**Assignment 2**

**Project Objectives and Approach**

The primary goal of this project is to assess and improve the quality of object-oriented design and maintainability in selected Java-based repositories. This entails a thorough analysis of the code structure, with a focus on identifying potential areas for improvement through the application of CK metrics and PMD code smells detection. By evaluating aspects such as complexity, inheritance, coupling, cohesiveness, and code smells, the project aims to offer actionable insights and recommendations for enhancing code quality.

To achieve these objectives, the project is guided by the Goal-Question-Metric (GQM) approach, addressing key questions related to each objective. For instance, to assess class complexity and maintainability, the project will examine the complexity of individual classes and its impact on maintainability. Inheritance and coupling will be evaluated by analyzing the depth of inheritance, number of subclasses, and coupling levels. Cohesion will be determined by investigating the cohesiveness of methods within classes and identifying factors leading to lack of cohesion. Code smells will be identified and quantified to guide refactoring efforts, and actionable recommendations will be provided based on the findings. Metrics such as Weighted Methods per Class (WMC), Response for a Class (RFC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Lack of Cohesion of Methods (LCOM), and various code smell metrics will be collected and analyzed to support this comprehensive assessment. Through this systematic approach, the project aims to enhance the overall quality and maintainability of the selected Java codebases.

**Dataset**

Following repositories from Github are downloaded and the CK metrics and PMD reports are generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **URL** | **Stars** | **Size** | **No.of Commits** |
| Shardingsphere | <https://github.com/apache/shardingsphere> | 481547 | 33758 | 342 |
| spring-boot | <https://github.com/spring-projects/spring-boot> | 61593 | 151778 | 38233 |
| RxJava | <https://github.com/ReactiveX/RxJava> | 46134 | 134465 | 5988 |
| **Dubbo** | <https://github.com/apache/dubbo> | 37385 | 39817 | 5496 |
| Glide | <https://github.com/bumptech/glide> | 32646 | 91776 | 2643 |
| Fresco | <https://github.com/facebook/fresco> | 16798 | 74299 | 3195 |
| NewPipe | <https://github.com/TeamNewPipe/NewPipe> | 19424 | 60806 | 9978 |
| Signal-Android | <https://github.com/signalapp/Signal-Android> | 22462 | 609700 | 9959 |
| jadx-master | https://github.com/skylot/jadx | 30429 | 18789 | 1766 |
| Easyexcel | <https://github.com/alibaba/easyexcel> | 23791 | 26031 | 872 |

**Tools used**

PMD and CK tools are used for this empirical study

PMD tool is downloaded from <https://docs.pmd-code.org/latest/index.html>

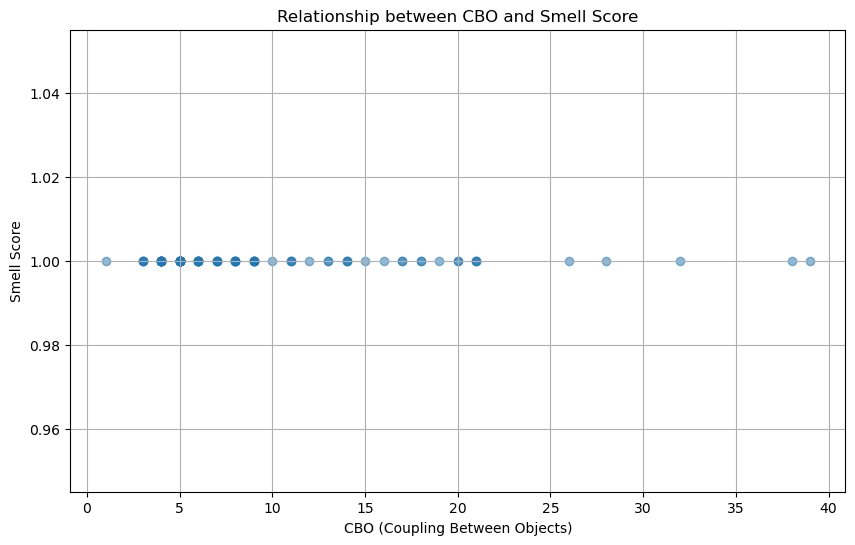
CK tool is downloaded from https://github.com/mauricioaniche/ck

**Results and Analysis**

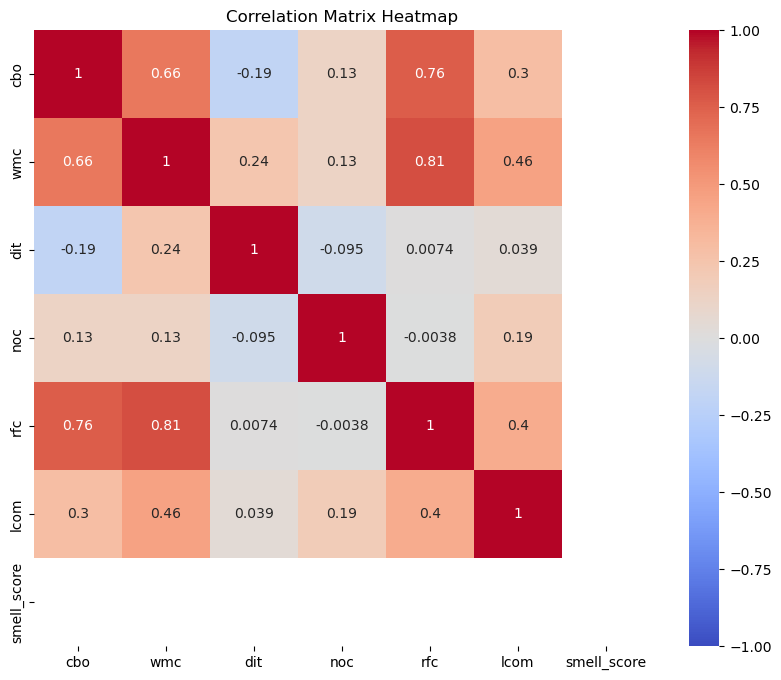
Relation between the CK metrics and the PMD report-based smells for all the repositories is compared and analyzed here and given below

**EasyExcel**

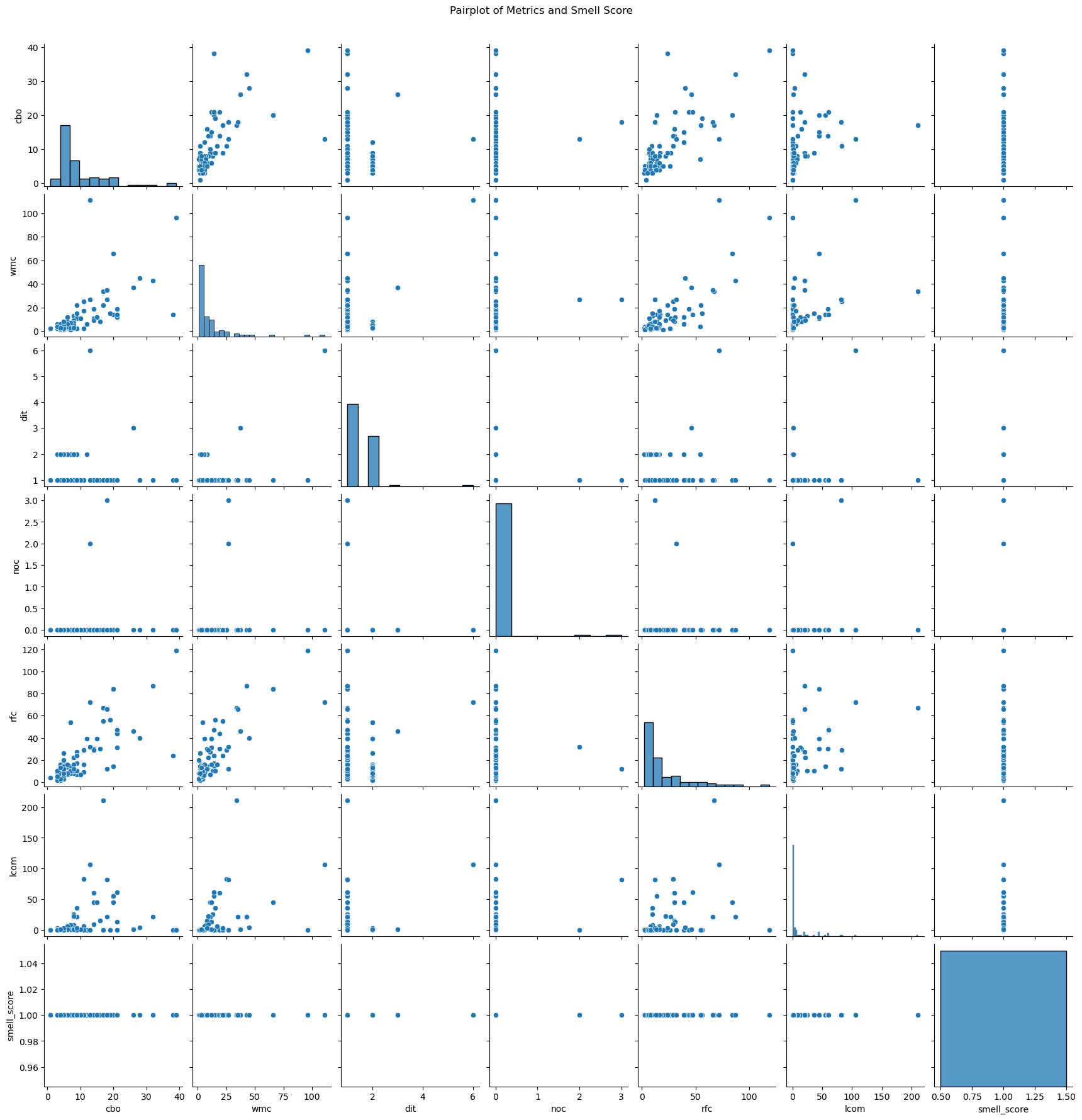
Relationship between CBO and Smell Score is given below



