Download two datasets: JDT.csv and PDE.csv. They are bug datasets of two projects JDT and PDE. Two datasets having the same column names:

**file**: name of the class in this project. This column should be used as row names in R.

**bug**: number of post-release bugs found in that class. If bug = 0 then the class is considered defect free. Otherwise, it is defect prone.

**bf**: number of previous bug fixes on this class.

**loc**: number of lines of code of this class.

**hcm**: a measurement of the complexity of the changes applied to this class in its history.

To answer the below questions, we need to set the directory where the “JDT.csv”, and “PDE.csv” files are located.

In my local machine, the code to set the working directory is

setwd(“C:/Users/cscha/OneDrive/Desktop/Software analytics”)

Now read the “JDT.csv” file by typing the following code

> data1 <- read.csv(“JDT.csv”)

Here, “data1” works as a variable to perform operations on the “JDT.csv” file

Now read the “PDE.csv” file by typing the following code

> data2 <- read.csv(“PDE.csv”)

Here, “data2” works as a variable to perform operations on the “PDE.csv” file

Q1. For each dataset, calculate the ranked (Spearman) correlation between **bug** and other variables. Are they strongly or weakly correlated? What variable has the strongest correlation to **bug**?

The ‘ R’ code to calculate the ranked(spearman) correlation between bug and other variables is as follows:

For the dataset of JDT, it is:

> data1\_correlation <- cor(data1[, c("bug", "bf", "loc", "hcm")], method = "spearman")

>print(data1\_correlation)

The output is:

bug bf loc hcm

bug 1.0000000 0.4492973 0.4089799 0.4531625

bf 0.4492973 1.0000000 0.7269227 0.8384459

loc 0.4089799 0.7269227 1.0000000 0.6446473

hcm 0.4531625 0.8384459 0.6446473 1.0000000

In the above line of code

data1\_correlation <- cor(data1[, c("bug", "bf", "loc", "hcm")], method = "spearman"),

For the JDT dataset, it estimates the Spearman correlation. To determine the correlation between the columns "bug," "bf," "loc," and "hcm" in the JDT.csv data, we utilize the cor function and the "Spearman" approach. The

data1\_correlation variable holds the generated correlation matrix.

From the above output, we can state that

The correlation between "bug" and "bf" is moderately positive, with a correlation of about 0.449.

"Bug" and "loc" have a moderately positive association of about 0.409, which is calculated.

The correlation between "bug" and "hcm" is moderately positive, with a correlation of about 0.453.

These correlation coefficients reveal a relatively positive association between the "bug" variable in data1 and the variables "bf," "loc," and "hcm” and conclude that they are not too weakly correlated.

So from the above output when the bug is compared to other variables in terms of correlation, the correlation between the bug and hcm is greater than other variables which hold the value of 0.45 leaving the correlation of the bug to itself which always stays as 1.

For the dataset of PDE, it is:

> data2\_correlation <- cor(data2[, c("bug", "bf", "loc", "hcm")], method = "spearman")

> print(data2\_correlation)

In the above line of code

data2\_correlation <- cor(data2[, c("bug", "bf", "loc", "hcm")], method = "spearman"),

For the PDE dataset, it estimates the Spearman correlation. To determine the correlation between the columns "bug," "bf," "loc," and "hcm" in the PDE.csv data, we utilize the cor function and the "Spearman" approach. The

data2\_correlation variable holds the generated correlation matrix.

The output is:  
bug bf loc hcm

bug 1.0000000 0.2895665 0.2663918 0.2617094

bf 0.2895665 1.0000000 0.5578313 0.6514259

loc 0.2663918 0.5578313 1.0000000 0.4856577

hcm 0.2617094 0.6514259 0.4856577 1.0000000

From the above output, we can state that

Indicating a weak positive association, the correlation between "bug" and "bf" is roughly 0.290.

Indicating a loose positive association, the correlation between "bug" and "loc" is roughly 0.266.

