PENGARUH PENYEIMBANGAN DATA BERBASIS KARAKTERISTIK FITUR TERHADAP KINERJA *RANDOM FOREST* DALAM PREDIKSI *DEFECT* PERANGKAT LUNAK

SKRIPSI

Oleh :
RIZQI AMALIA KARTIKA
NIM. 210605110064

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2025

PENGARUH PENYEIMBANGAN DATA BERBASIS KARAKTERISTIK FITUR TERHADAP KINERJA *RANDOM FOREST* DALAM PREDIKSI *DEFECT* PERANGKAT LUNAK

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
RIZQI AMALIA KARTIKA
NIM. 210605110064

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2025

HALAMAN PERSETUJUAN

PENGARUH PENYEIMBANGAN DATA BERBASIS KARAKTERISTIK FITUR TERHADAP KINERJA *RANDOM FOREST* DALAM PREDIKSI *DEFECT* PERANGKAT LUNAK

SKRIPSI

Oleh :
RIZQI AMALIA KARTIKA
NIM. 210605110064

Telah Diperiksa dan Disetujui untuk Diuji
Tanggal 19 Mei 2025

Pembimbing I,	Pembimbing II,
Fatchurrochman, M.Kom	Nur Fitriyah Ayu Tunjung Sari, M.Cs
NIP. 19700731 200501 1 002	NIP. 19911226 202012 2 001

Mengetahui,
Ketua Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Ir. Fachrul Kurniawan, M.MT., IPU
NIP. 19771020 200912 1 001

HALAMAN PENGESAHAN

PENGARUH PENYEIMBANGAN DATA BERBASIS KARAKTERISTIK FITUR TERHADAP KINERJA *RANDOM FOREST* DALAM PREDIKSI *DEFECT* PERANGKAT LUNAK

SKRIPSI

Oleh :
RIZQI AMALIA KARTIKA
NIM. 210605110064

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan
Dinyatakan Diterima Sebagai Salah Satu Persyaratan untuk
Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal : 19 Mei 2025

Susunan Dewan Penguji

Ketua Penguji : Dr. Ririen Kusumawati, S.Si, M.Kom ( )
NIP. 19720309 200501 2 002

Anggota Penguji I : Dr. Zainal Abidin, M.Kom ( )
NIP. 19760613 200501 1 004

Anggota Penguji II : Fatchurrohman, M.Kom ( )
NIP. 19700731 200501 1 002

Anggota Penguji III : Nur Fitriyah Ayu Tunjung Sari, M.Cs ( )
NIP. 19911226 202012 2 001

Mengetahui,dan Mengesahkan,
Ketua Program Studi Teknik Informatiksa
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Ir. Fachrul Kurniawan, M.MT., IPU
NIP. 19771020 200912 1 001

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	:	Rizqi Amalia Kartika
NIM	:	210605110064
Fakultas/Program Studi	:	Sains dan Teknologi / Teknik Informatika
Judul Skripsi	:	Pengaruh Penyeimbangan Data Berbasis Karakteristik Fitur Terhadap Kinerja *Random Forest* Dalam Prediksi *Defect* Perangkat Lunak

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran orang lain yag saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencatumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini merupakan hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 17 Juni 2025

Yang membuat pernyataan,

Rizqi Amalia Kartika

NIM.210605110064

MOTTO

Good Things Take Time,
Trust your own journey.

HALAMAN PERSEMBAHAN

Puji syukur atas kehadirat Allah Subhanahu wa Ta'ala, karena berkat rahmat dan petunjuk-Nya penulis dapat menyelesaikan skripsi ini. Shalawat serta salam kepada Nabi Muhammad Shallallahu 'alaihi wasallam, yang telah membawa umat manusia dari zaman kegelapan menuju cahaya Islam. Dengan penuh rasa syukur, karya ini penulis persembahkan untuk mama tercinta, Ani Wijayati, dan ayah tercinta, Moh. Irham, yang dengan kasih sayang, doa, dan pengorbanannya senantiasa menjadi sumber kekuatan dalam setiap langkah hidup penulis. Ucapan terima kasih juga penulis sampaikan kepada saudara dan seluruh keluarga tersayang atas dukungan dan cinta yang tidak pernah putus. Persembahan ini juga ditujukan untuk diri sendiri, sebagai bentuk penghargaan atas keteguhan hati, usaha, dan keberanian dalam menghadapi berbagai tantangan selama proses penyusunan skripsi ini. Tak lupa, penulis mempersembahkan karya ini untuk teman-teman seperjuangan yang telah menjadi bagian dari perjalanan ini dalam setiap kebersamaan, semangat, dan saling menguatkan yang tak ternilai. Semoga skripsi ini menjadi langkah awal menuju kebaikan dan kebermanfaatan.