Correlation analysis is given below



Pair plot of Metrics and Smell Score is shown below

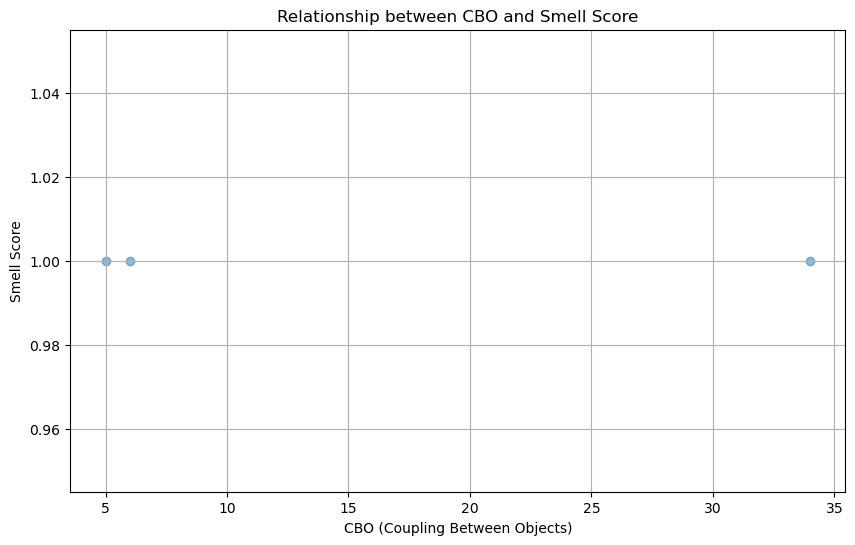


Classes with high cbo values typically indicate high coupling between different parts of the codebase, potentially leading to increased complexity and decreased modularity. For instance, class com.alibaba.easyexcel.test.demo.read.readtest stands out with a cbo of 20 and wmc of 14, suggesting it may benefit from refactoring to reduce its complexity and improve maintainability. Similarly, com.alibaba.excel.context.writecontextimpl exhibits exceptionally high values in wmc (96) and rfc (119), indicating it holds a large number of methods and responsibilities. Addressing these metrics could enhance code clarity and reduce the risk of errors or bugs.

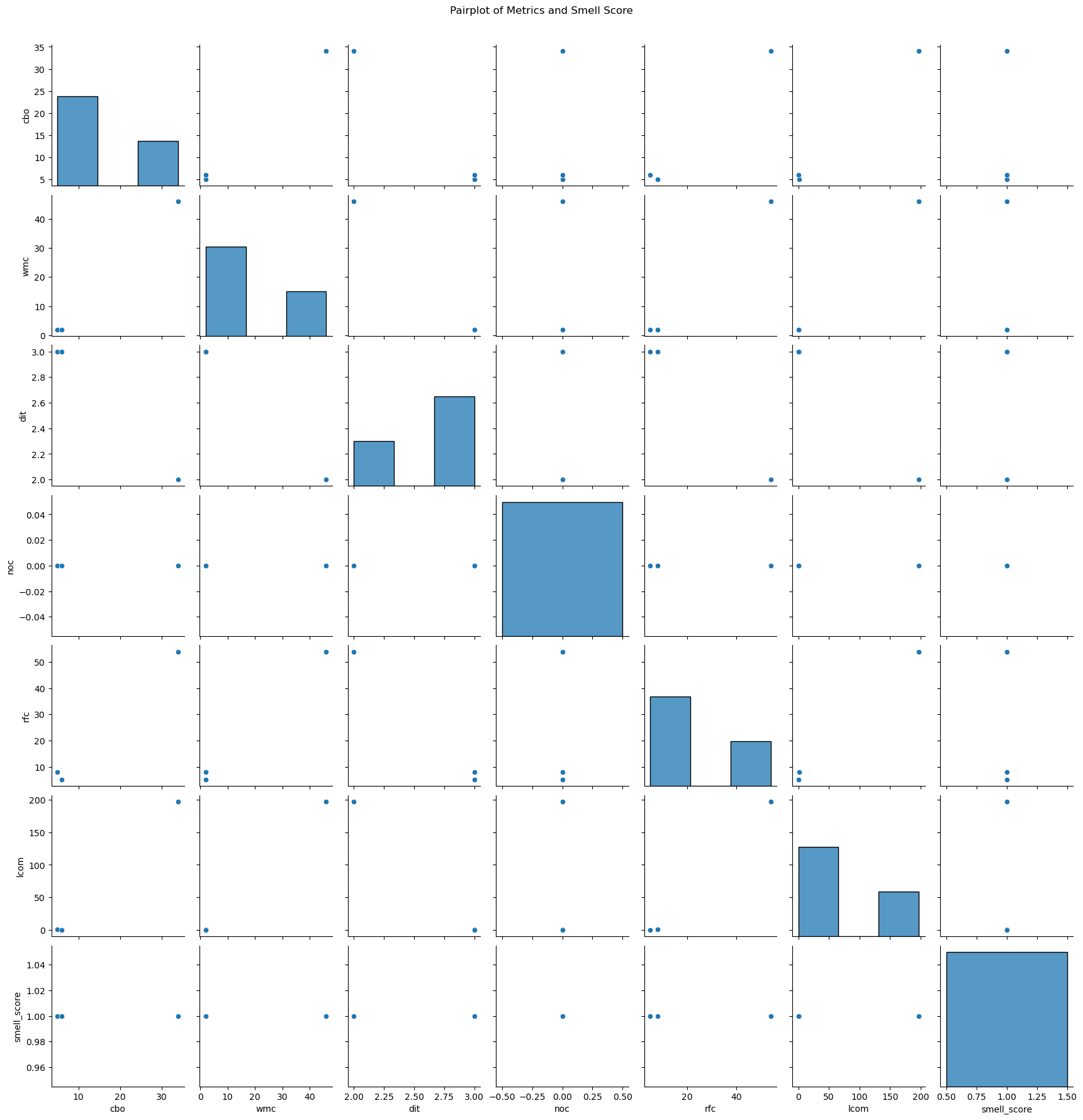
The distribution of these metrics across various classes underscores the importance of targeted refactoring efforts. Classes with elevated smell\_score indicate they are associated with top rules from pmd\_df, such as GuardLogStatement, UncommentedEmptyMethodBody, and CloseResource. These rules highlight potential code smells like incomplete error handling, excessive complexity, and resource management issues. By prioritizing classes based on their combination of high complexity metrics and smell\_score, developers can effectively allocate resources to refactor critical sections of the codebase, thereby improving overall code quality and maintainability.

**Fresco**

Relationship between CBO and Smell Score is given below



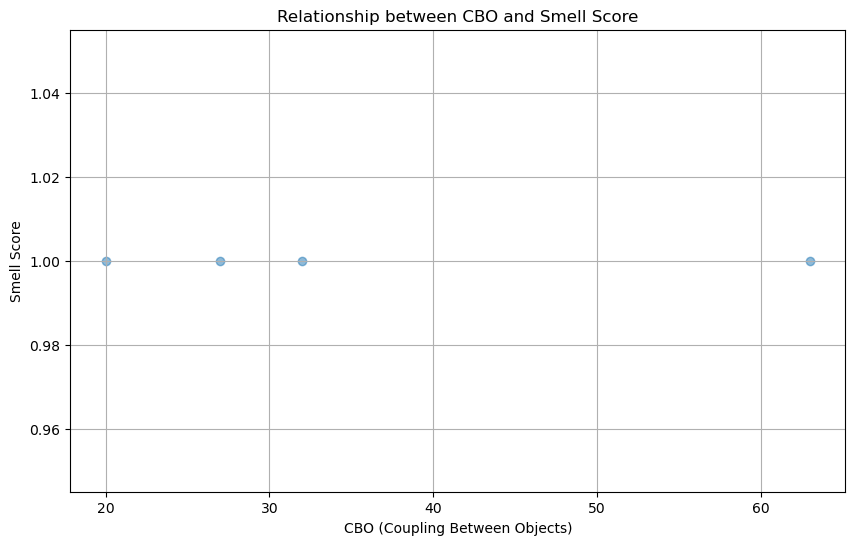
Pair plot of Metrics and Smell Score is shown below