Indicating a poor positive association, the correlation between "bug" and "hcm" is roughly 0.262.

So as the correlation values are roughly near 0.3 in each case when the bug is associated with other variables, we state that they are weakly correlated. In these produced correlation values, the correlation between bug and bf stays high with the value of 0.28 when compared to the correlation with bug to loc and bug to hcm leaving the correlation of the bug to itself which always stays as 1.

In summary, we can conclude that the "bug" variable is highly positively connected with other factors in JDT.csv,

however, this association is weaker and falls into the "weak positive" range in PDE.csv data.

Q2. What is the percentage of defect-free classes in each project? What is the percentage of bugs found in the top 10% and top 20% most defect-prone classes among bugs found in all classes?

From the analysis, can we state that *"most post-release bugs occur in a small portion of code"*? In other words, the distribution of post-release bugs is pretty skew.

The ‘R’ code to calculate the defect-free classes in the JDT.csv dataset is as follows:

> data1\_defect\_free <- sum(data1$bug ==0)

> data1\_total\_classes <- nrow(data1)

> data1\_defect\_free\_percentage <- (data1\_defect\_free / data1\_total\_classes) \* 100

> print(data1\_defect\_free\_percentage)

[1] 79.33801

Here in the above code,

data1\_defect\_free == sum (data1$bug == 0): performs how many times does the data1 dataset's bug column contain a value of 0? and in a variable called data1\_defect\_free, will keep track of this number.

data1\_total\_classes = nrow(data1), this line of code will track how many classes (or rows) are present in the data1

dataset.

The line of code: data1\_defect\_free\_percentage = (data1\_defect\_free / data1\_total\_classes) \* 100:

Determine the proportion of classes in the dataset that are free of defects.

To get the percentage, multiply the number of courses with no defects by the total number of classes.

data1\_defect\_free\_percentage is a variable where this percentage should be kept

This code just counts the number of classes in the dataset that are defect-free and expresses that number as a percentage of all the classes in the dataset.

The percentage of defect-free classes in data1 is 79.33801

The ‘R’ code to calculate the defect-free classes in the PDE.csv dataset is as follows:

> data2\_defect\_free <- sum(data2$bug ==0)

> data2\_total\_classes <- nrow(data2)

> data2\_defect\_free\_percentage <- (data2\_defect\_free / data2\_total\_classes) \* 100

> print(data2\_defect\_free\_percentage)

[1] 86.03874

Here in the above code,

data2\_defect\_free == sum (data2$bug == 0): performs how many times does the data2 dataset's bug column contain a value of 0? and in a variable called data2\_defect\_free, will keep track of this number.

data2\_total\_classes = nrow(data2), this line of code will track how many classes (or rows) are present in the data2

dataset.

The line of code: data2\_defect\_free\_percentage = (data2\_defect\_free / data2\_total\_classes) \* 100:

Determine the proportion of classes in the dataset that are free of defects.

To get the percentage, multiply the number of courses with no defects by the total number of classes.

Data2\_defect\_free\_percentage is a variable where this percentage should be kept

This code just counts the number of classes in the dataset that are defect-free and expresses that number as a percentage of all the classes in the dataset.

The percentage of defect-free classes in data1 is 86.03874

The ‘R’ code to calculate the percentage of bugs found in the top 10% and top 20% of most defect-prone classes among bugs found in all classes is as follows:

The ‘R’ code to calculate the top 10% of the most-defect-prone-classes of JDT.csv is as follows:

>library(dplyr)

> top10\_data1 <- data1 %>%

+ arrange(desc(bug)) %>%

+ slice (1: round (0.1\*data1\_total\_classes))

> data1\_total\_bugs <- sum(data1$bug)

> top10\_bugs <- sum(top10\_data1$bug)

> top10\_bugs\_percentage <- (top10\_bugs / data1\_total\_bugs) \* 100

> print(top10\_bugs\_percentage)