KATA PENGANTAR

Puji syukur penulis panjatkan kepada kehadirat Allah Swt yang telah melimpahkan nikmat serta karunia-Nya sehingga penulis mampu menyelesaikan penulisan Skripsi yang berjudul **Pengaruh Penyeimbangan Data Berbasis Karakteristik Fitur Terhadap Kinerja *Random Forest* Dalam Prediksi *Defect* Perangkat Lunak** dengan baik dan tepat waktu.

Dalam penulisan skripsi ini, penulis menyadari banyak pihak yang terlibat baik dalam proses membimbing penulisan dan juga memberikan semangat dan dukungan. Untuk itu, penulis ingin menyampaikan terima kasih kepada:

1. Prof. Dr. H.M. Zainuddin, MA., selaku rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang.
2. Prof. Dr. Sri Hariani, M.Si., selaku dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
3. Dr. Ir. Fachrul Kurniawan, M.MT., IPU, selaku Ketua Program Studi Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Fatchurrochman, M.Kom selaku Dosen Pembimbing I, yang telah memberikan banyak dukungan dan bimbingan dalam penulisan skripsi ini.
5. Nur Fitriyah Ayu Tunjung Sari, M.Cs. selaku Dosen Pembimbing II, yang telah memberikan banyak dukungan dan bimbingan dalam penulisan skripsi ini.

6. Dr. Ririen Kusumawati, S.Si, M.Kom selaku Ketua Penguji, atas segala saran dan masukan yang sangat membangun, sejak tahap seminar proposal hingga sidang skripsi.
7. Dr. Zainal Abidin, M.Kom selaku Anggota penguji I, atas semua saran dan masukan yang memperkuat kualitas skripsi ini, sejak tahap seminar proposal hingga sidang skripsi.
8. Nia Faricha S, Si., selaku Admin Program Studi Teknik Informatika yang selalu sabar memberikan informasi, membantu, dan memberikan arahan selama perkuliahan dan proses penulisan skripsi ini.
9. Okta Qomaruddin Aziz, M.Kom., selaku dosen yang telah berkenan memberikan arahan, masukan, serta solusi yang sangat membantu dalam penyusunan dan penyempurnaan penelitian ini. Seluruh Dosen dan Jajaran Staf Program Studi Teknik Informatika atas segala bentuk dukungan, bantuan, dan ilmu yang telah diberikan selama proses perkuliahan hingga penyusunan skripsi ini.
10. Kedua orang tua tercinta, Mama Ani dan Ayah Irham, yang selalu menjadi sumber doa, semangat, dan kasih sayang yang tak pernah padam. Segala pengorbanan, dukungan, dan ketulusan cinta yang telah diberikan menjadi kekuatan utama bagi penulis dalam menyelesaikan skripsi ini. Penulis juga menyampaikan rasa terima kasih yang mendalam kepada Kakak Kiki, Kakak Ade, serta Maher, yang senantiasa memberikan semangat, perhatian, dan motivasi di tengah berbagai tantangan. Kehadiran kalian menjadi

penyemangat tersendiri yang menguatkan penulis untuk terus melangkah hingga tahap akhir dalam proses ini.

11. Sahabat-sahabat terbaik sejak awal perkuliahan, Dinindriya Izzatinisa, Intan Tiara Dewi, dan Hamidah Lutfiyanti Maharani. Terima kasih atas kebersamaan, dukungan, dan segala tawa maupun tangis yang telah kita lalui bersama sejak semester pertama. Persahabatan kalian menjadi salah satu bagian terindah dalam perjalanan penulis selama menempuh studi ini.
12. Ahmad Arsha Albar, seorang yang begitu berarti dalam setiap proses dan langkah penulis. Dukungan, semangat, dan kebersamaan yang terjalin. Kehadiran dan pengertiannya telah banyak membantu penulis melalui berbagai fase selama menyelesaikan skripsi ini.
13. Teman-teman seperjuangan dalam masa penyusunan skripsi Suci, Adila, Nurul, Farros, Afiifah dan Nova. Terima kasih atas semangat, saling dukung, dan berbagi informasi maupun motivasi yang tak pernah putus. Perjalanan ini menjadi lebih ringan karena ditemani oleh teman-teman yang selalu menginspirasi dan menguatkan.
14. Seluruh teman-teman Angkatan 2021 Teknik Informatika yang telah banyak membantu, mendukung, dan memotivasi dalam menyelesaikan penyusunan skripsi ini.

Penulis menyadari bahwa skripsi ini masih terdapat berbagai kekurangan, baik dari segi isi maupun penulisan. Maka dari itu, penulis sangat mengharapkan kritik dan saran yang membangun demi perbaikan di masa mendatang. Semoga

dengan penyusunan skripsi ini dapat memberikan manfaat bagi semua pihak yang membacanya.

Malang, 17 Juni 2025

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Kartika, Rizqi Amalia. 2025. **Pengaruh Penyeimbangan Data Berbasis Karakteristik Fitur Terhadap Kinerja *Random Forest* Dalam Prediksi *Defect* Perangkat Lunak.** Skripsi. Program Studi Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Fatchurrochman, M.Kom, (II) Nur Fitriyah Ayu Tunjung Sari, M.Cs.