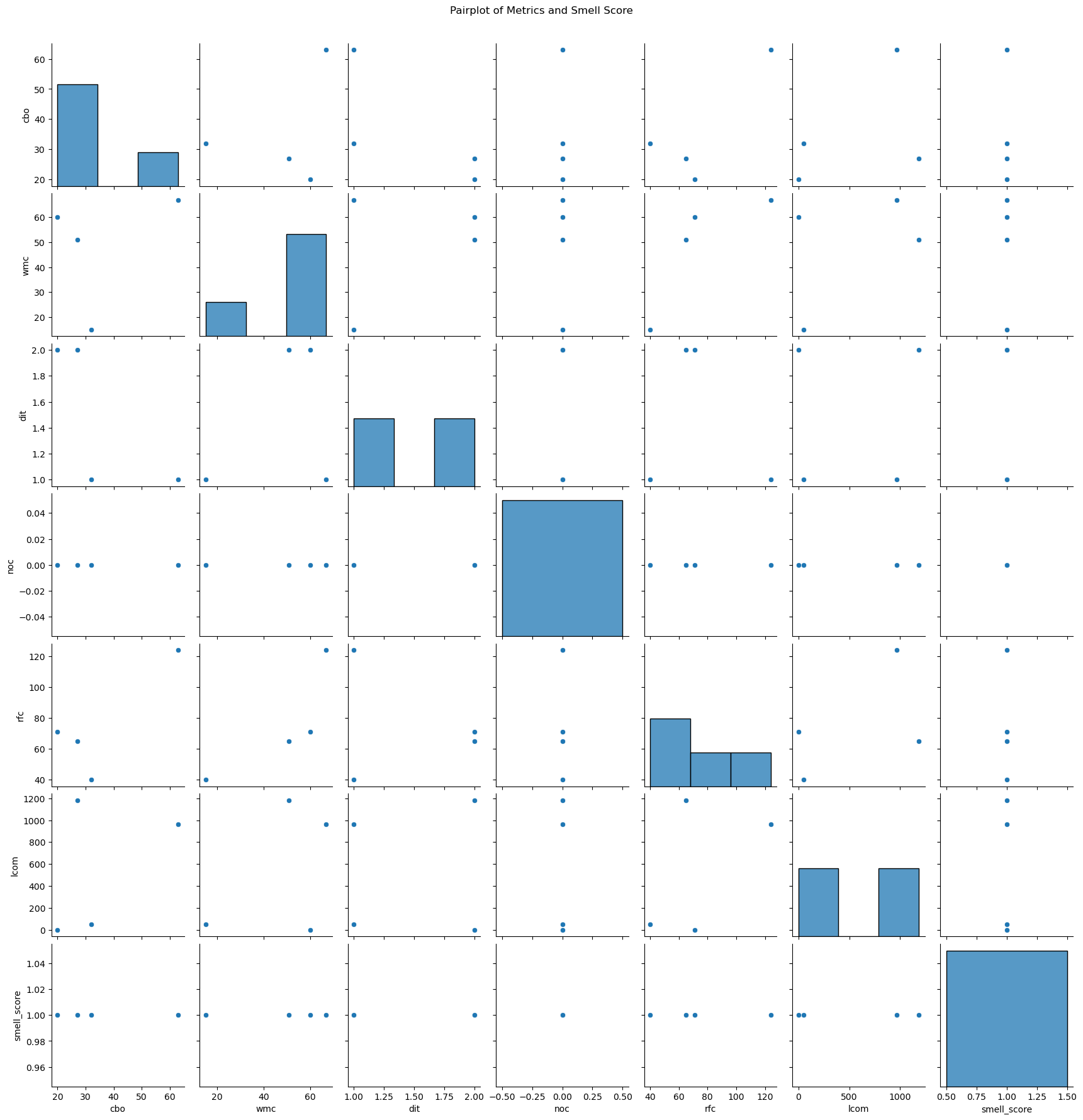
Classes such as GlideHolder and PicassoHolder from the comparison package in fresco-main demonstrate moderate to high values in metrics like Coupling Between Objects (CBO), Weighted Methods per Class (WMC), and Response for Class (RFC). For instance, FrescoFlipperPlugin in flipper-fresco exhibits a significant CBO of 34, indicating a complex interrelationship among objects within the class. This complexity can potentially lead to harder-to-maintain codebases and increased dependencies. The presence of high LCOM (Lack of Cohesion of Methods) scores, particularly noticeable in classes like FrescoFlipperPlugin, suggests that these classes might have methods that are not adequately cohesive. This lack of cohesion can hinder code comprehension and maintenance, making it challenging for developers to modify or extend the functionality without unintended consequences. The classes identified are concentrated in specific packages (comparison and flipper-fresco) within the larger fresco-main repository. This concentration could imply certain areas of the codebase that might benefit from refactoring efforts to improve readability, reduce complexity, and enhance maintainability. Addressing these issues could potentially lead to more robust software with reduced risk of introducing defects during future development cycles.

**Glide**

Relationship between CBO and Smell Score is given below



Pair plots are shown below



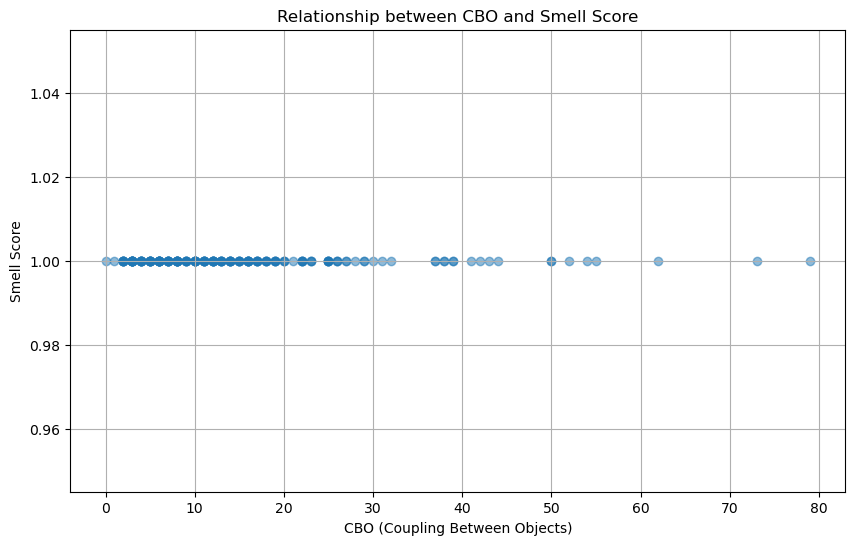
Classes such as RequestBuilderTest, DarkModeTest, GlideTest, and RequestOptionsTest from the Glide library's test suite exhibit substantial complexity metrics. For instance, GlideTest stands out with a high CBO (Coupling Between Objects) of 63 and an extensive RFC (Response for Class) of 124. These metrics suggest intricate interdependencies and potentially complex logic within these test classes, which are critical for ensuring the robustness and functionality of the Glide library across different scenarios.

The high values of WMC (Weighted Methods per Class), coupled with elevated LCOM (Lack of Cohesion of Methods) scores in classes like DarkModeTest and GlideTest, indicate potential areas of concern regarding code maintainability. High LCOM values, such as 1183 in DarkModeTest and 964 in GlideTest, suggest that these classes might have methods that are less cohesive, making them harder to understand and maintain over time. Addressing these cohesion issues through refactoring could improve code comprehensibility and reduce the risk of introducing unintended side effects during future modifications.