[1] 71.65775

The ‘R’ code to calculate the top 20% of the most-defect-prone-classes of JDT.csv is as follows:

> top20\_data1 <- data1 %>%

+ arrange(desc(bug)) %>%

+ slice (1: round (0.2\*data1\_total\_classes))

> top20\_bugs <- sum(top20\_data1$bug)

> top20\_bugs\_percentage <- (top20\_bugs / data1\_total\_bugs) \* 100

> print(top20\_bugs\_percentage)

[1] 98.12834

The ‘R’ code to calculate the top 10% of the most-defect-prone-classes of PDE.csv is as follows:

> top10\_data2 <- data2 %>%

+ arrange(desc(bug)) %>%

+ slice(1:round(0.1\* data2\_total\_classes))

> data2\_total\_bugs <- sum(data2$bug)

> top10\_bugs\_data2 <- sum(top10\_data2$bug)

> top10\_bugs\_percentage\_data2 <- (top10\_bugs\_data2 / data2\_total\_bugs) \* 100

> print(top10\_bugs\_percentage\_data2)

[1] 82.69795

The ‘R’ code to calculate the top 20% of the most-defect-prone-classes of PDE.csv is as follows:

> top20\_data2 <- data2 %>%

+ arrange(desc(bug)) %>%

+ slice(1:round(0.2\* data2\_total\_classes))

> data2\_total\_bugs <- sum(data2$bug)

> top20\_bugs\_data2 <- sum(top20\_data2$bug)

> top20\_bugs\_percentage\_data2 <- (top20\_bugs\_data2 / data2\_total\_bugs) \* 100

> print(top20\_bugs\_percentage\_data2)

[1] 100

From the above output, we can state that

The top 10% of classes (those with the greatest amount of flaws) account for roughly 71.66% of all the bugs in the dataset, according to your analysis.

The top 20% of classes are accountable for roughly 98.13% of the overall number of problems.

This shows that the bulk of post-release defects in JDT.csv are caused by a relatively small section of the codebase (between 10% and 20% of classes).

They account for about 82.70% of all defects in the dataset when the top 10% of classes are taken into account.

All the bugs are concentrated in the top 20% of classes, as shown by the fact that they account for 100% of all bugs when the top 20% of classes are taken into account.

This points to a more pronounced skew in the distribution of post-release issues in PDE.csv, where a tiny fraction of the codebase (between 10% and 20% of classes) has a preponderant number of bugs.

Q3. Use boxplot and t.test to compare the distribution of bugs in two datasets JDE and PDE. Are they having similar bug distributions? Or what project is more defect-prone (i.e. its classes often have most post-release bugs)?

The ‘R’ code to perform boxplot and visualize the distribution of bugs over the JDT and PDE datasets is as follows:

boxplot(data1$bug, data2$bug, names=c("JDT","PDE"), main="Bugs Distribution in JDT and PDE", ylab="Count of Bugs")

Output:

A screenshot of a computer screen

Description automatically generated

Overall, the box plot indicates that the "JDE" dataset has a more condensed distribution with fewer extreme outliers than the "PDE" dataset, where there is higher fluctuation in bug counts with some classes having a very high number of problems.

The R code for the t-test is as follows:

> t\_test <- t.test(data1$bug, data2$bug)

> print(t\_test)

Welch Two Sample t-test

data: data1$bug and data2$bug

t = 3.5757, df = 2007.2, p-value = 0.0003576

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.06652755 0.22814538

sample estimates:

mean of x mean of y

0.3751254 0.2277889

From the above results of the t-test, we can state that the p-value is 0.0003 which is less than the significance p-value which is 0.05. This shows that the means of the two datasets differ statistically significantly.

Q4. For each class, the total number of bugs is bug + bf. The defect density is the number of bugs in each KLOC (a thousand lines of code). So, defect density = (bug + bf)/loc \* 1000.

