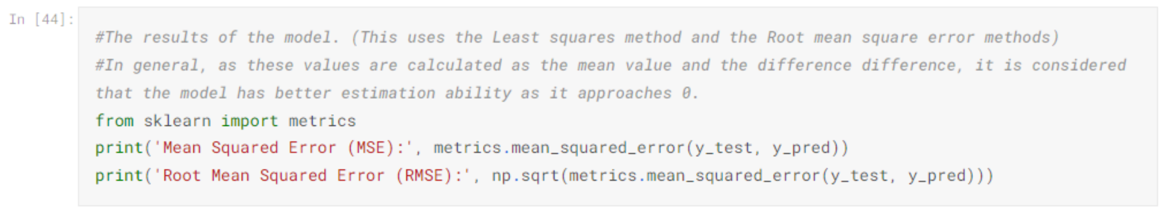
where:

- defect is the label of the software module (defective or not defective)
- features are the features of the software module
- P(defect|features) is the posterior probability of the software module being defective given the values of its features
- P(features|defect) is the likelihood of the features of the software module given that the module is defective
- P(defect) is the prior probability of the software module being defective
- P(features) is the prior probability of the features of the software module

Why Gaussian Naive Bayes is a good choice for software defect prediction:

- It is simple and easy to implement.
- It is efficient to train.
- It can be effective in problems with continuous features.
- It can be effective in problems where the assumption of independence between features is valid.

In Linear regression, we use the **Mean Squared Error(MSE)** method and the **Root Mean Square Error (RMSE)** method.



```
In [44]:
#The results of the model. (This uses the Least squares method and the Root mean square error methods)
#In general, as these values are calculated as the mean value and the difference difference, it is considered
that the model has better estimation ability as it approaches 0.
from sklearn import metrics
print('Mean Squared Error (MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Mean Squared Error (MSE) is a statistical measure that quantifies the average squared difference between the predicted values and the actual values in a dataset It is commonly used to evaluate the performance of regression models and assess the accuracy of predictions.

It is also a measure of the average squared difference between predicted values and actual values. It is a popular measure of accuracy in regression analysis.

It's important to note that MSE is always non-negative and not zero, primarily due to randomness or the model's inability to account for additional information that could lead to more accurate predictions.

The formula to calculate MSE is as follows

$MSE = (1/n) * \Sigma(yi - \hat{y}i)^2$

Where:

n is the number of observations in the dataset

yi is the ith observed value

ŷi is the corresponding predicted value for yi

MSE is used in software defect prediction as a quantitative measure of prediction accuracy, providing a standardised and interpretable way to evaluate the performance of regression models.Mean squared error (MSE) is commonly used in software defect prediction because it provides a quantitative measure of the accuracy of a regression model . In software defect prediction, the goal is to predict the number of defects in a software system based on various metrics and features.

The root mean square error (RMSE) is a measure of the accuracy of a prediction. It is calculated by taking the square root of the mean of the squared residuals. The residuals are the differences between the predicted values and the actual values.

The RMSE is a popular measure of accuracy because it is easy to understand and interpret. It is also a relatively robust measure of accuracy, meaning that it is not as sensitive to outliers as other measures of accuracy.

The RMSE is a unitless measure, meaning that it does not have any units. This makes it easy to compare the RMSE of models that are predicting different quantities.

The RMSE is calculated as follows:

RMSE = sqrt(sum((actual - predicted)^2) / n)

where:

- actual is the actual value
- predicted is the predicted value
- n is the number of data points

The RMSE is a good choice for software defect prediction because it is:

- Easy to understand: The RMSE is a simple and straightforward measure of accuracy. It is easy to calculate and interpret, even for non-technical audiences.
- Robust to outliers: The RMSE is relatively robust to outliers, meaning that a few extreme values will not significantly affect the overall score. This is important in software defect prediction, as there may be a few modules that have a significantly higher number of defects than the others.
- Comparable across models: The RMSE is a unitless measure, meaning that it can be used to compare the accuracy of models that are predicting different quantities. This makes it a useful tool for model selection and evaluation.

5.5 Weaknesses

While software defect prediction can be a valuable tool for improving software quality and development processes, it also has several weaknesses and limitations. It's important to be aware of these weaknesses when using defect prediction techniques to make informed decisions. Here are some common weaknesses of software defect prediction:

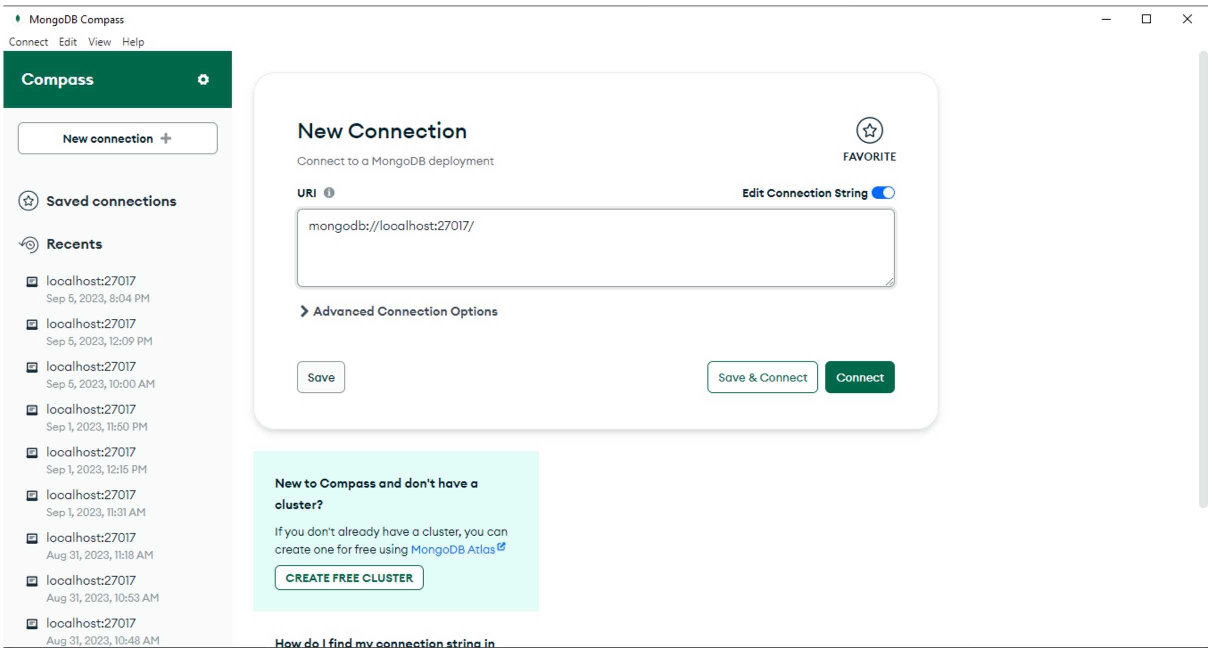
- Data Quality Issues: The accuracy and reliability of defect prediction models heavily depend on the quality of the historical data used for training. If the data is incomplete, inaccurate, or biased, it can lead to unreliable predictions.
- Imbalanced Data: In many software development projects, the number of defect-free instances (non-defective code) far outweighs the number of defective instances. This class imbalance can lead to skewed predictions, where the model may be biased towards predicting non-defects.
- Changing Environments: Software development environments are dynamic and can change over time. What may have been a reliable prediction model in the past might become less effective as the project evolves, leading to the need for continuous model retraining.
- Context Dependency: The effectiveness of defect prediction models can vary significantly based on the context of the project, programming language, technology stack, and development practices. A model that works well for one project may not work as well for another.
- Feature Selection: The choice of features (attributes or metrics) used in the defect prediction model is critical. Poor feature selection can lead to models that are less accurate in identifying defect-prone areas.
- Overfitting: Overfitting occurs when a model is too complex and fits the training data too closely, leading to poor generalization to new data. This can result in inaccurate predictions when applied to real-world projects.
- Limited Prediction Horizon: Most defect prediction models focus on short-term predictions, which means they may not identify defects that emerge in the long run. They are better at identifying immediate risks.
- Human Factors: Software development is influenced by human factors, such as developer expertise, collaboration, and code review processes. These factors can impact defect occurrence but are often not accounted for in defect prediction models.
- False Positives and False Negatives: Defect prediction models can generate false positives (predicting a defect that doesn't exist) and false negatives (failing to predict a defect that does exist). Balancing these errors is challenging.
- Lack of Causality: Defect prediction models can identify correlations but may not uncover the underlying causes of defects. Understanding why defects occur is essential for effective prevention.

- Resistance to Change: Developers and teams may be resistant to adopting defect prediction results, especially if they don't trust the model's accuracy or if they feel it challenges their autonomy.
- Ethical and Privacy Concerns: Collecting and using historical data for defect prediction must adhere to ethical and privacy standards. Ensuring that sensitive data is protected and used responsibly is essential.

In summary, software defect prediction is a valuable technique but has its weaknesses, including issues related to data quality, context dependency, and the complexities of software development. To mitigate these weaknesses, it's important to continually validate and refine prediction models, consider the specific context of the project, and integrate predictions into a broader quality assurance and risk management strategy. Additionally, it's crucial to communicate and collaborate effectively with development teams to ensure the practicality and acceptance of defect prediction results.

5.6 MongoDB, Jupyter Notebook

MongoDB Compass GUI to connect to MongoDB Server:



localhost:27017 Sep 5, 2023, 8:04 PM
localhost:27017 Sep 5, 2023, 12:09 PM
localhost:27017 Sep 5, 2023, 10:00 AM
localhost:27017 Sep 1, 2023, 11:50 PM
localhost:27017 Sep 1, 2023, 12:15 PM
localhost:27017 Sep 1, 2023, 11:31 AM
localhost:27017 Aug 31, 2023, 11:18 AM
localhost:27017 Aug 31, 2023, 10:53 AM
localhost:27017 Aug 31, 2023, 10:48 AM

New Connection

Connect to a MongoDB deployment

URI

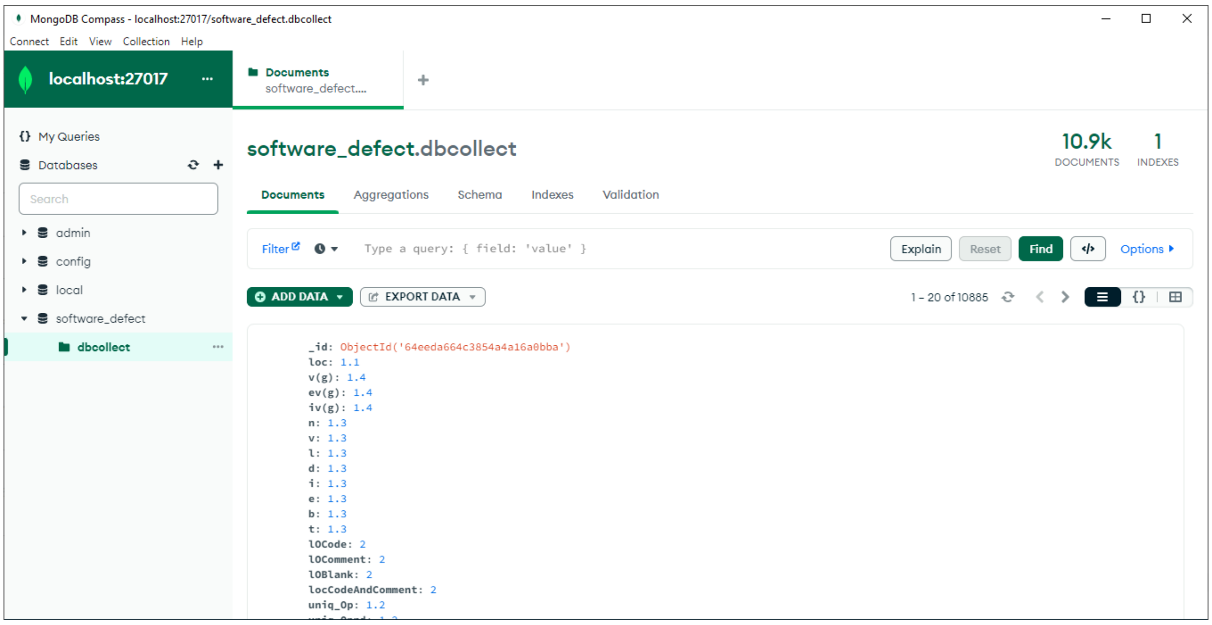
mongodb://localhost:27017/

New to Compass and don't have a cluster?

If you don't already have a cluster, you can create one for free using MongoDB Atlas

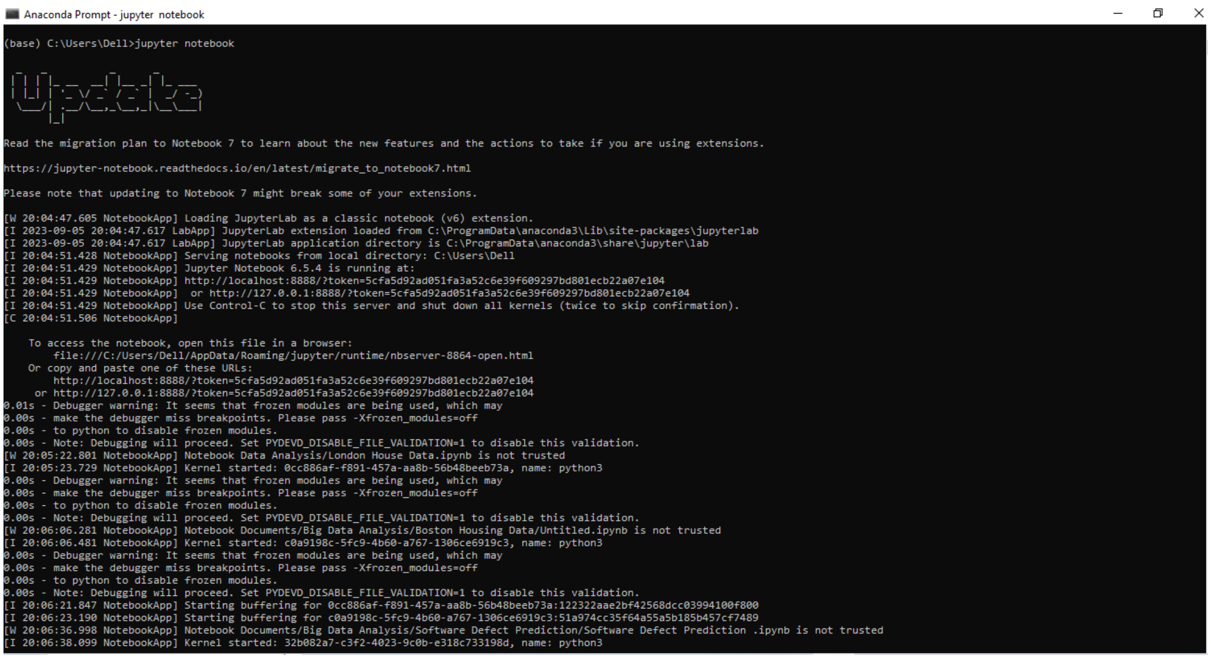
How do I find my connection string in

Inserting .csv into MongoDB database:


MongoDB Compass - localhost:27017/software_defect.dbcollect
Connect Edit View Collection Help
localhost:27017
Documents
software_defect....
My Queries
Databases
Search
admin
config
local
software_defect
dbcollect
software_defect.dbcollect
10.9k DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Filter
Type a query: { field: 'value' }
Explain Reset Find
Options
ADD DATA
EXPORT DATA
1 – 20 of 10885

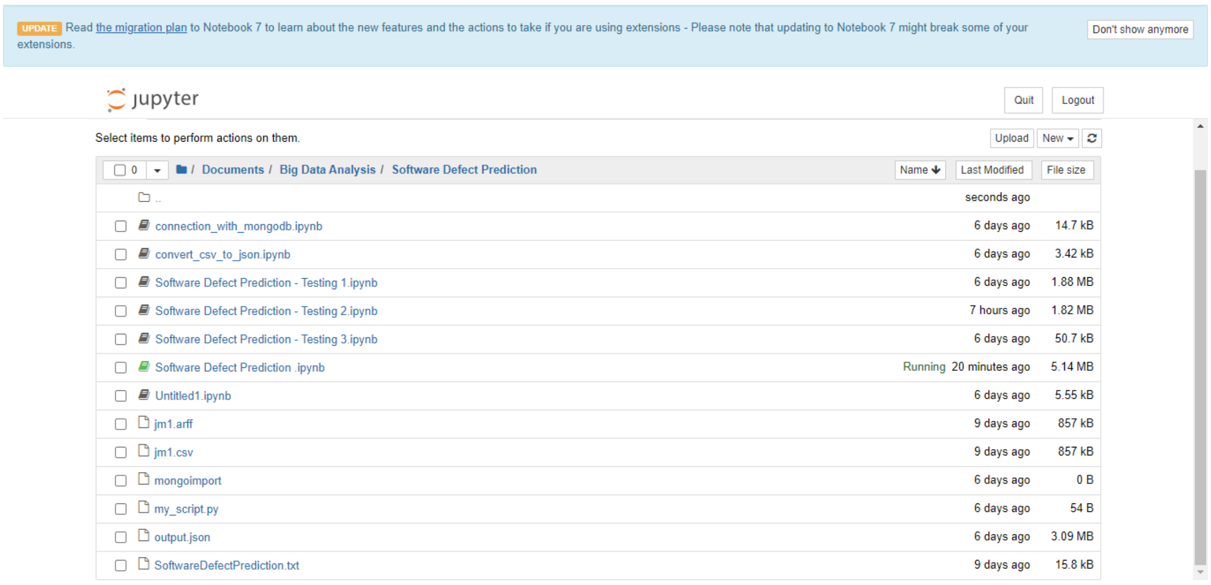
```
_id: ObjectId('64eeda664c3854a4a16a0bba')
loc: 1.1
v(g): 1.4
ev(g): 1.4
iv(g): 1.4
n: 1.3
v: 1.3
l: 1.3
d: 1.3
i: 1.3
e: 1.3
b: 1.3
t: 1.3
lOCode: 2
lOComment: 2
lOBlank: 2
locCodeAndComment: 2
uniq_Op: 1.2
```

Connect to Jupyter Notebook via Anacando:



```
(base) C:\Users\Dell>jupyter notebook

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[W 20:04:47.605 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[I 2023-09-05 20:04:47.617 LabApp] JupyterLab extension loaded from C:\ProgramData\anaconda3\Lib\site-packages\jupyterlab
[I 2023-09-05 20:04:47.617 LabApp] JupyterLab application directory is C:\ProgramData\anaconda3\share\jupyter\lab
[I 20:04:51.428 NotebookApp] Serving notebooks from local directory: C:\Users\Dell
[I 20:04:51.429 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 20:04:51.429 NotebookApp] http://localhost:8888/?token=5cfa5d92ad051fa3a52c6e39f609297bd801ecb22a07e104
[I 20:04:51.429 NotebookApp]  or http://127.0.0.1:8888/?token=5cfa5d92ad051fa3a52c6e39f609297bd801ecb22a07e104
[I 20:04:51.429 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 20:04:51.506 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/Dell/AppData/Roaming/jupyter/runtime/nbserver-8864-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=5cfa5d92ad051fa3a52c6e39f609297bd801ecb22a07e104
     or http://127.0.0.1:8888/?token=5cfa5d92ad051fa3a52c6e39f609297bd801ecb22a07e104
0.01s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[W 20:05:22.801 NotebookApp] Notebook Data Analysis/London House Data.ipynb is not trusted
[I 20:05:23.729 NotebookApp] Kernel started: 0cc886af-f891-457a-aa8b-56b48beeb73a, name: python3
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[W 20:06:06.281 NotebookApp] Notebook Documents/Big Data Analysis/Boston Housing Data/Untitled.ipynb is not trusted
[I 20:06:06.481 NotebookApp] Kernel started: c0a9198c-5fc9-4b60-a767-1306ce6919c3, name: python3
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[I 20:06:21.847 NotebookApp] Starting buffering for 0cc886af-f891-457a-aa8b-56b48beeb73a:122322aae2bf42568dcc03994100f800
[I 20:06:23.190 NotebookApp] Starting buffering for c0a9198c-5fc9-4b60-a767-1306ce6919c3:51a974cc35f64a55a5b185b457cf7489
[W 20:06:36.998 NotebookApp] Notebook Documents/Big Data Analysis/Software Defect Prediction/Software Defect Prediction .ipynb is not trusted
[I 20:06:38.099 NotebookApp] Kernel started: 32b082a7-c3f2-4023-9c0b-e318c733198d, name: python3
```

Jyputer Notebook:



UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions. Don't show anymore

jupyter Quit Logout

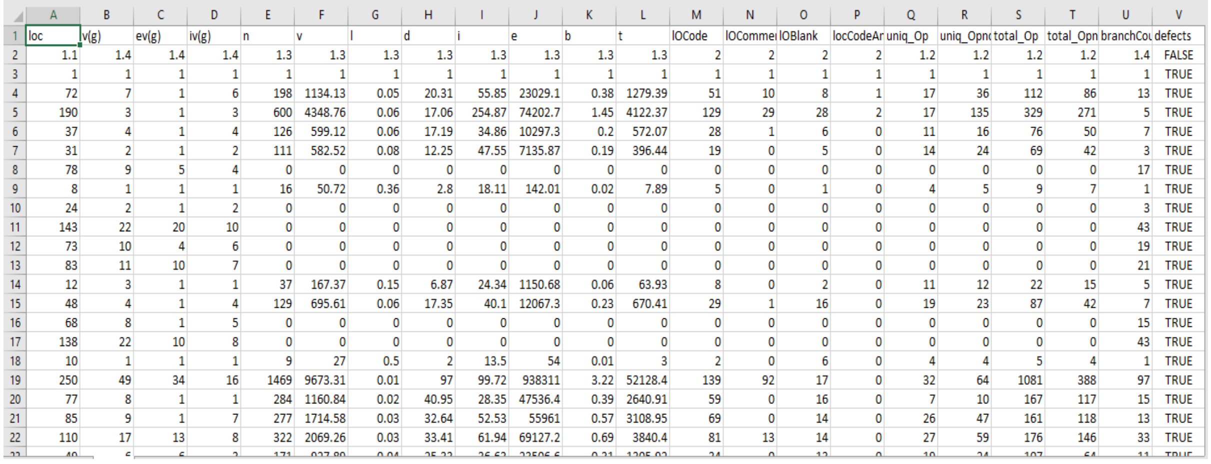
Select items to perform actions on them. Upload New

0 / Documents / Big Data Analysis / Software Defect Prediction — Name, Last Modified, File size

Name	Status	Last Modified	File size
..		seconds ago	
connection_with_mongodb.ipynb		6 days ago	14.7 kB
convert_csv_to_json.ipynb		6 days ago	3.42 kB
Software Defect Prediction - Testing 1.ipynb		6 days ago	1.88 MB
Software Defect Prediction - Testing 2.ipynb		7 hours ago	1.82 MB
Software Defect Prediction - Testing 3.ipynb		6 days ago	50.7 kB
Software Defect Prediction .ipynb	Running	20 minutes ago	5.14 MB
Untitled1.ipynb		6 days ago	5.55 kB
jm1.arff		9 days ago	857 kB
jm1.csv		9 days ago	857 kB
mongoimport		6 days ago	0 B
my_script.py		6 days ago	54 B
output.json		6 days ago	3.09 MB
SoftwareDefectPrediction.txt		9 days ago	15.8 kB

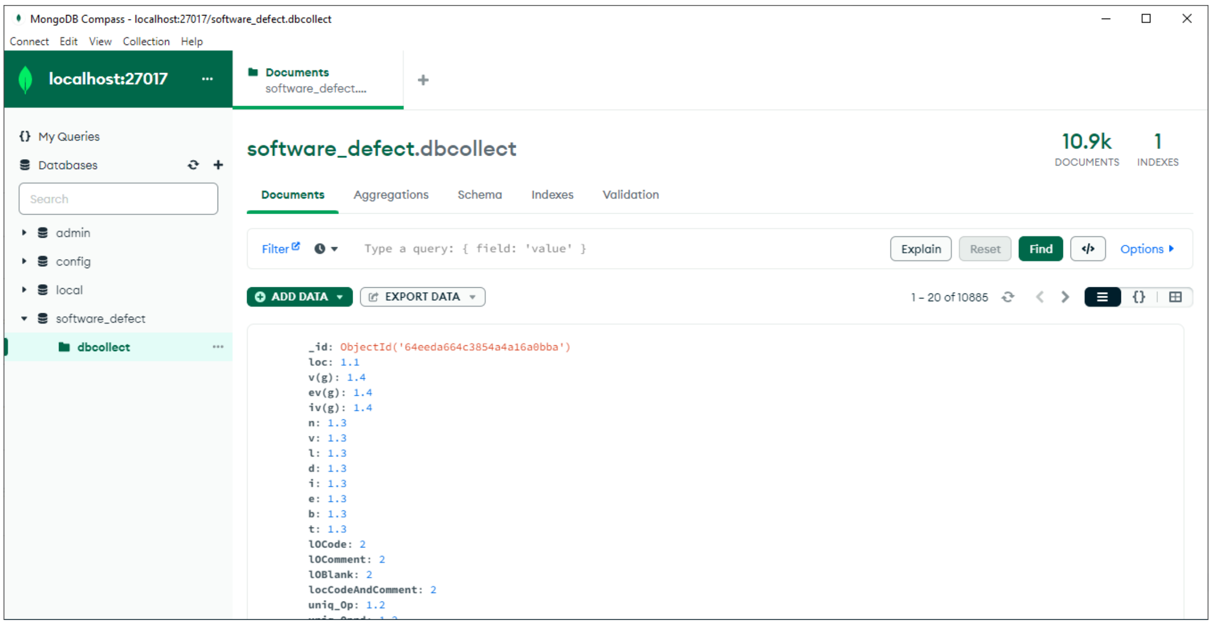
5.7 Training Data

Training Data set in CSV format:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOCommei	lOBlank	locCodeAr	uniq_Op	uniq_Opnc	total_Op	total_Opn	branchCou	defects
2	1.1	1.4	1.4	1.4	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2	2	2	2	1.2	1.2	1.2	1.2	1.4	FALSE
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	TRUE
4	72	7	1	6	198	1134.13	0.05	20.31	55.85	23029.1	0.38	1279.39	51	10	8	1	17	36	112	86	13	TRUE
5	190	3	1	3	600	4348.76	0.06	17.06	254.87	74202.7	1.45	4122.37	129	29	28	2	17	135	329	271	5	TRUE
6	37	4	1	4	126	599.12	0.06	17.19	34.86	10297.3	0.2	572.07	28	1	6	0	11	16	76	50	7	TRUE
7	31	2	1	2	111	582.52	0.08	12.25	47.55	7135.87	0.19	396.44	19	0	5	0	14	24	69	42	3	TRUE
8	78	9	5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	TRUE
9	8	1	1	1	16	50.72	0.36	2.8	18.11	142.01	0.02	7.89	5	0	1	0	4	5	9	7	1	TRUE
10	24	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	TRUE
11	143	22	20	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	TRUE
12	73	10	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	TRUE
13	83	11	10	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	TRUE
14	12	3	1	1	37	167.37	0.15	6.87	24.34	1150.68	0.06	63.93	8	0	2	0	11	12	22	15	5	TRUE
15	48	4	1	4	129	695.61	0.06	17.35	40.1	12067.3	0.23	670.41	29	1	16	0	19	23	87	42	7	TRUE
16	68	8	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	TRUE
17	138	22	10	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	TRUE
18	10	1	1	1	9	27	0.5	2	13.5	54	0.01	3	2	0	6	0	4	4	5	4	1	TRUE
19	250	49	34	16	1469	9673.31	0.01	97	99.72	938311	3.22	52128.4	139	92	17	0	32	64	1081	388	97	TRUE
20	77	8	1	1	284	1160.84	0.02	40.95	28.35	47536.4	0.39	2640.91	59	0	16	0	7	10	167	117	15	TRUE
21	85	9	1	7	277	1714.58	0.03	32.64	52.53	55961	0.57	3108.95	69	0	14	0	26	47	161	118	13	TRUE
22	110	17	13	8	322	2069.26	0.03	33.41	61.94	69127.2	0.69	3840.4	81	13	14	0	27	59	176	146	33	TRUE

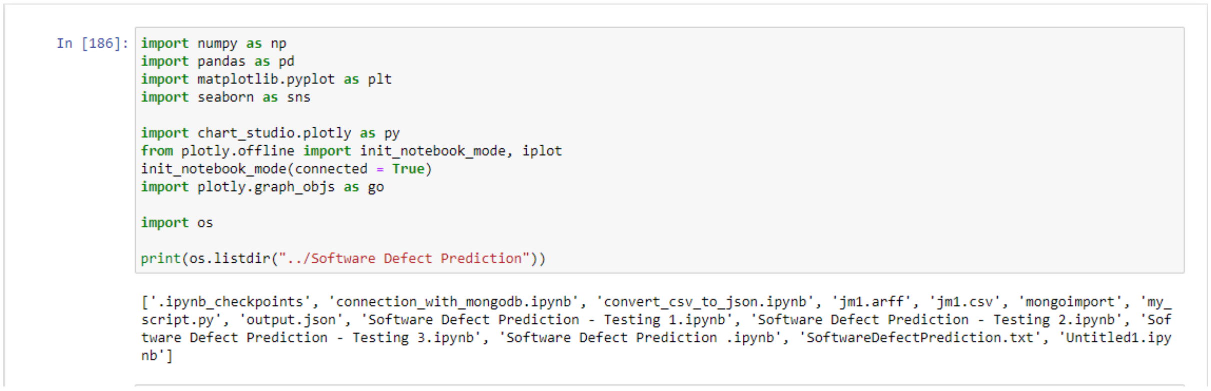
Actual big data training set(Mongodb data format) for software defect prediction:


MongoDB Compass - localhost:27017/software_defect.dbcollect
Connect Edit View Collection Help
localhost:27017
Documents
software_defect....
My Queries
Databases
Search
admin
config
local
software_defect
dbcollect
software_defect.dbcollect
10.9k DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Filter
Type a query: { field: 'value' }
Explain Reset Find
Options
ADD DATA
EXPORT DATA
1 – 20 of 10885