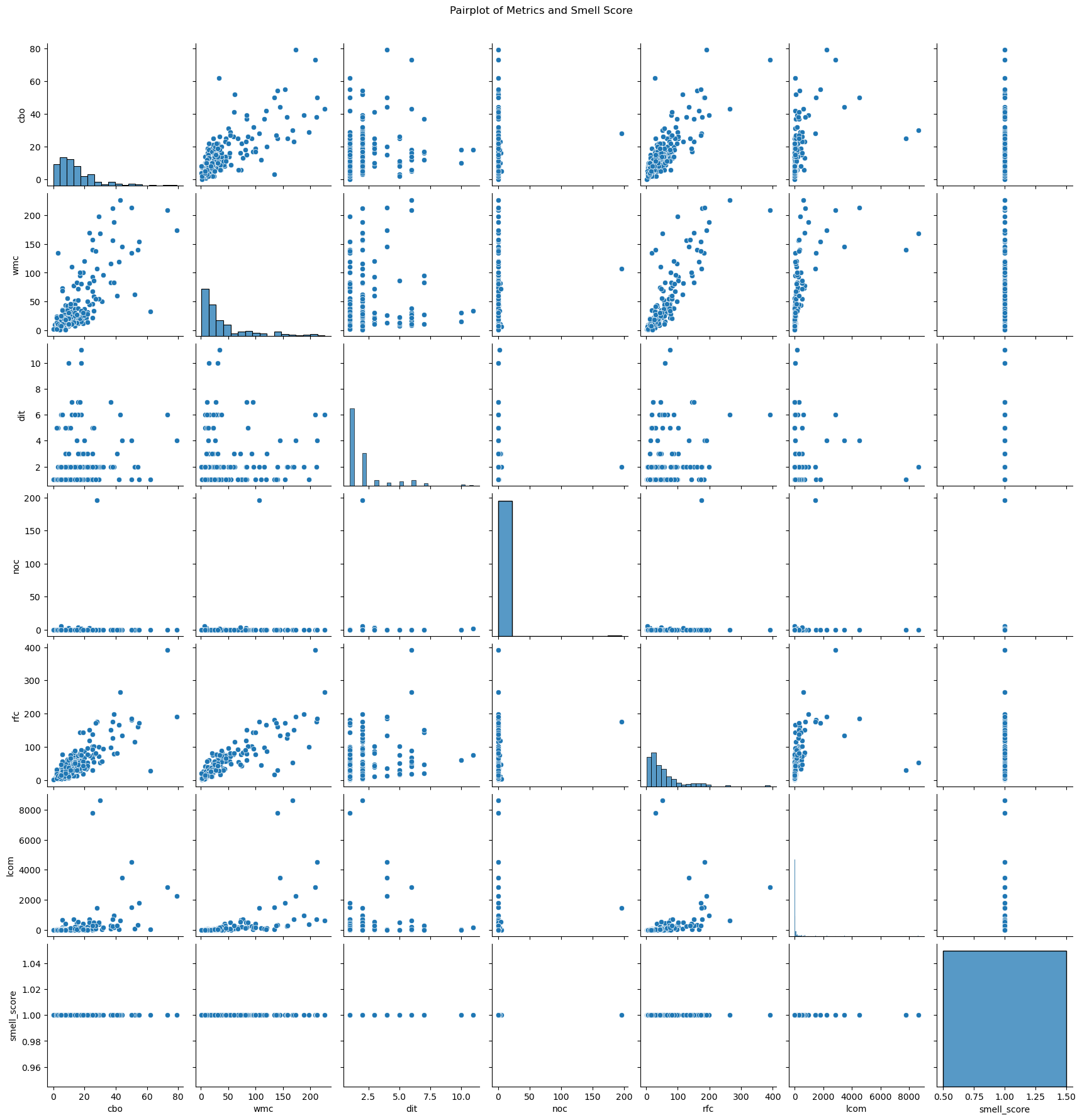
Given the concentration of these complex and cohesive-related metrics within the Glide test suite, developers could consider targeted refactoring efforts to streamline code logic, reduce interdependencies, and enhance maintainability. Such improvements not only ensure that tests remain effective in validating the library's behavior but also contribute to overall code quality and developer productivity. By addressing these insights, the Glide library can better maintain its high standards of reliability and performance across different testing scenarios and environments.

**Jadx**

Relationship between CBO and Smell Score is given below



Pair plot is given below



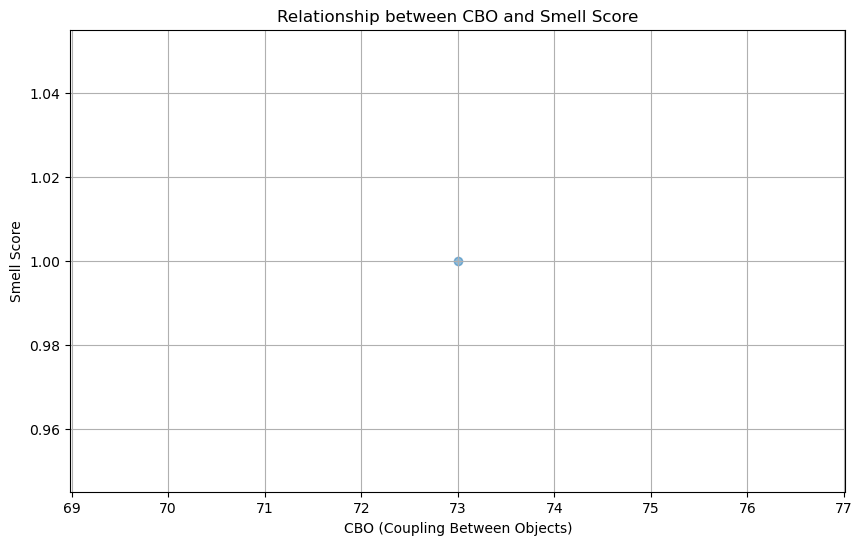
Classes like UsageInfoVisitor, GenericTypesVisitor, and CodeRenameVisitor from the jadx.core package exhibit notable complexity metrics such as high CBO (Coupling Between Objects) and RFC (Response for Class). For example, UsageInfoVisitor has a CBO of 22 and an RFC of 53, indicating significant interdependencies and method invocations within these classes. These metrics highlight the intricate nature of the core functionalities in jadx, which involve analyzing and transforming code representations at the bytecode level. Such classes play a crucial role in the core functionalities of jadx, contributing to its capability to decompile and analyze Java bytecode effectively.

dditionally, classes like CodeStringCache, QuarkReportNode, ApkSignature, CodeArea, and RelativePathTypeAdapter from the jadx.gui and related packages represent user interface components and utility functionalities. These classes show varying levels of complexity, with metrics indicating their role in managing GUI elements, caching data, or providing utility functions. For instance, CodeArea stands out with a high RFC of 81 and LCOM (Lack of Cohesion of Methods) of 273, suggesting it manages a complex set of functionalities related to displaying and manipulating code in the GUI.

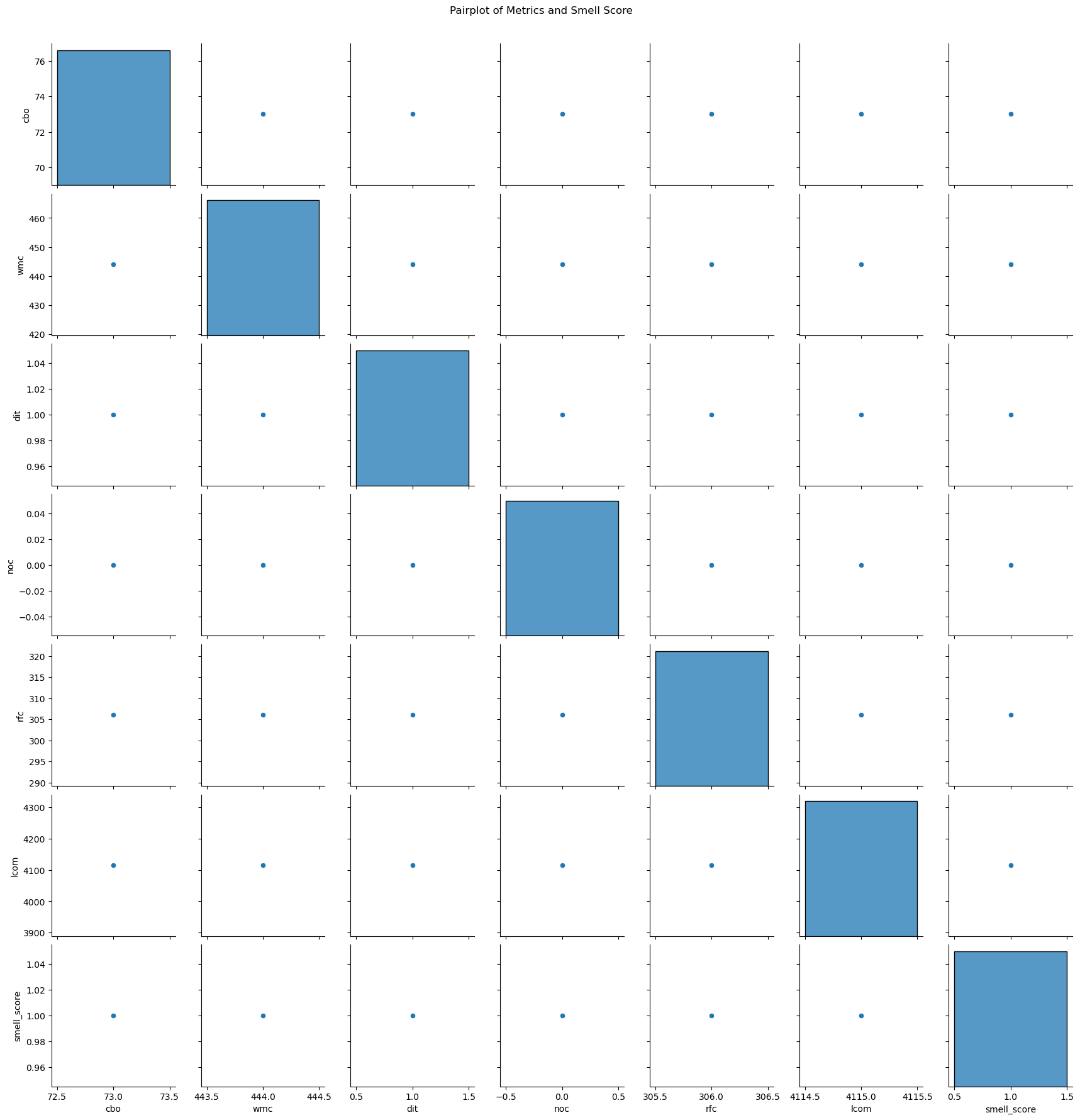
The insights derived from these metrics not only underscore the complexity and functionality breadth of the jadx project but also highlight potential areas for improvement. High values in metrics like RFC and CBO may indicate areas where refactoring or optimization could enhance code maintainability and performance. For instance, reducing method interdependencies or improving method cohesion in classes with high LCOM scores could streamline code execution paths and facilitate easier maintenance in the long term.