Use boxplot and t.test to compare the distribution of defect density in two datasets JDE and PDE. Are they having similar distributions? Or what has higher density?

The ‘R’ code for box plot and t-test comaprision is as follows:

> data1$defect\_desnity <- ((data1$bug + data1$bf) / data1$loc) \* 1000

> data2$defect\_desnity <- ((data2$bug + data2$bf) / data2$loc) \* 1000

> data1 <- data1[is.finite(data1$defect\_density), ]

> data2 <- data2[is.finite(data2$defect\_density), ]

> boxplot(data1$defect\_density, data2$defect\_density, names = c("JDT", "PDE"),

+ ylab = "Defect Density(bugs per KLOC)", main = "Distribution of Defect Density")

> t\_test\_output <- t.test(data1$defect\_density, data2$defect\_density)

> print(t\_test\_output)

In the above code

The defect densities for two datasets, which represent the density of respective defects in each dataset such as JDT.csv and PDE.csv, are constructed as data1$defect\_density and data2$defect\_density.

The formula for calculating defect density is (bugs + bf) / loc \* 1000, where bugs, bf, and loc are the dataset’s respective columns.

To remove rows where the defect density is non-finite, data1 and data2 are filtered using is.finite().

To compare the defect density distribution for the two datasets (data1 and data2), a boxplot is made.

The two datasets are given the labels "JDT" and "PDE" by the names parameter.

The title of the graph is "Distribution of Defect Density" and the y-axis label is "Defect Density (bugs per KLOC)".

To statistically assess the defect density between the two datasets (data1 and data2), a two-sample t-test is used.

The t\_test\_output variable holds the t-test's outcome.

Using print(t\_test\_output), the t-test results, which include statistics like the t-value, degrees of freedom, and p-value, are displayed on the console.

In summary, this code calculates the defect density for two datasets, plots their distributions using a boxplot for visualization, and comparison, and uses a t-test to see if there is a statistically significant difference between the two datasets' defect densities.

The output of the box plot is as follows:

A screenshot of a computer

Description automatically generated

As a result of box plot visualization, we can draw the conclusion that the defect density distribution between JDT

and PDE differs significantly.

The output of the t-test is as follows:

Welch Two Sample t-test

data: data1$defect\_density and data2$defect\_density

t = 4.3554, df = 1209.9, p-value = 1.441e-05

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

25.36038 66.93640

sample estimates:

mean of x mean of y

111.82765 65.67926

In conclusion, the t-test findings show that there is a statistically significant difference between the defect densities of the two datasets provided the p-value, which is 1.441e-05, is considered.

Q5. Explain why in defect prediction, it is more important to identify the most defect-prone code than making the prediction with smallest average errors? If we use a linear regression model to predict the number of post-release bugs, what measurement of predictive power (or prediction accuracy) should we use?

In defect prediction, identifying the most defect-prone code was given first priority over the way of reducing average prediction errors for some aspects.

First of all, identifying high-risk code enables efficient testing and resource allocation, improving software quality. Second, it lessens risks related to flaws and mitigates important problems. Thirdly, it makes it easier to prioritize strategically for code inspection and upkeep. Finally, it stops technical debt from building up.

When using a linear regression model to predict defects, measuring predictive power should, for example, concentrate on metrics like R-squared to assess the proportion of explained variance. This metric measures how well the model captures underlying patterns, which is essential for efficient defect identification and risk mitigation.

The other metrics that are used are Mean Absolute Error(MAE), Root Mean Squared Error(RMSE) and Mean Squared Error(MSE). Your regression model's specific objectives and specifications will determine the best metric to use. R-squared provides information on the general goodness of fit of the model, whereas MAE, RMSE, and MSE offer more specific details on the prediction errors. The effectiveness of a linear regression model for forecasting the quantity of post-release problems is frequently evaluated thoroughly using a combination of these criteria.