```
_id: ObjectId('64eeda664c3854a4a16a0bba')
loc: 1.1
v(g): 1.4
ev(g): 1.4
iv(g): 1.4
n: 1.3
v: 1.3
l: 1.3
d: 1.3
i: 1.3
e: 1.3
b: 1.3
t: 1.3
lOCode: 2
lOComment: 2
lOBlank: 2
locCodeAndComment: 2
uniq_Op: 1.2
```

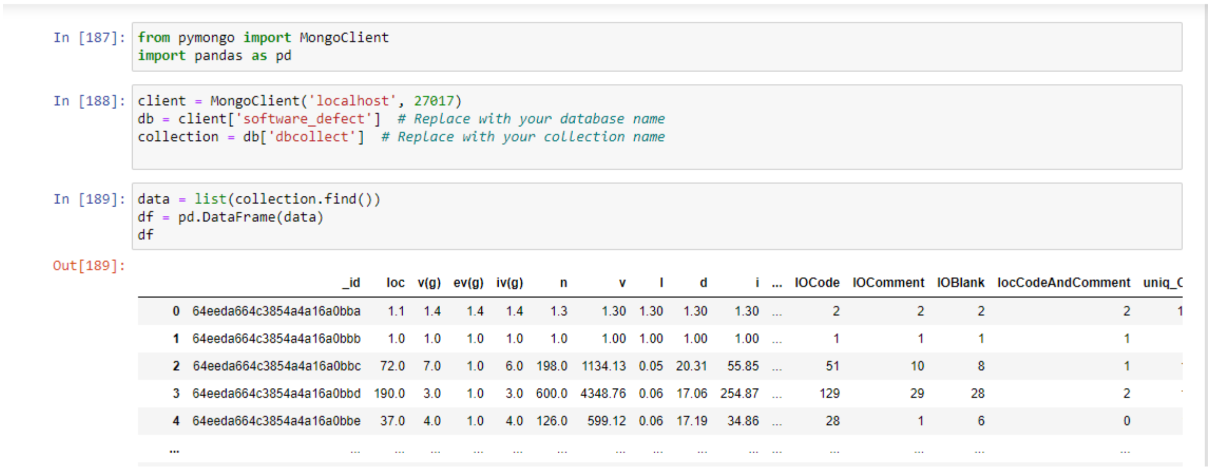
5.8 Output

Importing necessary library file:



```
In [186]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns

          import chart_studio.plotly as py
          from plotly.offline import init_notebook_mode, iplot
          init_notebook_mode(connected = True)
          import plotly.graph_objs as go

          import os

          print(os.listdir("../Software Defect Prediction"))

          ['.ipynb_checkpoints', 'connection_with_mongodb.ipynb', 'convert_csv_to_json.ipynb', 'jm1.arff', 'jm1.csv', 'mongoimport', 'my_
          script.py', 'output.json', 'Software Defect Prediction - Testing 1.ipynb', 'Software Defect Prediction - Testing 2.ipynb', 'Sof
          tware Defect Prediction - Testing 3.ipynb', 'Software Defect Prediction .ipynb', 'SoftwareDefectPrediction.txt', 'Untitled1.ipy
          nb']
```

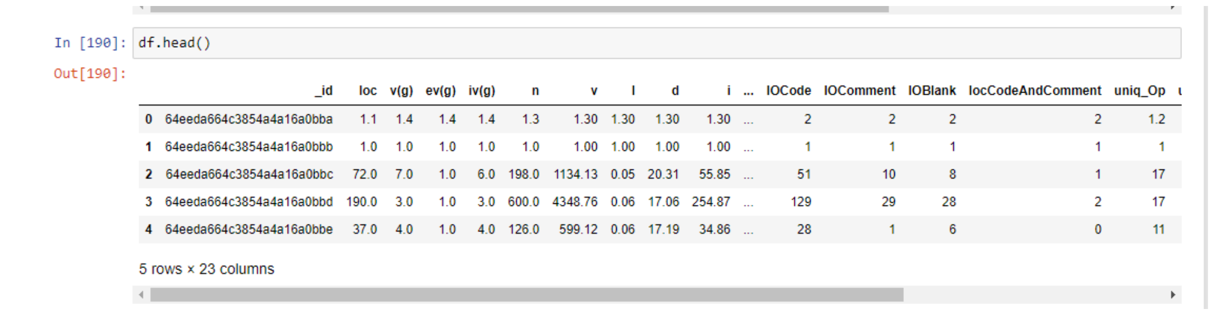
Connect to MongoDB and retrieve dataset from database:



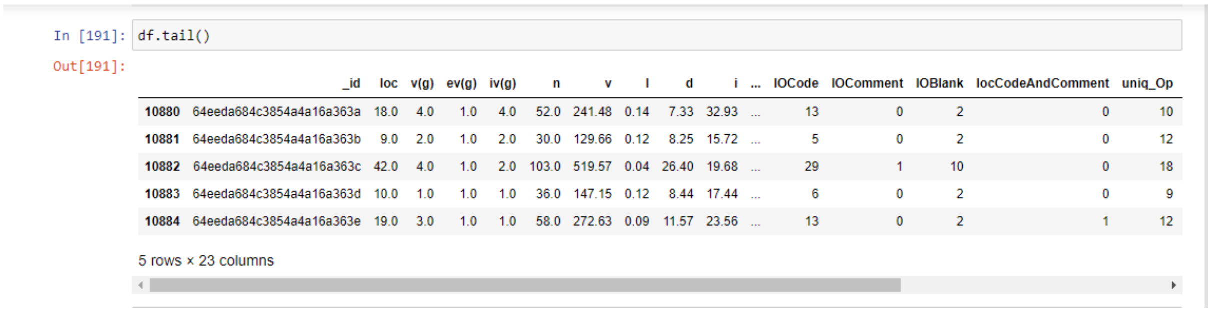
```
In [187]: from pymongo import MongoClient
          import pandas as pd

In [188]: client = MongoClient('localhost', 27017)
          db = client['software_defect']  # Replace with your database name
          collection = db['dbcollect']  # Replace with your collection name

In [189]: data = list(collection.find())
          df = pd.DataFrame(data)
          df

Out[189]:
                            _id    loc  v(g)  ev(g)  iv(g)      n        v     l      d       i  ...  IOCode  IOComment  IOBlank  locCodeAndComment  uniq_C
   0  64eeda664c3854a4a16a0bba    1.1   1.4    1.4    1.4    1.3     1.30  1.30   1.30    1.30  ...       2          2        2                  2       1
   1  64eeda664c3854a4a16a0bbb    1.0   1.0    1.0    1.0    1.0     1.00  1.00   1.00    1.00  ...       1          1        1                  1
   2  64eeda664c3854a4a16a0bbc   72.0   7.0    1.0    6.0  198.0  1134.13  0.05  20.31   55.85  ...      51         10        8                  1
   3  64eeda664c3854a4a16a0bbd  190.0   3.0    1.0    3.0  600.0  4348.76  0.06  17.06  254.87  ...     129         29       28                  2
   4  64eeda664c3854a4a16a0bbe   37.0   4.0    1.0    4.0  126.0   599.12  0.06  17.19   34.86  ...      28          1        6                  0
 ...                       ...    ...   ...    ...    ...    ...      ...   ...    ...     ...  ...     ...        ...      ...                ...
```

Data Discovery

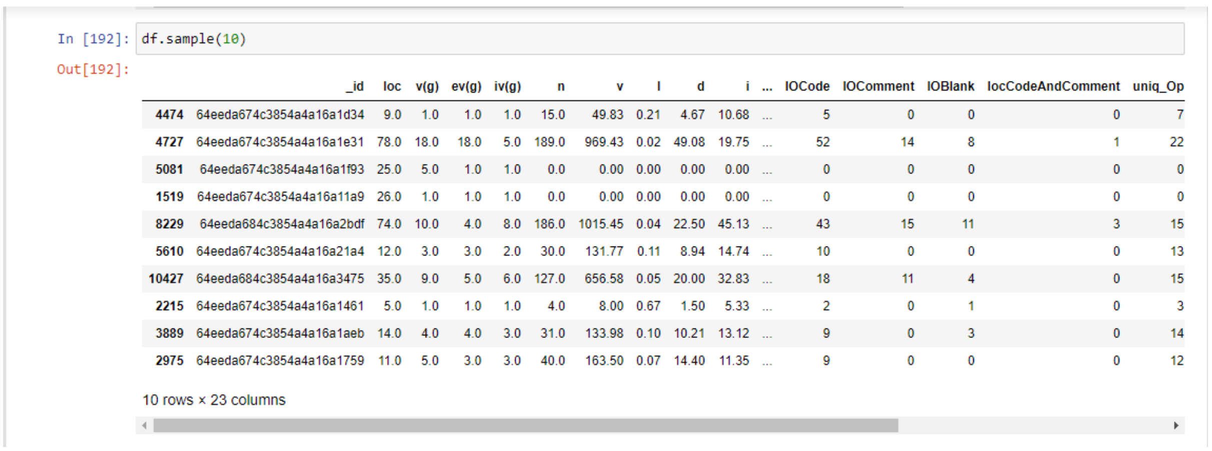
Retrieve rows in different form to understand the nature of attributes:



```
In [190]: df.head()

Out[190]:
                         _id    loc  v(g)  ev(g)  iv(g)      n        v     l      d       i  ...  IOCode  IOComment  IOBlank  locCodeAndComment  uniq_Op
0  64eeda664c3854a4a16a0bba    1.1   1.4    1.4    1.4    1.3     1.30  1.30   1.30    1.30  ...       2          2        2                  2      1.2
1  64eeda664c3854a4a16a0bbb    1.0   1.0    1.0    1.0    1.0     1.00  1.00   1.00    1.00  ...       1          1        1                  1        1
2  64eeda664c3854a4a16a0bbc   72.0   7.0    1.0    6.0  198.0  1134.13  0.05  20.31   55.85  ...      51         10        8                  1       17
3  64eeda664c3854a4a16a0bbd  190.0   3.0    1.0    3.0  600.0  4348.76  0.06  17.06  254.87  ...     129         29       28                  2       17
4  64eeda664c3854a4a16a0bbe   37.0   4.0    1.0    4.0  126.0   599.12  0.06  17.19   34.86  ...      28          1        6                  0       11

5 rows × 23 columns
```



```
In [191]: df.tail()

Out[191]:
                             _id   loc  v(g)  ev(g)  iv(g)      n       v     l      d      i  ...  IOCode  IOComment  IOBlank  locCodeAndComment  uniq_Op
10880  64eeda684c3854a4a16a363a  18.0   4.0    1.0    4.0   52.0  241.48  0.14   7.33  32.93  ...      13          0        2                  0       10
10881  64eeda684c3854a4a16a363b   9.0   2.0    1.0    2.0   30.0  129.66  0.12   8.25  15.72  ...       5          0        2                  0       12
10882  64eeda684c3854a4a16a363c  42.0   4.0    1.0    2.0  103.0  519.57  0.04  26.40  19.68  ...      29          1       10                  0       18
10883  64eeda684c3854a4a16a363d  10.0   1.0    1.0    1.0   36.0  147.15  0.12   8.44  17.44  ...       6          0        2                  0        9
10884  64eeda684c3854a4a16a363e  19.0   3.0    1.0    1.0   58.0  272.63  0.09  11.57  23.56  ...      13          0        2                  1       12