**NewPipe**

Relationship between CBO and Smell Score is given below



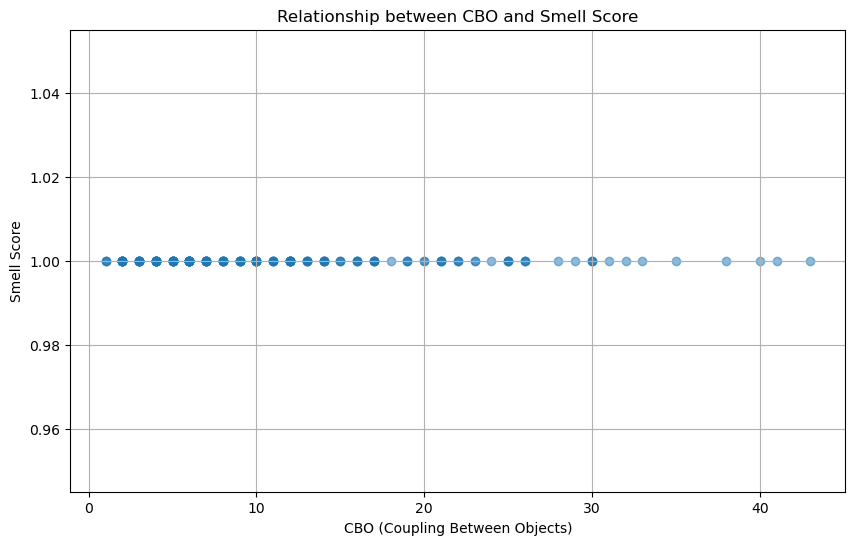
Pair plot is shown below



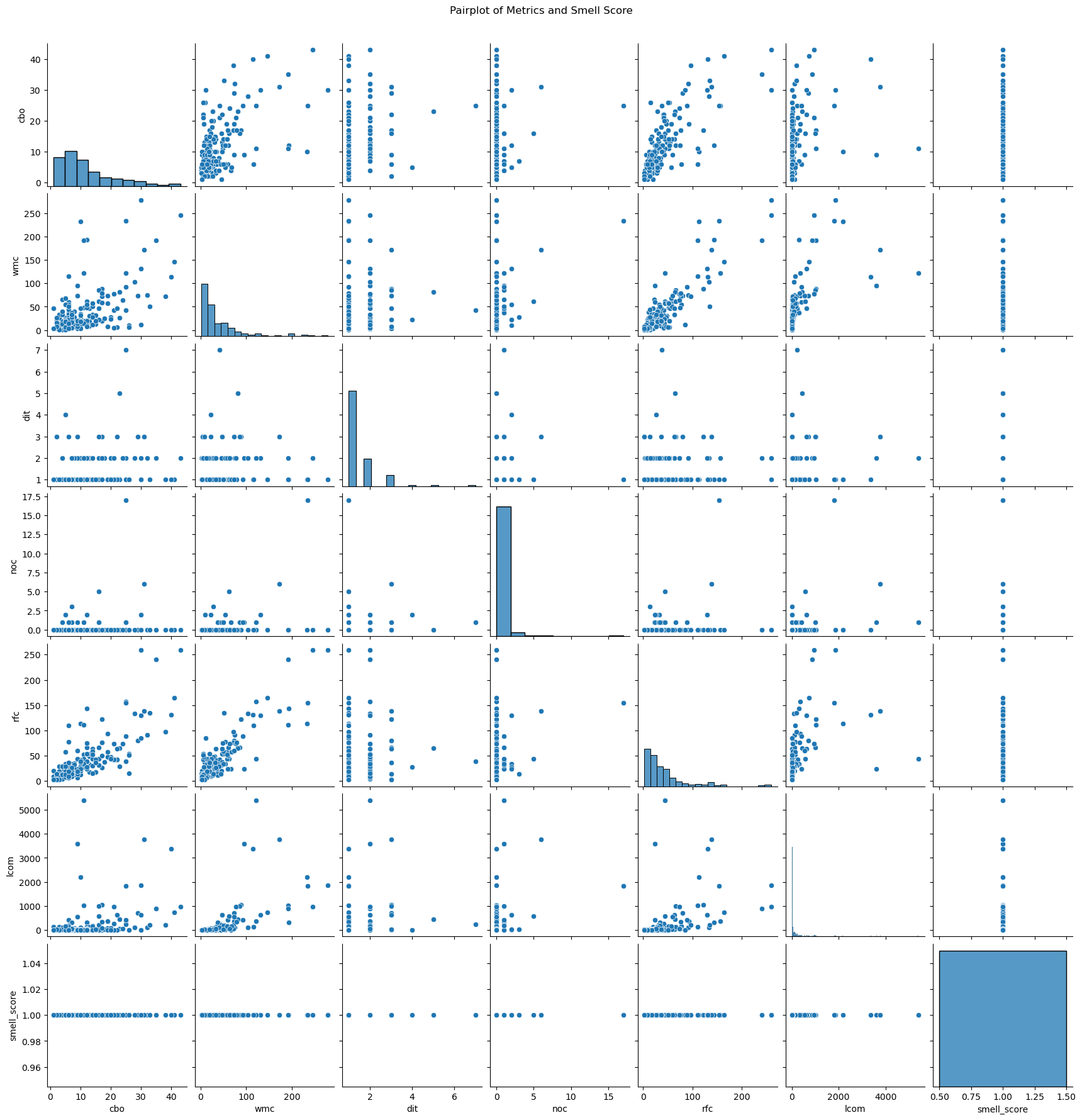
With a value of 444, the Player class contains a substantial number of methods. This suggests that it likely encapsulates a wide range of functionalities related to media playback or related operations within the NewPipe application. The RFC value of 306 indicates the number of unique methods that can potentially be executed in response to a message received by an object of this class. This metric reflects the class's depth and complexity in terms of method interactions and potential code execution paths. A remarkably high LCOM value of 4115 highlights potential concerns regarding the class's cohesion. A high LCOM often suggests that the methods within the class may not be strongly related or that there could be opportunities to refactor the class for better modularity and maintainability.

**Dubbo**

Relationship between CBO and Smell Score is given below



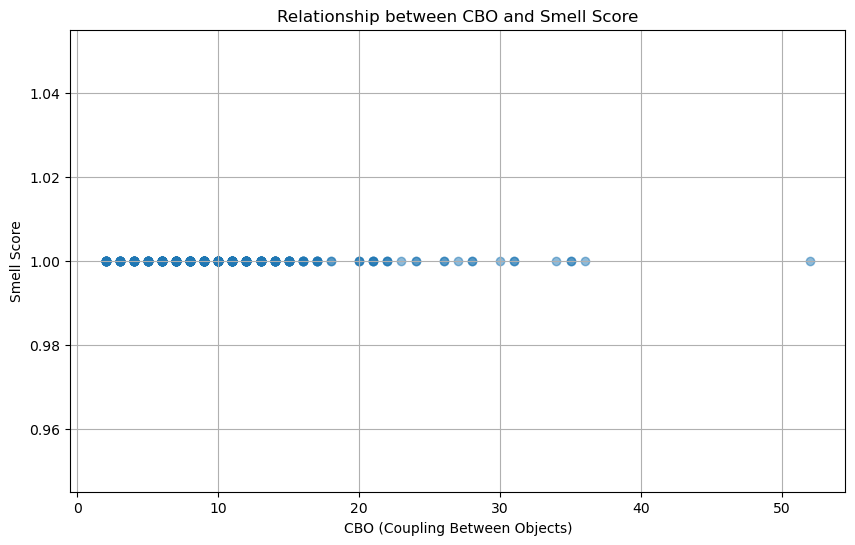
Pair plots are given below



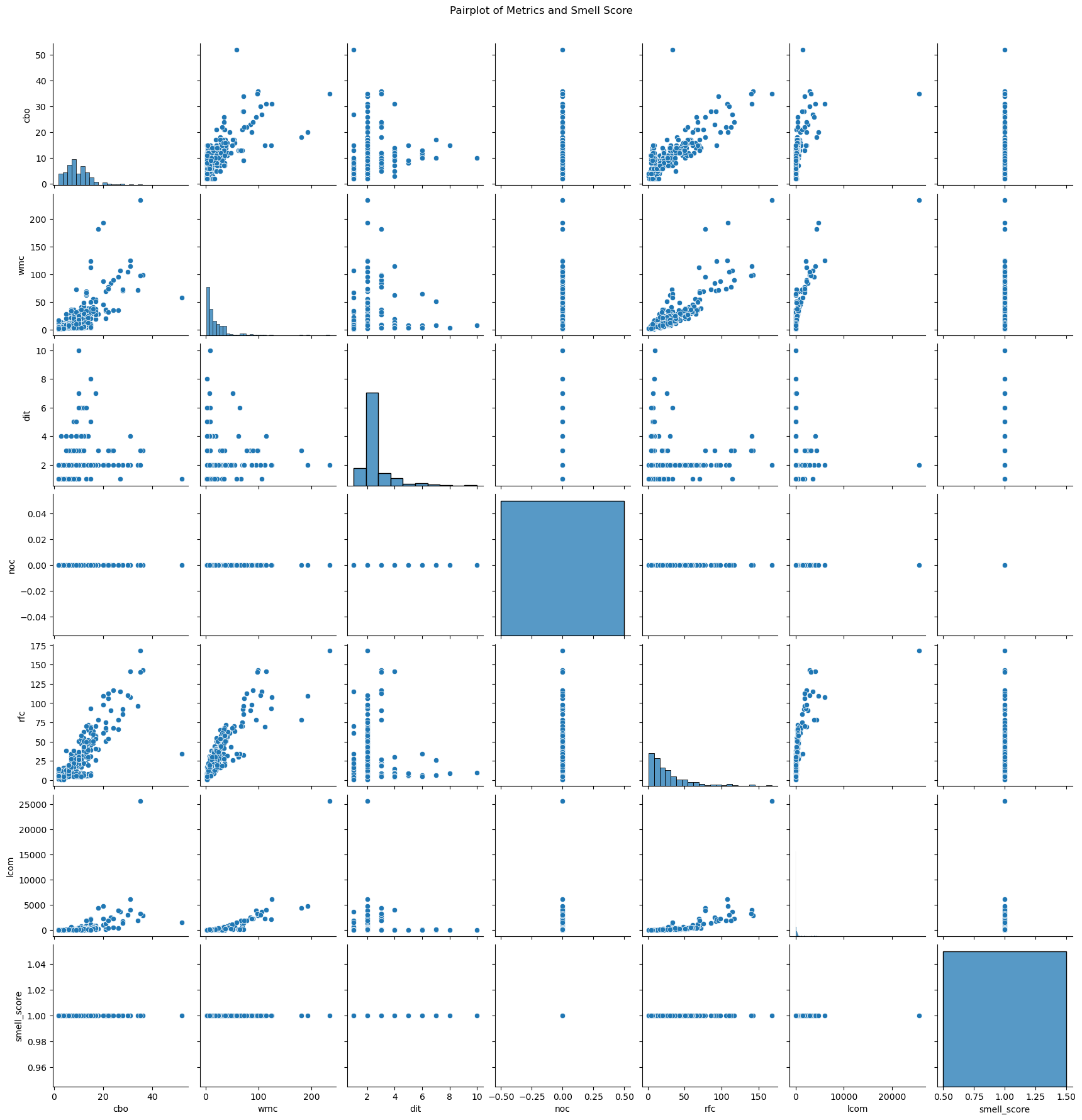
The values vary significantly across the classes, highlighting their different sizes and functionalities within the Dubbo ecosystem. For instance, DefaultApplicationContext stands out with a WMC of 246, indicating a complex implementation possibly managing various aspects of application context configuration. Classes like ReferenceConfig and AbstractClusterInvoker show higher RFC values (122 and 157 respectively), indicating extensive method interactions and potential code execution paths within these components. Some classes, such as AbstractClusterInvoker and DefaultApplicationContext, exhibit higher LCOM values (356 and 960 respectively), suggesting potential concerns about method cohesion and the need for refactoring to enhance modularity and maintainability. Classes like AbstractClusterInvoker, ReferenceConfig, and DefaultApplicationContext play pivotal roles in the Dubbo framework. Their high complexity metrics underscore their central positions in managing clustering strategies, service references, and application context configurations, respectively.

**RX JAVA**

Relationship between CBO and Smell Score is given below



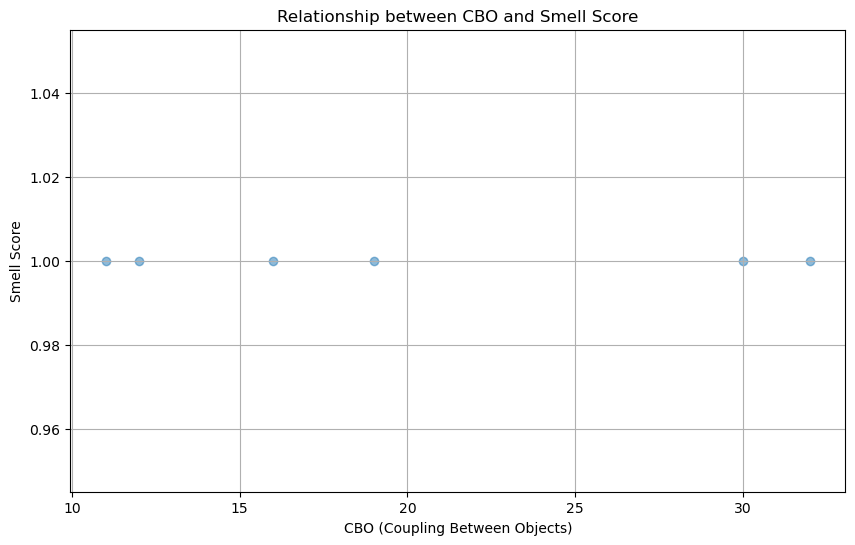
Pair plots are shown below



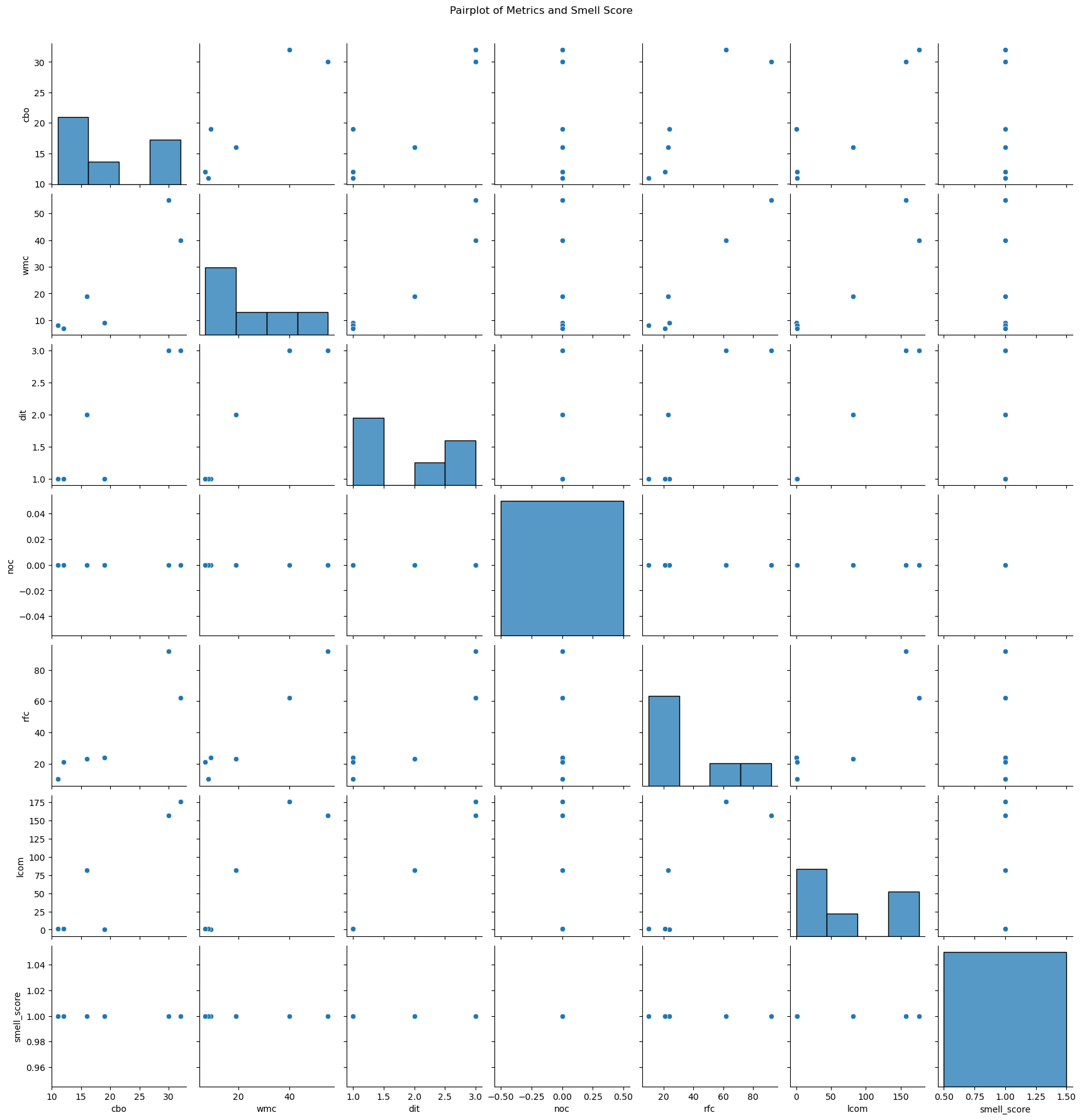
The RXJava classes exhibit a range of WMC values, reflecting their different functionalities and complexities. For instance, classes like FlowableConcatMapCompletable and FlowableConcatMapMaybe have higher WMC values (35 and 54 respectively), indicating substantial logic and operations within these reactive operators. Several classes, such as FlowableConcatMapCompletable and ObservableConcatMap, demonstrate higher RFC values (65 and 70 respectively). This suggests these classes are involved in numerous method interactions and are pivotal in managing reactive streams and transformations within the RXJava framework. lasses like CompletableConcat and SingleConcatMapIterable exhibit higher LCOM values (398 and 820 respectively). These values imply potential concerns regarding method cohesion within these classes, possibly indicating opportunities for refactoring to enhance code clarity and maintainability.