5 rows × 23 columns
```

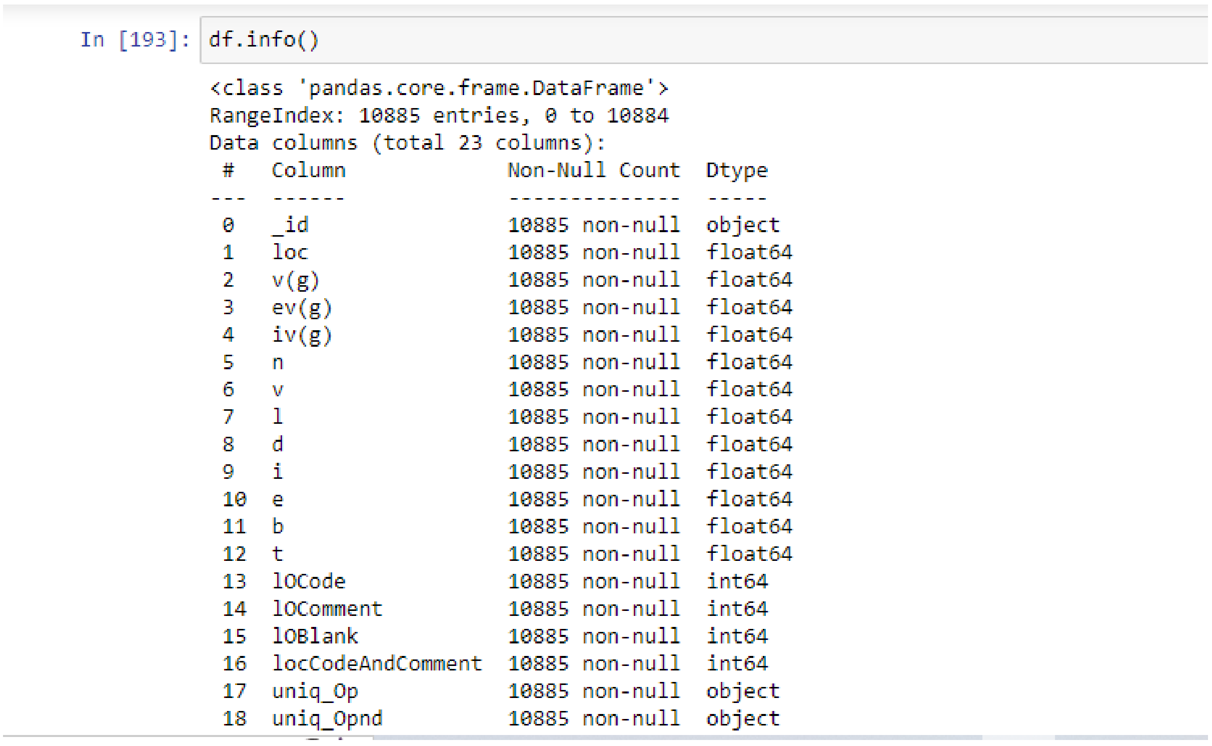


```
In [192]: df.sample(10)
Out[192]:
```

	_id	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op
4474	64eeda674c3854a4a16a1d34	9.0	1.0	1.0	1.0	15.0	49.83	0.21	4.67	10.68	...	5	0	0	0	7
4727	64eeda674c3854a4a16a1e31	78.0	18.0	18.0	5.0	189.0	969.43	0.02	49.08	19.75	...	52	14	8	1	22
5081	64eeda674c3854a4a16a1f93	25.0	5.0	1.0	1.0	0.0	0.00	0.00	0.00	0.00	...	0	0	0	0	0
1519	64eeda674c3854a4a16a11a9	26.0	1.0	1.0	1.0	0.0	0.00	0.00	0.00	0.00	...	0	0	0	0	0
8229	64eeda684c3854a4a16a2bdf	74.0	10.0	4.0	8.0	186.0	1015.45	0.04	22.50	45.13	...	43	15	11	3	15
5610	64eeda674c3854a4a16a21a4	12.0	3.0	3.0	2.0	30.0	131.77	0.11	8.94	14.74	...	10	0	0	0	13
10427	64eeda684c3854a4a16a3475	35.0	9.0	5.0	6.0	127.0	656.58	0.05	20.00	32.83	...	18	11	4	0	15
2215	64eeda674c3854a4a16a1461	5.0	1.0	1.0	1.0	4.0	8.00	0.67	1.50	5.33	...	2	0	1	0	3
3889	64eeda674c3854a4a16a1aeb	14.0	4.0	4.0	3.0	31.0	133.98	0.10	10.21	13.12	...	9	0	3	0	14
2975	64eeda674c3854a4a16a1759	11.0	5.0	3.0	3.0	40.0	163.50	0.07	14.40	11.35	...	9	0	0	0	12

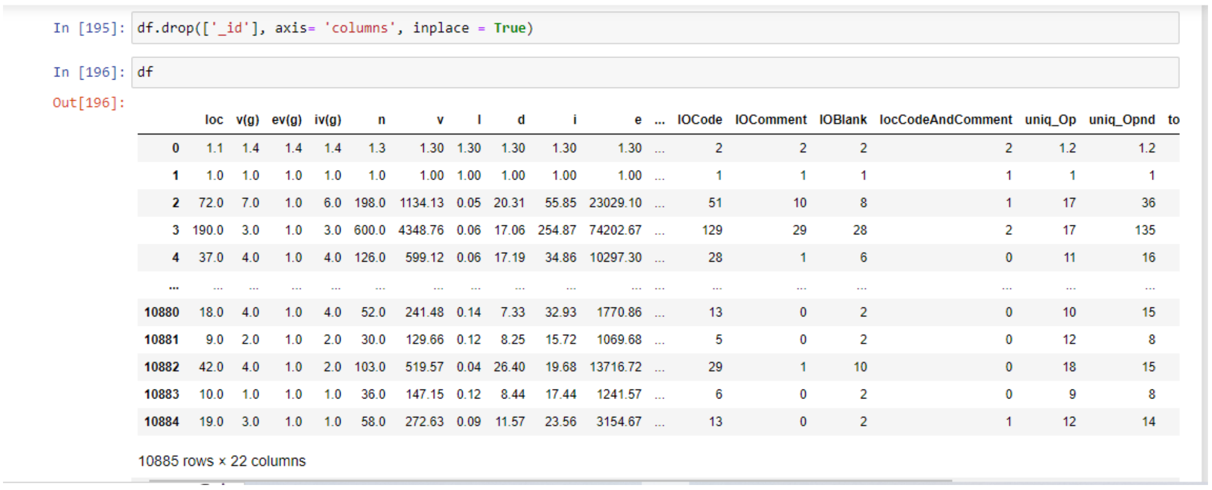
10 rows × 23 columns

Information about data (memory usage, data types, etc.)



```
In [193]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   _id                10885 non-null  object
 1   loc                10885 non-null  float64
 2   v(g)               10885 non-null  float64
 3   ev(g)              10885 non-null  float64
 4   iv(g)              10885 non-null  float64
 5   n                  10885 non-null  float64
 6   v                  10885 non-null  float64
 7   l                  10885 non-null  float64
 8   d                  10885 non-null  float64
 9   i                  10885 non-null  float64
 10  e                  10885 non-null  float64
 11  b                  10885 non-null  float64
 12  t                  10885 non-null  float64
 13  lOCode             10885 non-null  int64
 14  lOComment          10885 non-null  int64
 15  lOBlank            10885 non-null  int64
 16  locCodeAndComment  10885 non-null  int64
 17  uniq_Op            10885 non-null  object
 18  uniq_Opnd          10885 non-null  object
```

Dropping id column that is automatically set by MongoDB for indexing:

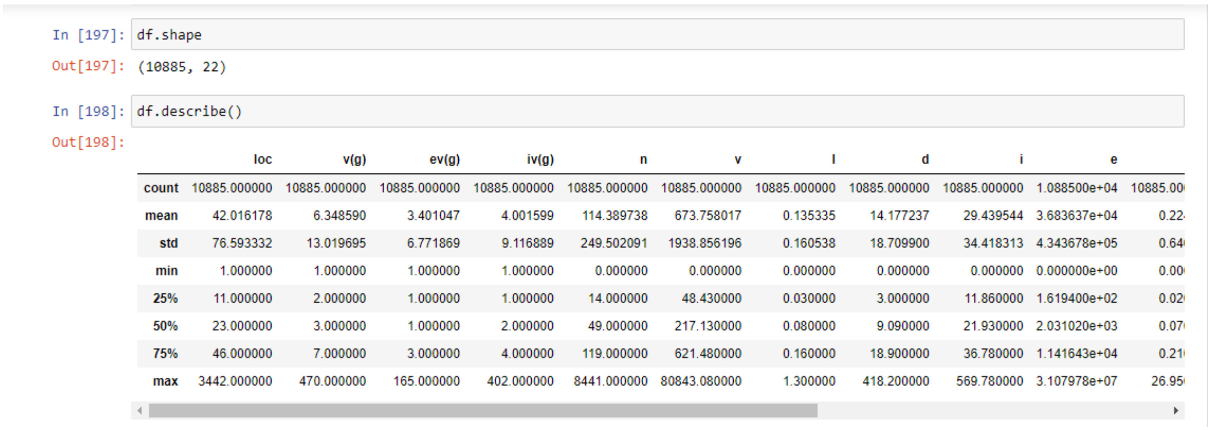


```
In [195]: df.drop(['_id'], axis= 'columns', inplace = True)

In [196]: df
Out[196]:
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	to
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	13	0	2	0	10	15	
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	5	0	2	0	12	8	
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	29	1	10	0	18	15	
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	6	0	2	0	9	8	
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	13	0	2	1	12	14	

10885 rows × 22 columns

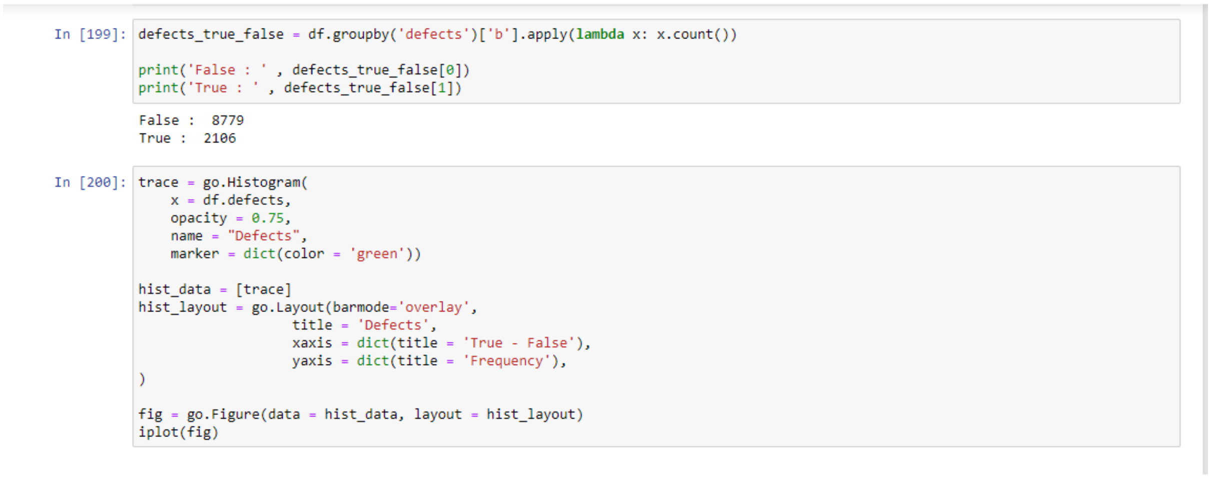
Retrieve information about dataset:



```
In [197]: df.shape
Out[197]: (10885, 22)

In [198]: df.describe()
Out[198]:
```

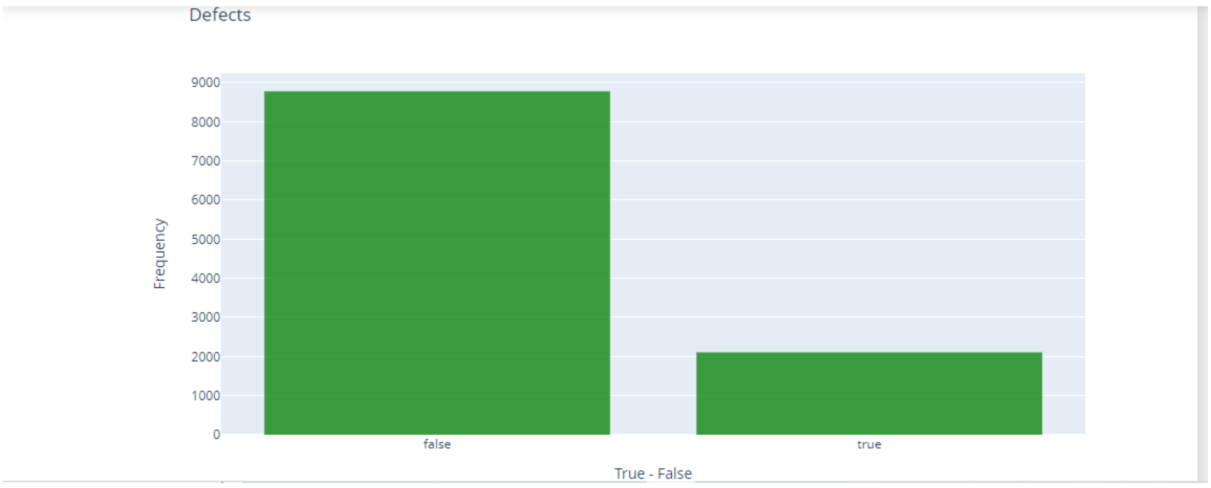
	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	1.088500e+04	10885.00
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335	14.177237	29.439544	3.683637e+04	0.22
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538	18.709900	34.418313	4.343678e+05	0.64
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.00
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.860000	1.619400e+02	0.02
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.930000	2.031020e+03	0.07
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000	18.900000	36.780000	1.141643e+04	0.21
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.780000	3.107978e+07	26.95

Classified by whether there are defects or not:

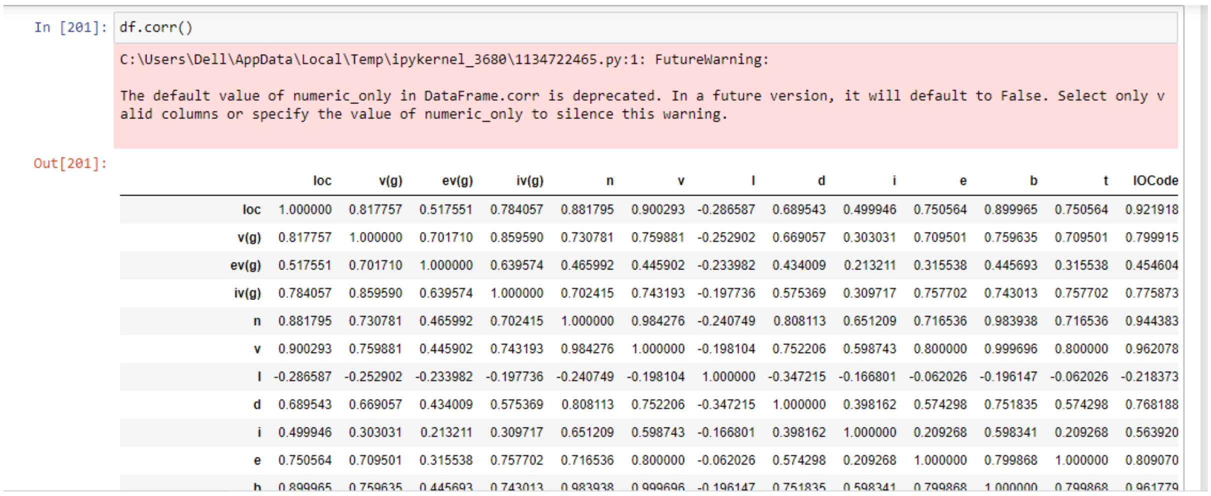


```
In [199]: defects_true_false = df.groupby('defects')['b'].apply(lambda x: x.count())

          print('False : ' , defects_true_false[0])
          print('True : ' , defects_true_false[1])

          False :  8779
          True :  2106

In [200]: trace = go.Histogram(
              x = df.defects,
              opacity = 0.75,
              name = "Defects",
              marker = dict(color = 'green'))

          hist_data = [trace]
          hist_layout = go.Layout(barmode='overlay',
                             title = 'Defects',
                             xaxis = dict(title = 'True - False'),
                             yaxis = dict(title = 'Frequency'),
          )

          fig = go.Figure(data = hist_data, layout = hist_layout)
          iplot(fig)
```

Histogram (for defect):