**Signal Android**

Relationship between CBO and Smell Score is given below



Pair plots are shown below



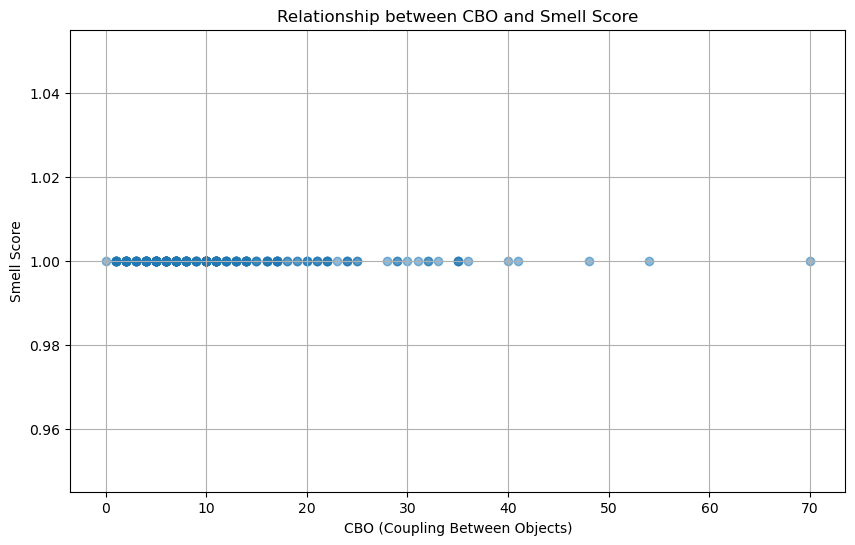
Classes such as CreateGroupActivity, CallFragment, and AvatarImageView demonstrate varied complexity metrics. For example, CallFragment exhibits a high Weighted Methods per Class (WMC) value of 55, indicating it involves a significant amount of logic and functionality related to web real-time communication (WebRTC). Similarly, AvatarImageView has a notable Response for Class (RFC) of 62, suggesting it manages numerous method interactions, possibly related to avatar image rendering and manipulation within the application.

The presence of classes like CreateGroupActivity and ConfirmPaymentFragment with moderate to high Coupling Between Objects (CBO) values (16 and 19 respectively) suggests these classes may have dependencies that could potentially affect their maintainability and extensibility. Careful management of these dependencies and adhering to design principles such as SOLID could enhance code quality and flexibility.

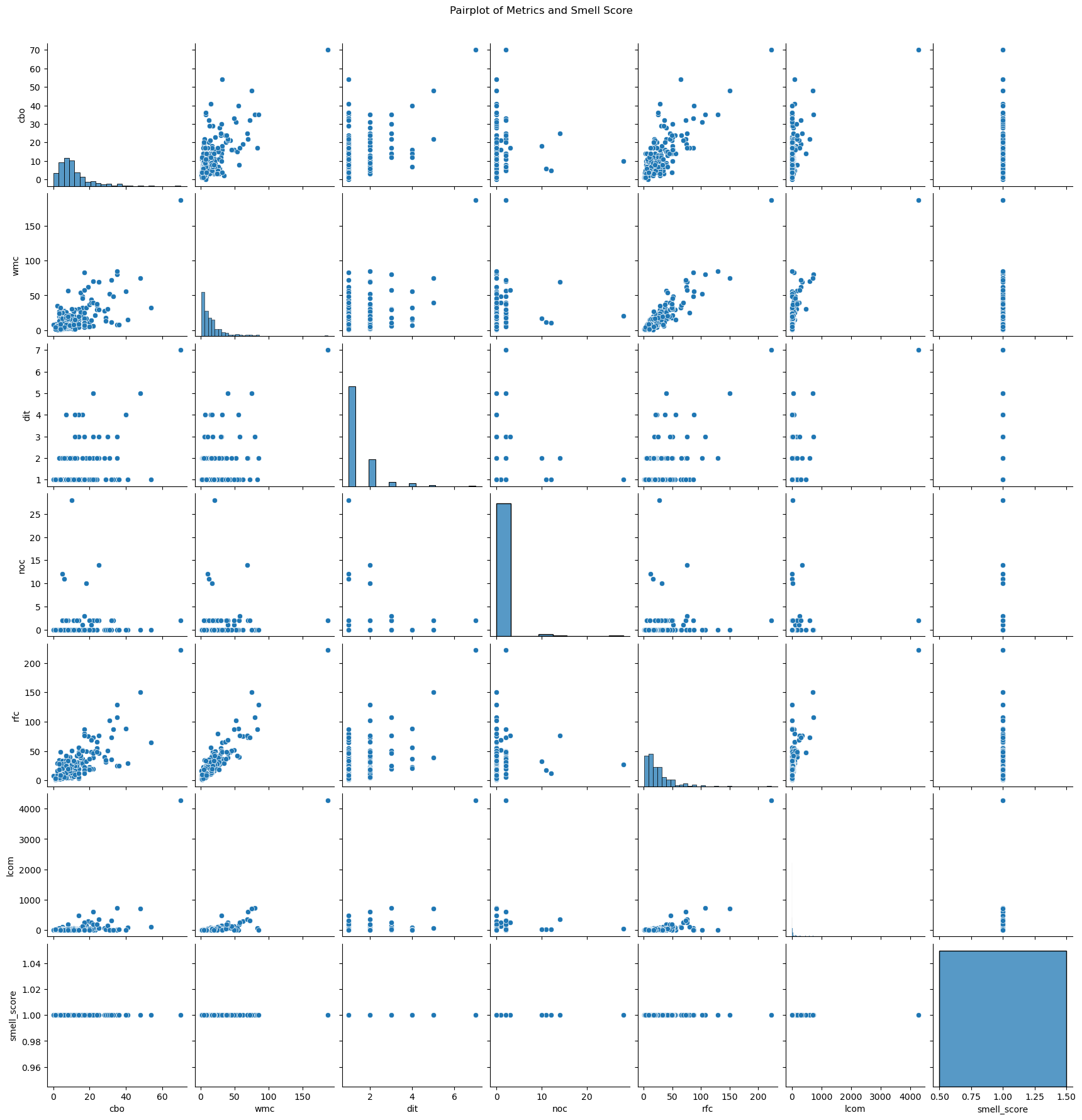
Classes such as GiphyMp4Resource and CreateGroupActivity exhibit low Lack of Cohesion in Methods (LCOM) values, indicating good method cohesion within these classes. This characteristic is beneficial for code readability and comprehensibility, contributing to easier maintenance and debugging processes.