Covariance (Relationship between attributes):



```
In [201]: df.corr()
```

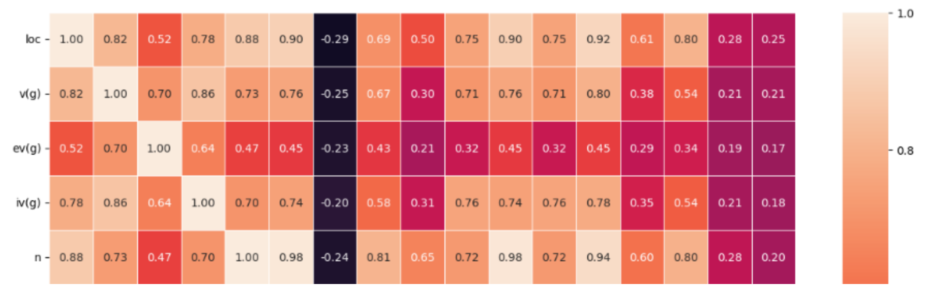
```
C:\Users\Dell\AppData\Local\Temp\ipykernel_3680\1134722465.py:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only v
alid columns or specify the value of numeric_only to silence this warning.
```

```
Out[201]:
              loc      v(g)     ev(g)     iv(g)         n         v         l         d         i         e         b         t    IOCode
   loc   1.000000  0.817757  0.517551  0.784057  0.881795  0.900293 -0.286587  0.689543  0.499946  0.750564  0.899965  0.750564  0.921918
  v(g)   0.817757  1.000000  0.701710  0.859590  0.730781  0.759881 -0.252902  0.669057  0.303031  0.709501  0.759635  0.709501  0.799915
 ev(g)   0.517551  0.701710  1.000000  0.639574  0.465992  0.445902 -0.233982  0.434009  0.213211  0.315538  0.445693  0.315538  0.454604
 iv(g)   0.784057  0.859590  0.639574  1.000000  0.702415  0.743193 -0.197736  0.575369  0.309717  0.757702  0.743013  0.757702  0.775873
     n   0.881795  0.730781  0.465992  0.702415  1.000000  0.984276 -0.240749  0.808113  0.651209  0.716536  0.983938  0.716536  0.944383
     v   0.900293  0.759881  0.445902  0.743193  0.984276  1.000000 -0.198104  0.752206  0.598743  0.800000  0.999696  0.800000  0.962078
     l  -0.286587 -0.252902 -0.233982 -0.197736 -0.240749 -0.198104  1.000000 -0.347215 -0.166801 -0.062026 -0.196147 -0.062026 -0.218373
     d   0.689543  0.669057  0.434009  0.575369  0.808113  0.752206 -0.347215  1.000000  0.398162  0.574298  0.751835  0.574298  0.768188
     i   0.499946  0.303031  0.213211  0.309717  0.651209  0.598743 -0.166801  0.398162  1.000000  0.209268  0.598341  0.209268  0.563920
     e   0.750564  0.709501  0.315538  0.757702  0.716536  0.800000 -0.062026  0.574298  0.209268  1.000000  0.799868  1.000000  0.809070
     b   0.899965  0.759635  0.445693  0.743013  0.983938  0.999696 -0.196147  0.751835  0.598341  0.799868  1.000000  0.799868  0.961779
```

Heatmap (Visualization of relationship between attributes):

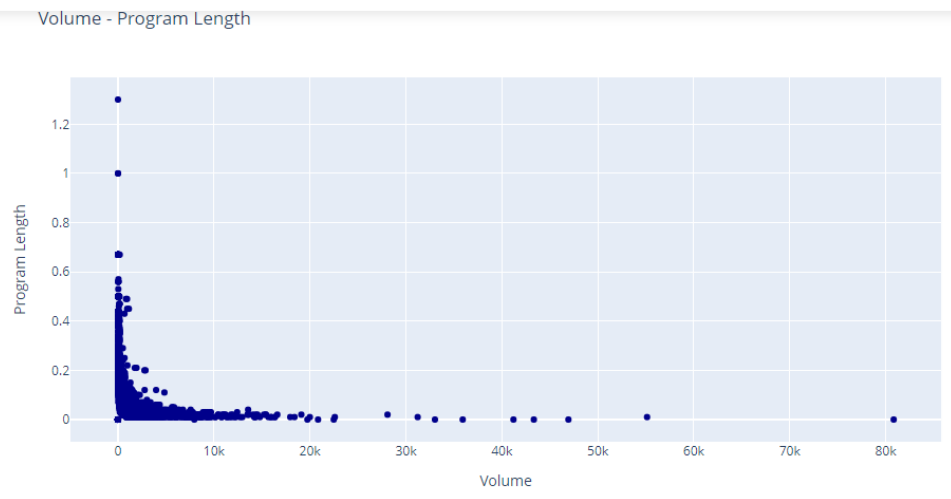
```
In [203]: f,ax = plt.subplots(figsize = (15, 15))
          sns.heatmap(df.corr(), annot = True, linewidths = .5, fmt = '.2f')
          plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_3680\796610870.py:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only v
alid columns or specify the value of numeric_only to silence this warning.
```



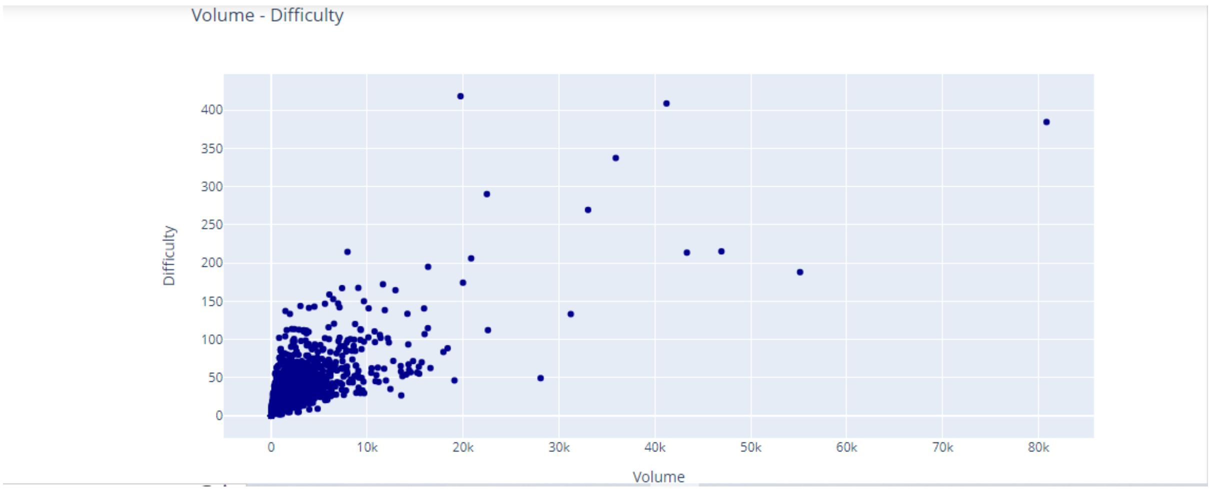
Scatter Plot (Volume - Program Length)

```
In [204]: trace = go.Scatter(
              x = df.v,
              y = df.l,
              mode = "markers",
              name = "Volume - Program Length",
              marker = dict(color = 'darkblue'),
              text = "Program Length (l)")

          scatter_data = [trace]
          scatter_layout = dict(title = 'Volume - Program Length',
                        xaxis = dict(title = 'Volume', ticklen = 5),
                        yaxis = dict(title = 'Program Length' , ticklen = 5),
                       )

          fig = dict(data = scatter_data, layout = scatter_layout)
          iplot(fig)
```

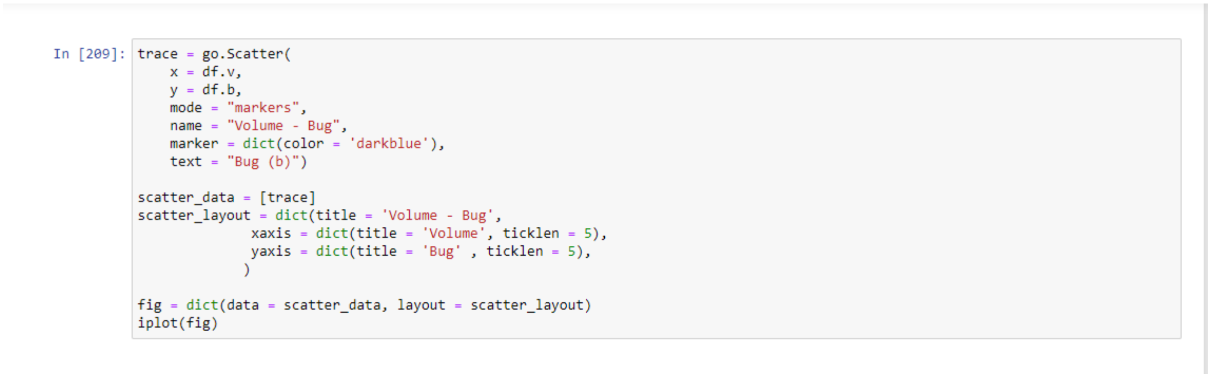


Scatter Plot (Volume - Difficulty)

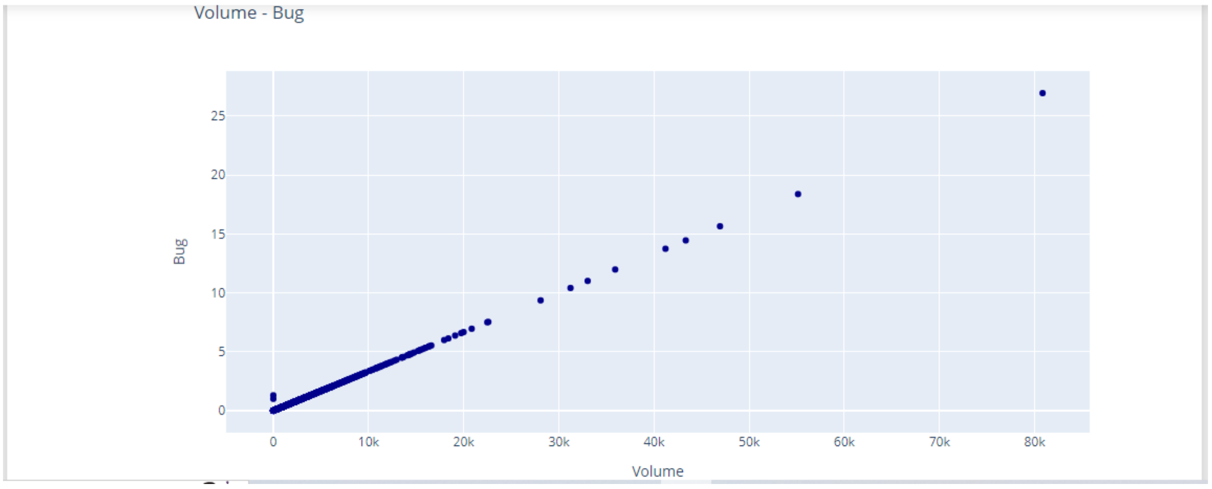
```
In [208]: trace = go.Scatter(
              x = df.v,
              y = df.d,
              mode = "markers",
              name = "Volume - Difficulty",
              marker = dict(color = 'darkblue'),
              text = "Difficulty (d)")

          scatter_data = [trace]
          scatter_layout = dict(title = 'Volume - Difficulty',
                        xaxis = dict(title = 'Volume', ticklen = 5),
                        yaxis = dict(title = 'Difficulty' , ticklen = 5),
                       )

          fig = dict(data = scatter_data, layout = scatter_layout)
          iplot(fig)
```



Scatter Plot (Volume - Bug)

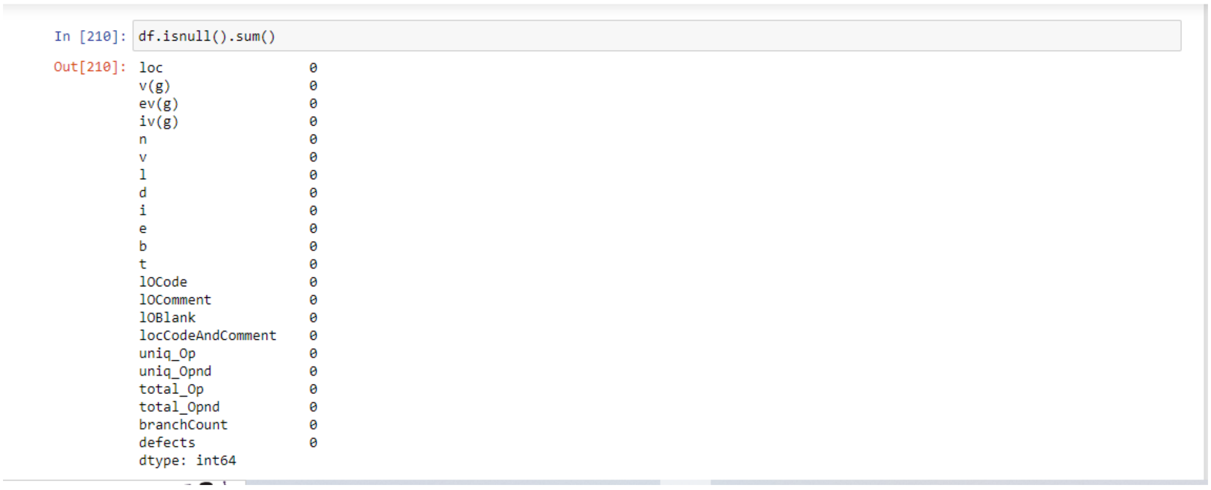


```
In [209]: trace = go.Scatter(
              x = df.v,
              y = df.b,
              mode = "markers",
              name = "Volume - Bug",
              marker = dict(color = 'darkblue'),
              text = "Bug (b)")

          scatter_data = [trace]
          scatter_layout = dict(title = 'Volume - Bug',
                        xaxis = dict(title = 'Volume', ticklen = 5),
                        yaxis = dict(title = 'Bug' , ticklen = 5),
                       )

          fig = dict(data = scatter_data, layout = scatter_layout)
          iplot(fig)
```



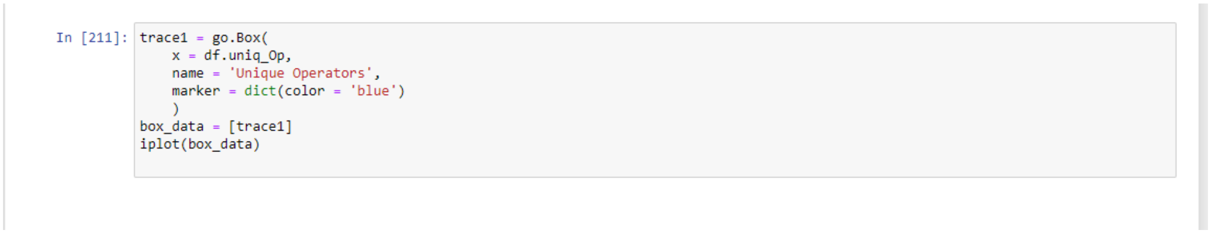
Data Preprocessing

Null Value Detection:

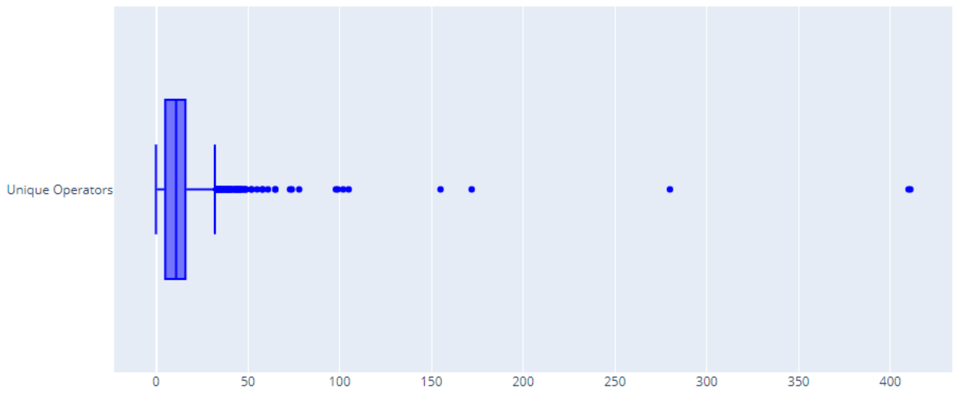


```
In [210]: df.isnull().sum()

Out[210]: loc                  0
          v(g)                 0
          ev(g)                0
          iv(g)                0
          n                    0
          v                    0
          l                    0
          d                    0
          i                    0
          e                    0
          b                    0
          t                    0
          lOCode               0
          lOComment            0
          lOBlank              0
          locCodeAndComment    0
          uniq_Op              0
          uniq_Opnd            0
          total_Op             0
          total_Opnd           0
          branchCount          0
          defects              0
          dtype: int64
```

Outlier Detection (Box Plot):

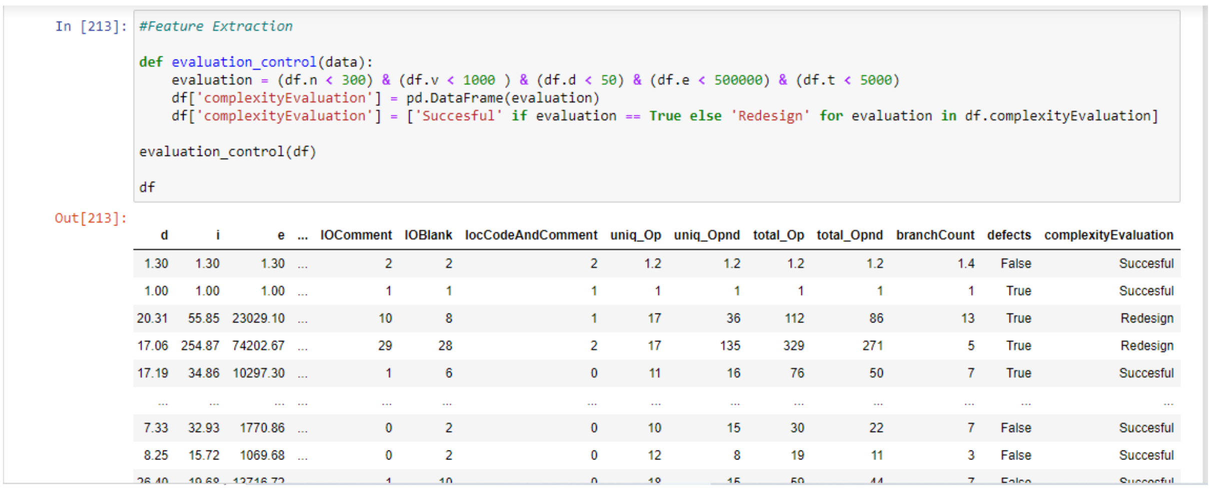


```
In [211]: trace1 = go.Box(
              x = df.uniq_Op,
              name = 'Unique Operators',
              marker = dict(color = 'blue')
              )
          box_data = [trace1]
          iplot(box_data)
```



Feature Extraction

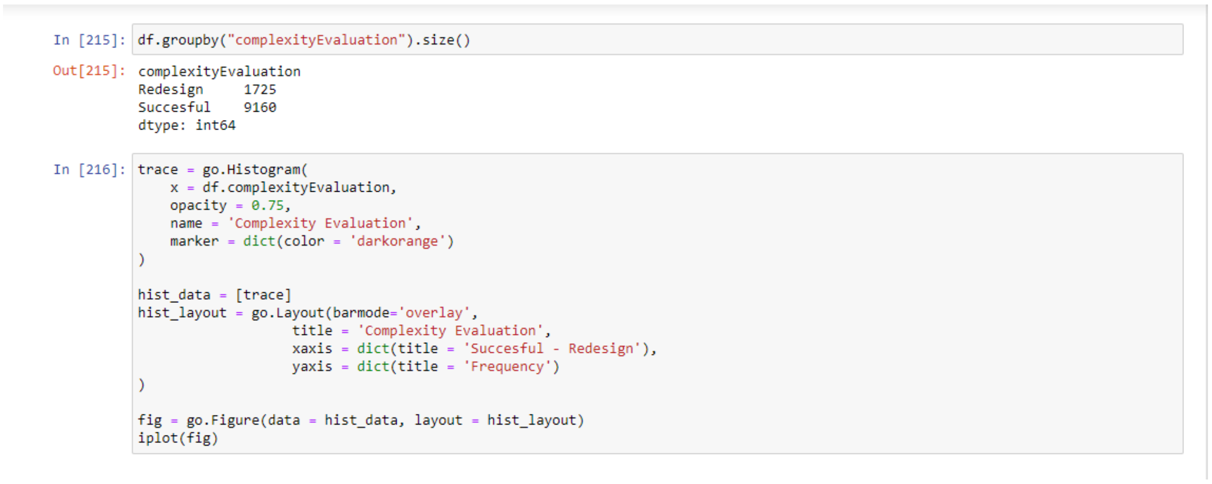
Adding new feature (complexityEvaluation):



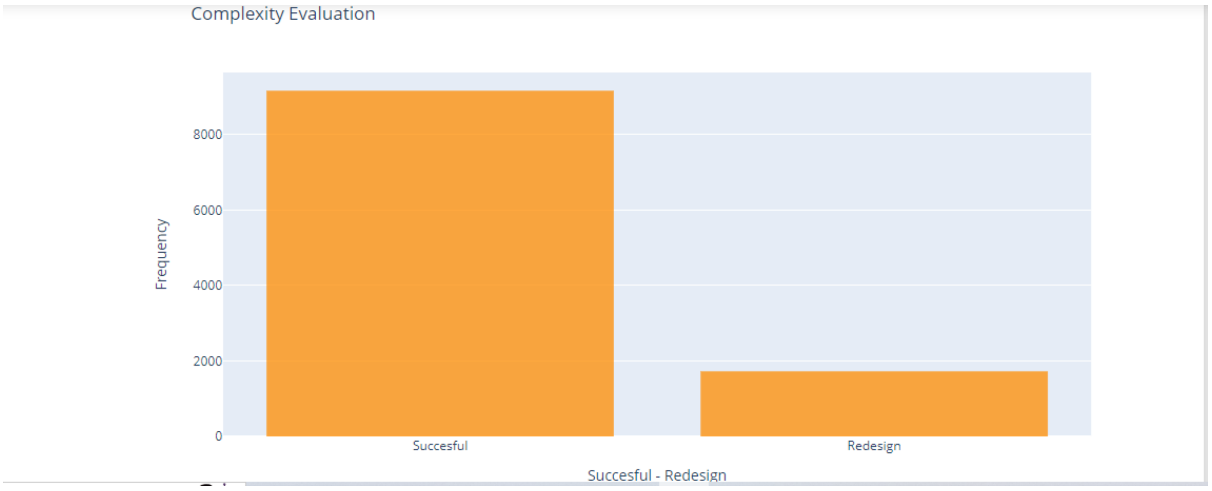
```
In [213]: #Feature Extraction

          def evaluation_control(data):
              evaluation = (df.n < 300) & (df.v < 1000 ) & (df.d < 50) & (df.e < 500000) & (df.t < 5000)
              df['complexityEvaluation'] = pd.DataFrame(evaluation)
              df['complexityEvaluation'] = ['Succesful' if evaluation == True else 'Redesign' for evaluation in df.complexityEvaluation]

          evaluation_control(df)

          df
Out[213]:
```

d	i	e	...	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount	defects	complexityEvaluation
1.30	1.30	1.30	...	2	2	2	1.2	1.2	1.2	1.2	1.4	False	Succesful
1.00	1.00	1.00	...	1	1	1	1	1	1	1	1	True	Succesful
20.31	55.85	23029.10	...	10	8	1	17	36	112	86	13	True	Redesign
17.06	254.87	74202.67	...	29	28	2	17	135	329	271	5	True	Redesign
17.19	34.86	10297.30	...	1	6	0	11	16	76	50	7	True	Succesful
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7.33	32.93	1770.86	...	0	2	0	10	15	30	22	7	False	Succesful
8.25	15.72	1069.68	...	0	2	0	12	8	19	11	3	False	Succesful
26.40	19.68	13716.72	...	1	10	0	18	15	59	44	7	False	Succesful

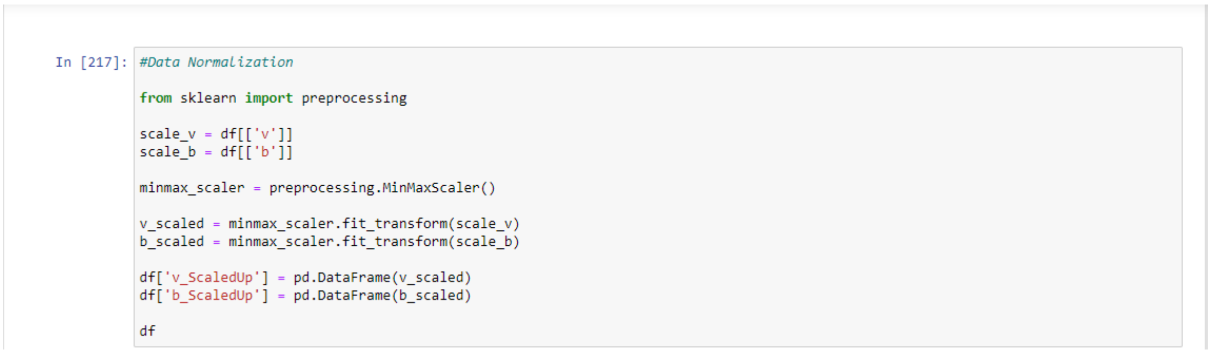
Classified by new feature (complexityEvaluation) and visualization (Histogram):

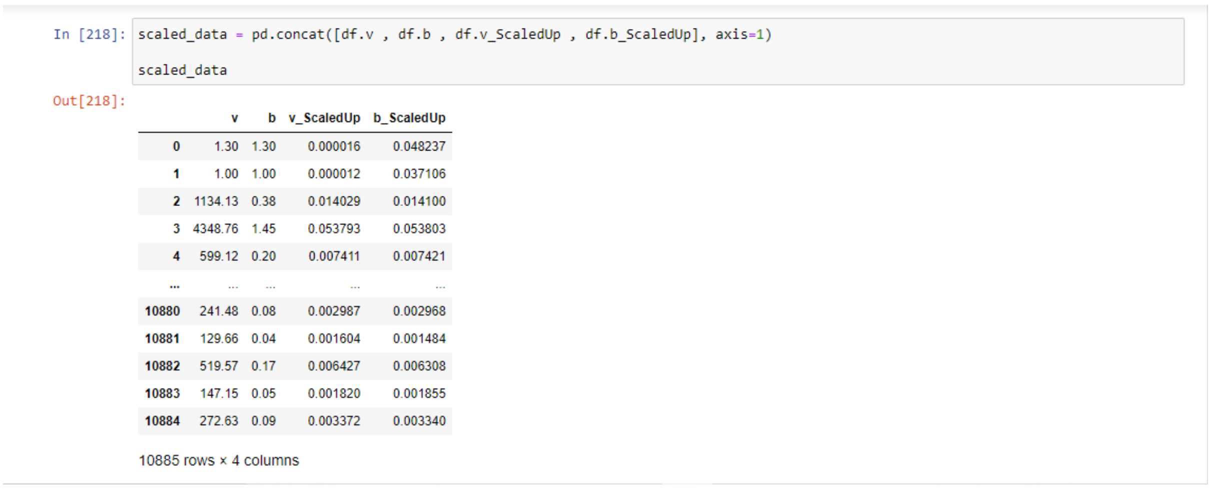


```
In [215]: df.groupby("complexityEvaluation").size()

Out[215]: complexityEvaluation
          Redesign      1725
          Succesful     9160
          dtype: int64

In [216]: trace = go.Histogram(
              x = df.complexityEvaluation,
              opacity = 0.75,
              name = 'Complexity Evaluation',
              marker = dict(color = 'darkorange')
          )

          hist_data = [trace]
          hist_layout = go.Layout(barmode='overlay',
                             title = 'Complexity Evaluation',
                             xaxis = dict(title = 'Succesful - Redesign'),
                             yaxis = dict(title = 'Frequency')
          )

          fig = go.Figure(data = hist_data, layout = hist_layout)
          iplot(fig)
```



Data Normalization (Min-Max Scaler)


```
In [217]: #Data Normalization

          from sklearn import preprocessing

          scale_v = df[['v']]
          scale_b = df[['b']]

          minmax_scaler = preprocessing.MinMaxScaler()

          v_scaled = minmax_scaler.fit_transform(scale_v)
          b_scaled = minmax_scaler.fit_transform(scale_b)

          df['v_ScaledUp'] = pd.DataFrame(v_scaled)
          df['b_ScaledUp'] = pd.DataFrame(b_scaled)

          df
```

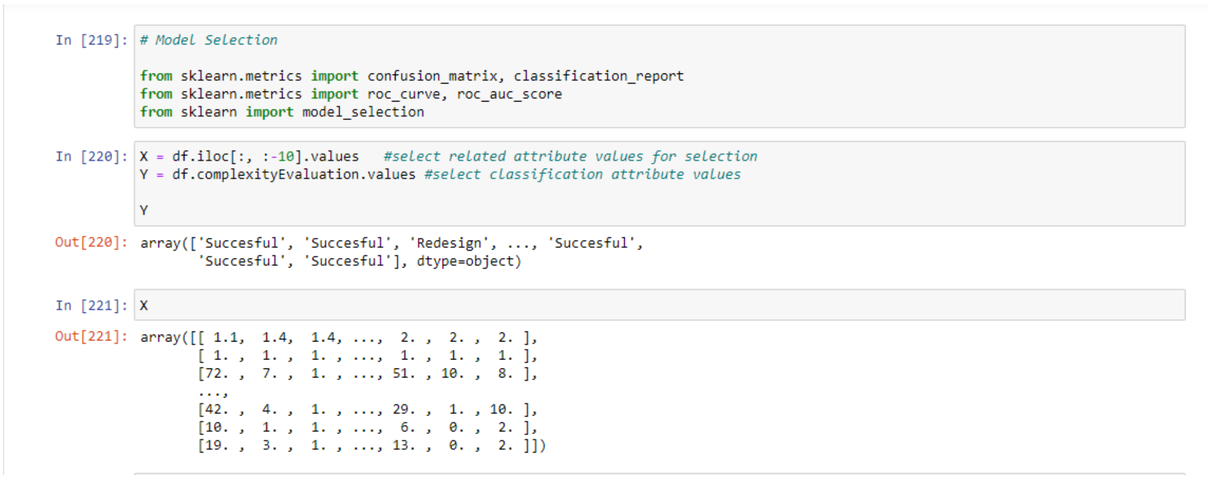

```
In [218]: scaled_data = pd.concat([df.v , df.b , df.v_ScaledUp , df.b_ScaledUp], axis=1)

          scaled_data

Out[218]:
```

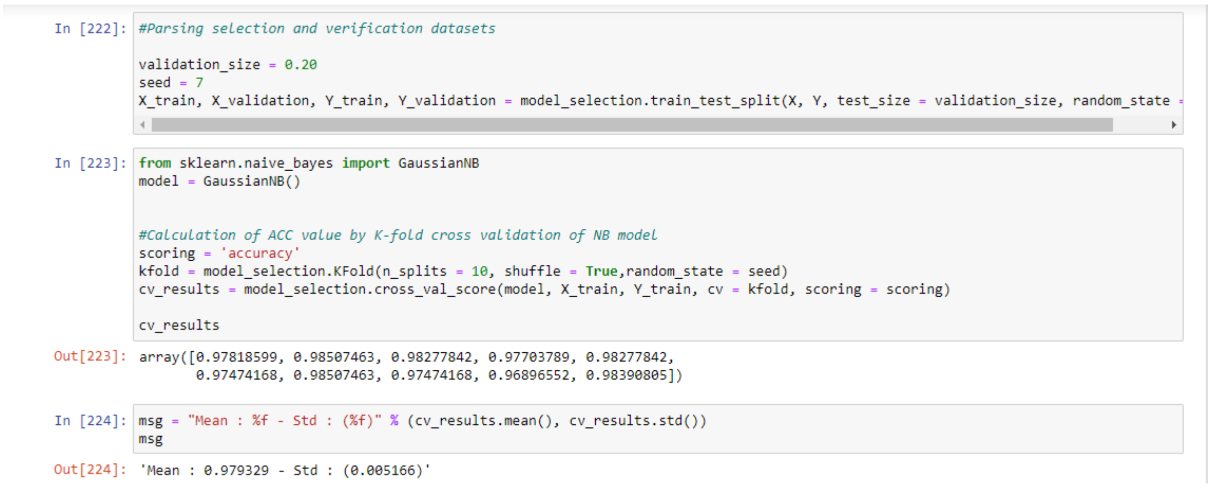
	v	b	v_ScaledUp	b_ScaledUp
0	1.30	1.30	0.000016	0.048237
1	1.00	1.00	0.000012	0.037106
2	1134.13	0.38	0.014029	0.014100
3	4348.76	1.45	0.053793	0.053803
4	599.12	0.20	0.007411	0.007421
...	...	...	...	...
10880	241.48	0.08	0.002987	0.002968
10881	129.66	0.04	0.001604	0.001484
10882	519.57	0.17	0.006427	0.006308
10883	147.15	0.05	0.001820	0.001855
10884	272.63	0.09	0.003372	0.003340

10885 rows × 4 columns

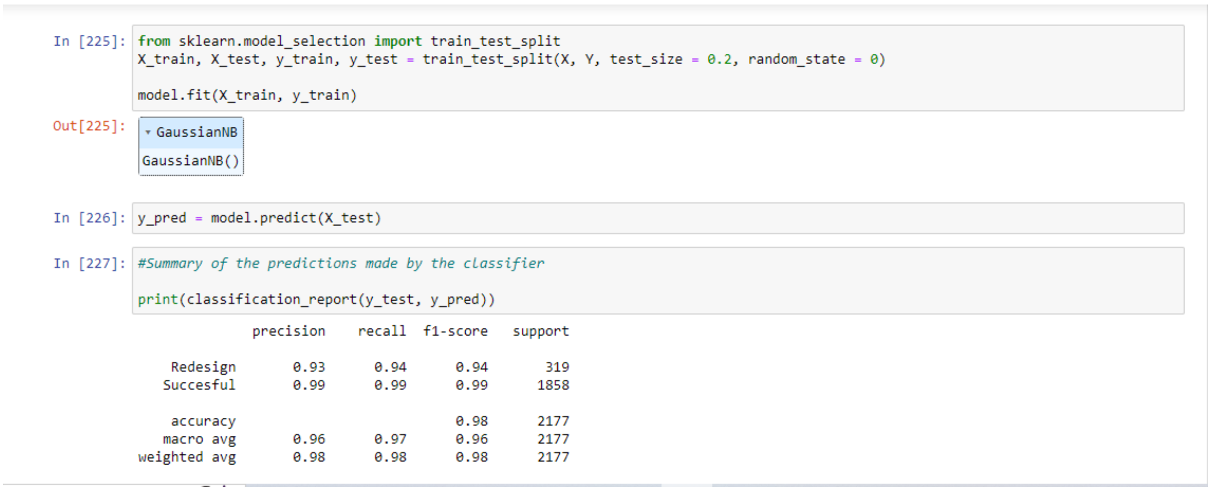
Model Selection

Naive Bayes:

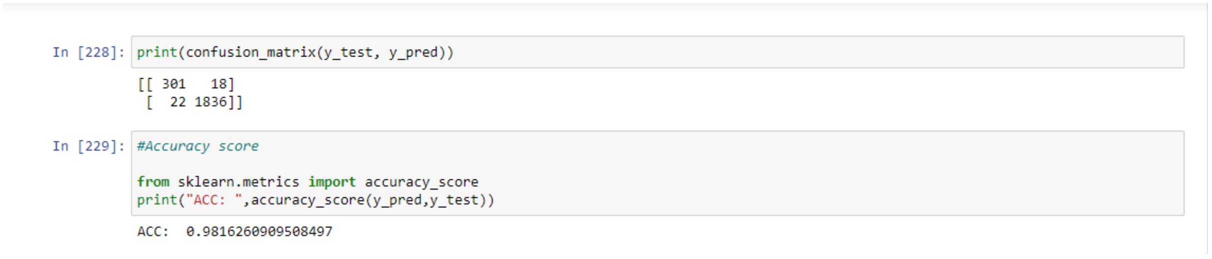

```
In [219]: # Model Selection

          from sklearn.metrics import confusion_matrix, classification_report
          from sklearn.metrics import roc_curve, roc_auc_score
          from sklearn import model_selection

In [220]: X = df.iloc[:, :-10].values   #select related attribute values for selection
          Y = df.complexityEvaluation.values #select classification attribute values

          Y

Out[220]: array(['Succesful', 'Succesful', 'Redesign', ..., 'Succesful',
                 'Succesful', 'Succesful'], dtype=object)

In [221]: X

Out[221]: array([[ 1.1,  1.4,  1.4, ...,  2. ,  2. ,  2. ],
                 [ 1. ,  1. ,  1. , ...,  1. ,  1. ,  1. ],
                 [72. ,  7. ,  1. , ..., 51. , 10. ,  8. ],
                 ...,
                 [42. ,  4. ,  1. , ..., 29. ,  1. , 10. ],
                 [10. ,  1. ,  1. , ...,  6. ,  0. ,  2. ],
                 [19. ,  3. ,  1. , ..., 13. ,  0. ,  2. ]])
```



```
In [222]: #Parsing selection and verification datasets

          validation_size = 0.20
          seed = 7
          X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size = validation_size, random_state =

In [223]: from sklearn.naive_bayes import GaussianNB
          model = GaussianNB()


          #Calculation of ACC value by K-fold cross validation of NB model
          scoring = 'accuracy'
          kfold = model_selection.KFold(n_splits = 10, shuffle = True,random_state = seed)
          cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv = kfold, scoring = scoring)

          cv_results

Out[223]: array([0.97818599, 0.98507463, 0.98277842, 0.97703789, 0.98277842,
                 0.97474168, 0.98507463, 0.97474168, 0.96896552, 0.98390805])

In [224]: msg = "Mean : %f - Std : (%f)" % (cv_results.mean(), cv_results.std())
          msg

Out[224]: 'Mean : 0.979329 - Std : (0.005166)'
```

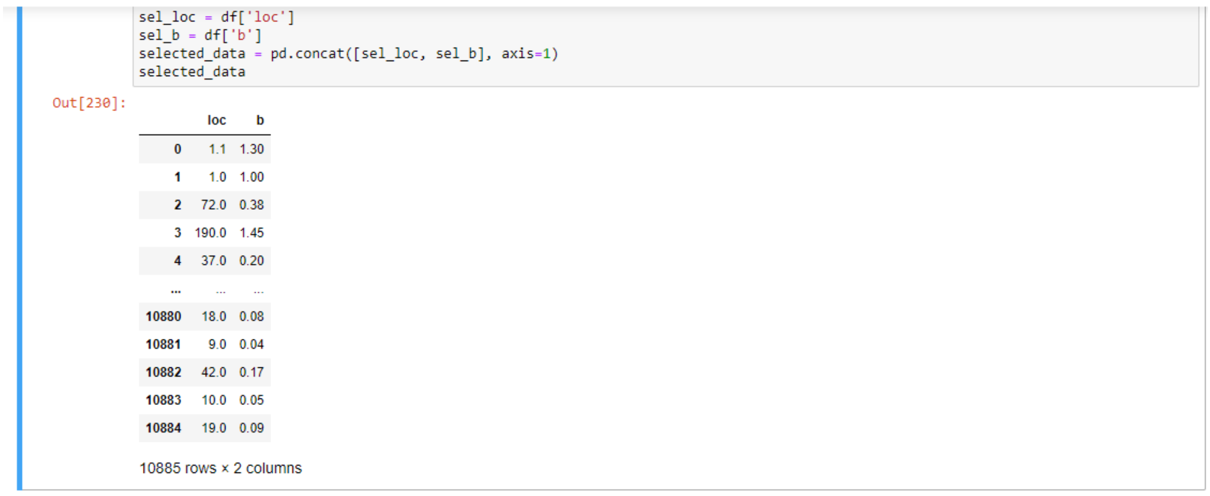


```
In [225]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

          model.fit(X_train, y_train)

Out[225]: ▾ GaussianNB
          GaussianNB()

In [226]: y_pred = model.predict(X_test)

In [227]: #Summary of the predictions made by the classifier

          print(classification_report(y_test, y_pred))

                        precision    recall  f1-score   support

              Redesign       0.93      0.94      0.94       319
             Succesful       0.99      0.99      0.99      1858

              accuracy                           0.98      2177
             macro avg       0.96      0.97      0.96      2177
          weighted avg       0.98      0.98      0.98      2177
```



```
In [228]: print(confusion_matrix(y_test, y_pred))

          [[ 301   18]
           [  22 1836]]

In [229]: #Accuracy score

          from sklearn.metrics import accuracy_score
          print("ACC: ",accuracy_score(y_pred,y_test))

          ACC:  0.9816260909508497
```

Linear Regression:

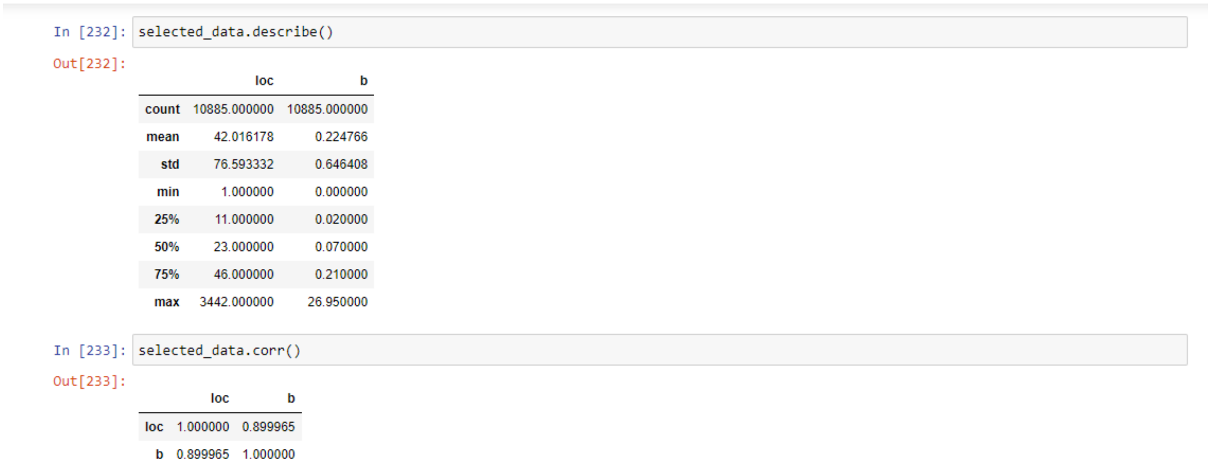


```
sel_loc = df['loc']
sel_b = df['b']
selected_data = pd.concat([sel_loc, sel_b], axis=1)
selected_data
```

Out[230]:

	loc	b
0	1.1	1.30
1	1.0	1.00
2	72.0	0.38
3	190.0	1.45
4	37.0	0.20
...	...	...
10880	18.0	0.08
10881	9.0	0.04
10882	42.0	0.17
10883	10.0	0.05
10884	19.0	0.09