**Springboot**

Relationship between CBO and Smell Score is given below



Pari plots are given below

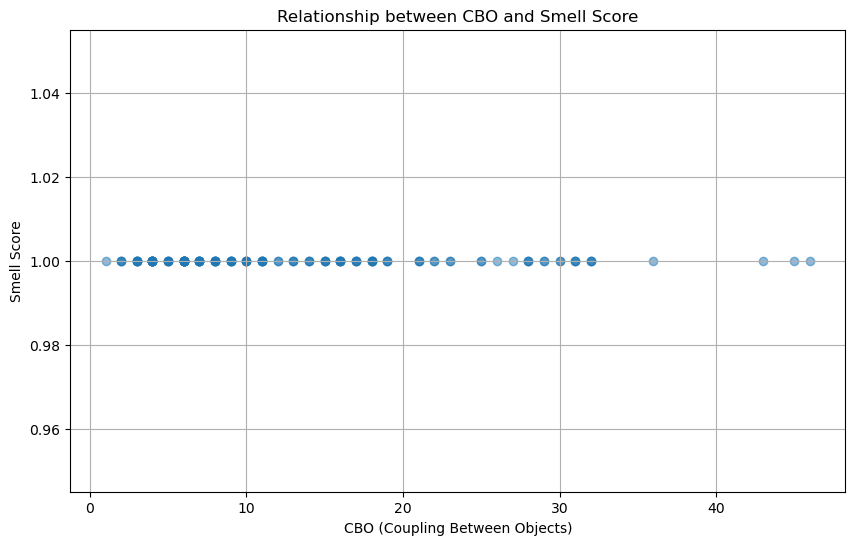


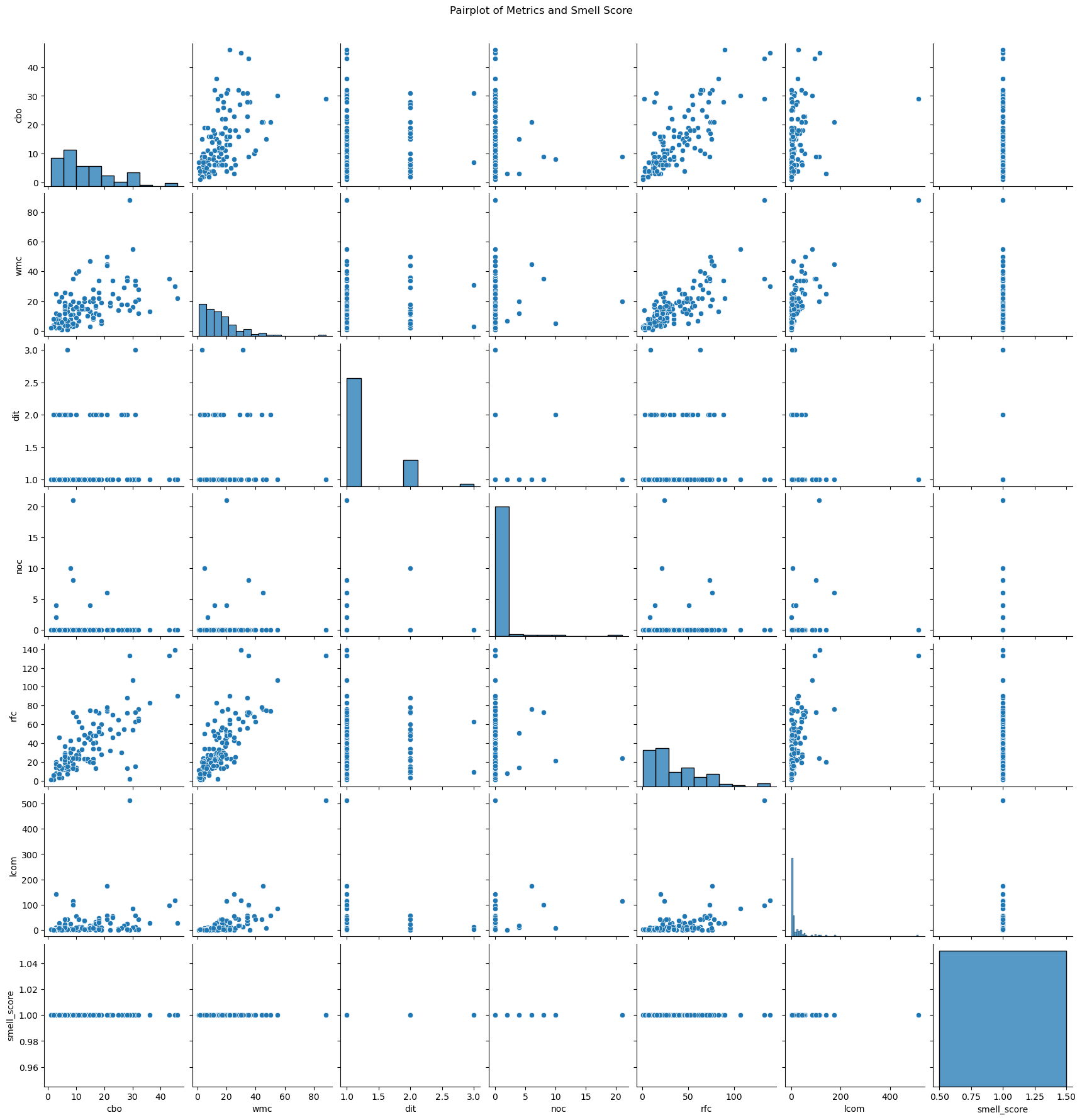
Classes such as SampleWebSocketsApplication, BuildInfoMojo, and AbstractAotMojo exhibit different complexity metrics. For instance, SampleWebSocketsApplication has a relatively low Weighted Methods per Class (WMC) of 6, indicating it might be a straightforward class handling WebSocket functionality in a testing context. In contrast, BuildInfoMojo shows a higher WMC of 16, suggesting it manages more complex build information tasks within the Maven framework. The classes vary in their Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM) metrics.

AbstractAotMojo, for example, has a CBO of 14, indicating it might have dependencies on many other classes or components. Conversely, DefaultSslBundleResource exhibits an LCOM of 0, suggesting highly cohesive methods, potentially related to managing SSL certificates and bundles effectively. Classes like AutoConfigurationMetadata and HikariCheckpointRegistry, with higher Response for Class (RFC) values, suggest they might manage substantial interaction complexity or orchestrate various components within the Spring Boot framework. This complexity can influence how modular and maintainable these components are. Optimizing classes with higher complexity metrics (e.g., WMC, RFC) could enhance system performance. For instance, refactoring HikariCheckpointRegistry with an RFC of 33 could potentially streamline database checkpoint management, leading to improved database performance in Spring applications.

**SS**

Relationship between CBO and Smell Score is given below





Classes like org.apache.shardingsphere.data.pipeline.core.engine.InputGroup, org.apache.shardingsphere.data.pipeline.core.converter.StandardConverter, and org.apache.shardingsphere.data.pipeline.core.context.QueryDataContext are integral to managing data processing within the SS project's data pipeline core. These classes typically exhibit moderate to high Weighted Methods per Class (WMC) and Response for Class (RFC) metrics, indicating they handle significant logic and orchestrate complex interactions.

Classes such as org.apache.shardingsphere.test.e2e.transaction.TransactionCommitIT, involved in end-to-end testing of transactional behavior, show lower complexity metrics like WMC and RFC, suggesting simpler functionality focused on specific testing scenarios.

The Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM) metrics vary across these classes. For instance, InputGroup has a CBO of 3, indicating fewer dependencies, while StandardConverter shows an LCOM of 0, indicating highly cohesive methods. These metrics reflect design considerations such as modularity and code maintainability. Classes with higher RFC values, such as QueryDataContext with an RFC of 72, likely manage more interactions and complexity within their respective functionalities. This complexity can influence scalability, performance, and the ease of future modifications.

**Conclusion**

The analysis of various classes across multiple Java repositories has highlighted critical areas that would benefit from targeted refactoring to improve code maintainability, readability, and overall quality. High Coupling Between Objects (CBO) values, as seen in classes like com.alibaba.easyexcel.test.demo.read.readtest and FrescoFlipperPlugin, indicate significant interdependencies, potentially leading to increased complexity and decreased modularity. These classes should be refactored to decouple their components, simplifying the codebase and enhancing its modularity. Similarly, high Weighted Methods per Class (WMC) and Response for Class (RFC) values in classes such as com.alibaba.excel.context.writecontextimpl and DarkModeTest point to excessive complexity and responsibilities. Refactoring these classes to distribute functionality more evenly can reduce the risk of errors and improve code clarity.

The concentration of complex metrics and code smells in specific packages, such as the comparison and flipper-fresco packages within the fresco-main repository, underscores the need for localized refactoring efforts. Classes like GlideHolder and PicassoHolder from the Glide library’s test suite exhibit high metrics, suggesting intricate logic and dependencies that could be streamlined for better maintainability. The presence of high Lack of Cohesion of Methods (LCOM) scores in classes like FrescoFlipperPlugin and CodeArea further emphasizes the need for improving method cohesion. Addressing these issues through refactoring can enhance code comprehension and reduce the risk of unintended side effects during modifications.

By prioritizing classes with a combination of high complexity metrics and code smells, developers can effectively allocate resources to refactor critical sections of the codebase. This approach will improve the robustness and maintainability of the software, reducing the likelihood of defects and easing future development efforts. Such targeted refactoring not only enhances code quality but also ensures that the software remains scalable and efficient over time, contributing to the overall stability and performance of the projects analyzed.

**References**

Badri, M., Badri, L. and Flageol, W. (2016) ‘Source and test code size prediction - A comparison between use case metrics and objective class points’, *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering* [Preprint]. doi:10.5220/0005857601720180.

Sharma, T. and Kessentini, M. (2021) ‘QScored: A large dataset of code smells and Quality Metrics’, *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)* [Preprint]. doi:10.1109/msr52588.2021.00080.