10885 rows × 2 columns



In [232]:
```
selected_data.describe()
```

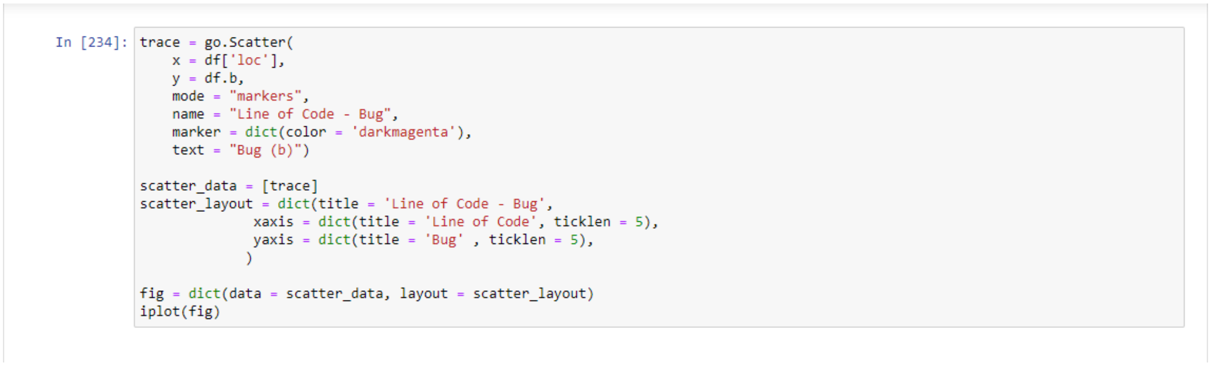
Out[232]:

	loc	b
count	10885.000000	10885.000000
mean	42.016178	0.224766
std	76.593332	0.646408
min	1.000000	0.000000
25%	11.000000	0.020000
50%	23.000000	0.070000
75%	46.000000	0.210000
max	3442.000000	26.950000

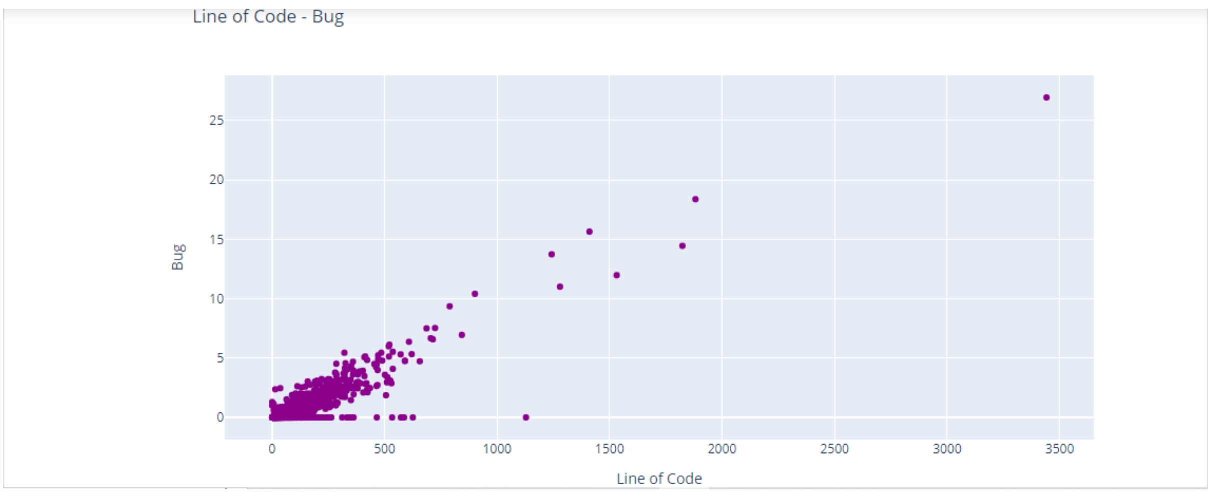
In [233]:
```
selected_data.corr()
```

Out[233]:

	loc	b
loc	1.000000	0.899965
b	0.899965	1.000000



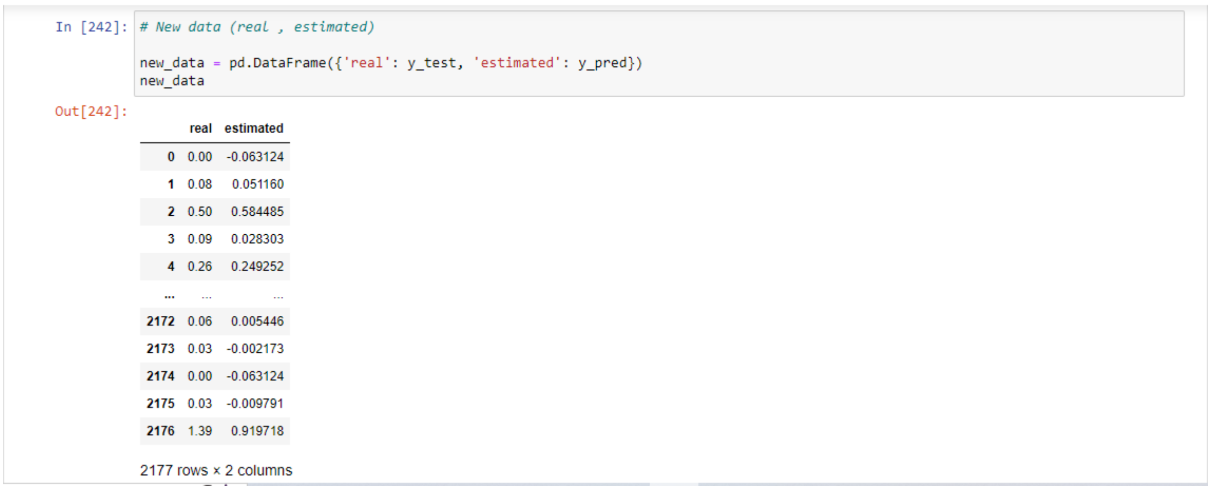
In [234]:
```
trace = go.Scatter(
    x = df['loc'],
    y = df.b,
    mode = "markers",
    name = "Line of Code - Bug",
    marker = dict(color = 'darkmagenta'),
    text = "Bug (b)")

scatter_data = [trace]
scatter_layout = dict(title = 'Line of Code - Bug',
              xaxis = dict(title = 'Line of Code', ticklen = 5),
              yaxis = dict(title = 'Bug' , ticklen = 5),
             )

fig = dict(data = scatter_data, layout = scatter_layout)
iplot(fig)
```



```
In [235]: Y = selected_data['b'].values
          X = selected_data['loc'].values
          X = X.reshape(-1,1)
          |
          Y

Out[235]: array([1.3 , 1.  , 0.38, ..., 0.17, 0.05, 0.09])

In [236]: X

Out[236]: array([[ 1.1],
                 [ 1. ],
                 [72. ],
                 ...,
                 [42. ],
                 [10. ],
                 [19. ]])
```

```
In [237]: #Parsing selection and verification datasets

          from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

In [238]: #Creation of Linear Regression model

          from sklearn.linear_model import LinearRegression
          model = LinearRegression()
          model.fit(X_train, y_train)

Out[238]: ▾ LinearRegression
          LinearRegression()

In [239]: print("Intercept :", model.intercept_)
          print("Coef :", model.coef_)

          Intercept : -0.09359968647139849
          Coef : [0.00761893]
```

```
In [240]: X_test

Out[240]: array([[  4.],
                 [ 19.],
                 [ 89.],
                 ...,
                 [  4.],
                 [ 11.],
                 [133.]])

In [241]: y_pred = model.predict(X_test)
          y_pred

Out[241]: array([-0.06312398,  0.05115992,  0.58448478, ..., -0.06312398,
                 -0.00979149,  0.91971754])
```

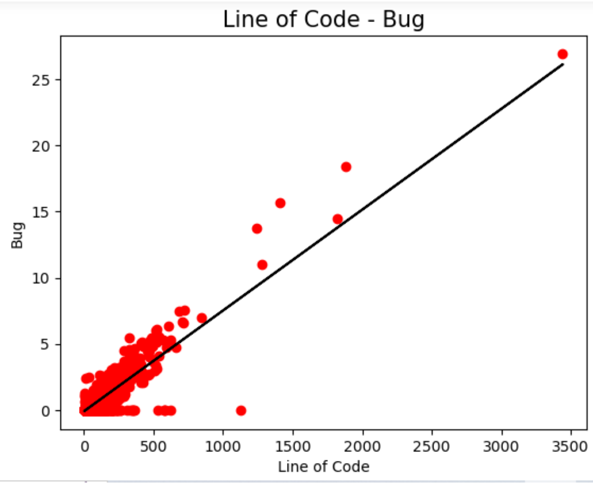


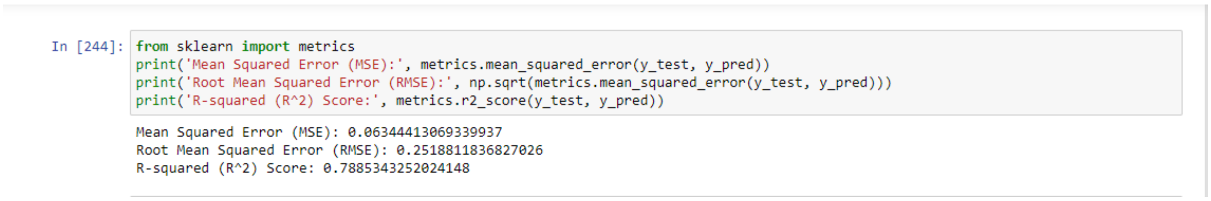
```
In [242]: # New data (real , estimated)

          new_data = pd.DataFrame({'real': y_test, 'estimated': y_pred})
          new_data
```

Out[242]:

	real	estimated
0	0.00	-0.063124
1	0.08	0.051160
2	0.50	0.584485
3	0.09	0.028303
4	0.26	0.249252
...	...	...
2172	0.06	0.005446
2173	0.03	-0.002173
2174	0.00	-0.063124
2175	0.03	-0.009791
2176	1.39	0.919718

2177 rows × 2 columns





```
In [244]: from sklearn import metrics
          print('Mean Squared Error (MSE):', metrics.mean_squared_error(y_test, y_pred))
          print('Root Mean Squared Error (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
          print('R-squared (R^2) Score:', metrics.r2_score(y_test, y_pred))

          Mean Squared Error (MSE): 0.06344413069339937
          Root Mean Squared Error (RMSE): 0.2518811836827026
          R-squared (R^2) Score: 0.7885343252024148
```

5.9 Issues with the training data set

There are several common issues that can be encountered in a machine learning project.

1. Data Quality Issues:

- Missing Values: One common issue is missing data. If there are missing values in the dataset, it can lead to problems during training and evaluation. These missing values need to be handled, either by imputation or by removing rows or columns with missing data.
- Outliers: Outliers are data points that significantly differ from the majority of the data. They can skew statistical measures and affect the performance of some machine learning algorithms. Outlier detection and treatment are important steps in data preprocessing.
- Incorrect Data Types: Data columns may have incorrect data types (e.g., categorical variables represented as numerical or vice versa). Ensuring that data types are appropriate is crucial.

2. Data Imbalance:

- Class Imbalance: If you're working on a classification problem and one class significantly outnumbers the others, it can lead to a biased model. Techniques like oversampling, undersampling, or using different evaluation metrics may be needed to address this issue.

3. Feature Engineering:

- Feature Selection: The dataset may contain irrelevant or redundant features that don't contribute to the model's predictive power. Feature selection methods can help identify and remove such features.
- Feature Scaling: Depending on the algorithm used, features may need to be scaled or normalized to ensure that no single feature dominates the learning process.

4. Target Variable Issues:

- Data Leakage: Make sure that the target variable (in this case, 'complexityEvaluation') is not influenced by features that would not be available at the time of prediction. Data leakage can lead to overly optimistic model performance.
- Encoding: Ensure that the target variable is properly encoded, especially if it's a categorical variable.

5. Data Exploration:

- Data Skewness: Check whether the data distributions are skewed. Some machine learning algorithms may perform poorly on heavily skewed data. Transformation techniques like log transforms can be applied if needed.
- Correlations: Understand the correlations between features. High multicollinearity (correlations between predictors) can cause problems in some models.

6. Dataset Size:

- The size of the training dataset can impact model performance. If the dataset is very small, it may lead to overfitting. If it's too large, training times can become a bottleneck.

7. Validation and Cross-Validation:

- Ensure that proper validation and cross-validation procedures are in place to assess model performance. Cross-validation helps estimate how well the model will generalize to unseen data.

8. Domain-specific Issues:

- Depending on the domain of the data, there may be specific issues to consider. For example, in medical data, handling missing values and imbalanced classes can be critical.

6. Conclusion

.

The conclusion of a software defect prediction analysis can vary depending on the specific goals and findings of the analysis. However, here are some common conclusions and key takeaways that can be drawn from such an analysis:

1. Identification of High-Risk Areas: Software defect prediction helps in identifying the modules, components, or code segments that are at higher risk of containing defects. This information is valuable for prioritizing testing efforts and allocating resources effectively.
2. Improved Software Quality: By focusing testing and quality assurance efforts on the predicted high-risk areas, organizations can potentially reduce the number of defects in their software, leading to higher software quality and improved customer satisfaction.
3. Resource Allocation: Defect prediction analysis helps in making informed decisions about resource allocation. It allows organizations to allocate testing resources where they are most needed, optimizing their testing efforts and reducing overall testing costs.
4. Early Detection and Prevention: Identifying potential defect-prone areas early in the development process enables proactive measures to be taken to prevent defects from occurring in the first place. This can result in significant cost savings and time-to-market improvements.
5. Continuous Improvement: Software defect prediction is not a one-time task. It can be used as part of a continuous improvement process, allowing organizations to refine their development and testing practices over time based on historical defect data and predictive models.

6. Model Accuracy and Validation: It's crucial to assess the accuracy of the defect prediction models used in the analysis. This involves evaluating how well the models perform in practice and whether they provide meaningful insights.
7. Data Quality and Availability: The success of defect prediction analysis relies heavily on the quality and availability of historical data. Ensuring that the data used for analysis is accurate, complete, and representative of the software development process is essential.
8. Collaboration and Communication: Effective communication of the analysis results to relevant stakeholders, such as development teams and project managers, is essential. Collaboration between different teams can help in translating the insights gained from the analysis into actionable strategies for defect prevention and mitigation.
9. Risk Mitigation Strategies: The analysis may lead to the development of specific strategies and best practices for mitigating defects in high-risk areas. These strategies can include code reviews, additional testing, process improvements, and training.
10. Monitoring and Feedback: After implementing defect prediction strategies, it's important to monitor their effectiveness over time. Gathering feedback and continuously refining the analysis and mitigation strategies is crucial for ongoing improvement.

In conclusion, software defect prediction analysis is a valuable technique for identifying and mitigating potential defects in software development projects. It can lead to better resource allocation, improved software quality, and more efficient development processes when used effectively. However, it's essential to continuously assess and adapt the analysis and mitigation strategies to the evolving needs of the organization and the software being developed.

7. References

https://www.kaggle.com/code/semustafacevik/software-defect-prediction-data-analysis

***