Kata Kunci: Prediksi *defect*, ketidakseimbangan kelas, *Random Forest*, SMOTE, *filtering* berbasis fitur.

Ketidakseimbangan kelas pada *dataset* prediksi *defect* perangkat lunak dapat menyebabkan model cenderung mengabaikan kelas minoritas. Penelitian ini mengevaluasi kinerja model *Random Forest* dalam tiga skenario: tanpa penyeimbangan, menggunakan SMOTE, dan SMOTE dengan *filtering* berbasis karakteristik fitur. Pengujian dilakukan pada 13 *dataset* dengan berbagai rasio pelatihan-pengujian (90:10 hingga 70:30), menggunakan metrik akurasi dan *F1-score*. Hasil menunjukkan bahwa kombinasi SMOTE dan *filtering* menghasilkan *F1-score* tertinggi sebesar 90,72% pada *dataset* PC2 dengan rasio 70:30, serta performa paling stabil pada rasio yang sama dengan rata-rata akurasi 78,32% dan standar deviasi 2,38%. Pendekatan ini secara signifikan meningkatkan kemampuan model dalam mendeteksi *defect* dibandingkan dua skenario lainnya. Dengan demikian, metode ini terbukti efektif dalam mengatasi ketidakseimbangan kelas tanpa menimbulkan noise berlebih, serta meningkatkan keandalan prediksi perangkat lunak.

ABSTRACT

Kartika, Rizqi Amalia. 2025. **Effect of Feature Characteristic Based Data Balancing on Random Forest Performance in Software Defect Prediction.** Thesis. Department of Informatics Engineering, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University Malang. Supervisors : (I) Fatchurrochman, M.Kom, (II) Nur Fitriyah Ayu Tunjung Sari, M.Cs.

Keywords: *Defect prediction, class imbalance*, *Random Forest*, *SMOTE, feature-based filtering.*

Class imbalance in software defect prediction datasets can cause models to overlook the minority class. This study evaluates the performance of a Random Forest model under three scenarios: without resampling, with SMOTE, and with SMOTE combined with feature-based filtering. Experiments were conducted on 13 datasets with varying training-testing ratios (from 90:10 to 70:30), using accuracy and F1-score as evaluation metrics. The results show that the combination of SMOTE and filtering achieved the highest F1-score of 90.72% on the PC2 dataset with a 70:30 split, and demonstrated the most stable performance at the same ratio, with a mean accuracy of 78.32% and a standard deviation of 2.38%**.** This approach significantly improved the model's ability to detect defects compared to the other two scenarios. Therefore, the proposed method is proven to be effective in addressing class imbalance without introducing excessive noise, while enhancing the reliability of software defect prediction.

مستخلص البحث

كارتيكا، رزقي أماليا. 2025. **تأثير موازنة البيانات القائمة على الخصائص المميزة على أداء الغابة العشوائية في التنبؤ بعيوب البرمجيات. البحث الجامعي**. قسم الهندسة المعلوماتية، كلية العلوم والتكنولوجيا بجامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: : (1)فاتشورشمان ،ماجستير في علوم الحاسوب، (2) نور فتريه أيو تونجونج ساري، ماجستير في .علوم الحاسوب

الكلمات الرئيسية :التنبؤ بالعيوب، اختلال التوازن في الفئات، الغابة العشوائية، SMOTE، الترشيح المستند إلى الخصائص.

يمكن أن يؤدي اختلال التوازن في الفئات داخل مجموعات بيانات التنبؤ بعيوب البرمجيات إلى جعل النموذج يتجاهل الفئة الأقل تمثيلًا. تهدف هذه الدراسة إلى تقييم أداء نموذج الغابة العشوائية (*Random Forest*) في ثلاث سيناريوهات: بدون توازن، باستخدام تقنية SMOTE ، واستخدام SMOTE مع ترشيح يعتمد على خصائص الميزات. تم إجراء الاختبارات على 13 مجموعة بيانات بنسب مختلفة بين التدريب والاختبار (من 90:10 إلى 70:30)، باستخدام مقاييس الدقة و .*F1-score* أظهرت النتائج أن دمج SMOTE مع الترشيح أعطى أعلى *F1-score* بنسبة 90.72% على مجموعة بيانات PC2 بنسبة تقسيم 70:30، كما أظهر أداءً أكثر استقرارًا على نفس النسبة بمتوسط دقة 78.32% وانحراف معياري قدره 2.38%. تعزز هذه الطريقة قدرة النموذج على الكشف عن العيوب بشكل كبير مقارنة بالسيناريوهين الآخرين. وعليه، فإن هذا النهج يثبت فعاليته في معالجة اختلال توازن الفئات دون التسبب في ضجيج زائد، كما يعزز موثوقية التنبؤ في البرمجيات.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam dunia digital yang serba cepat saat ini, perangkat lunak telah menjadi bagian penting dari kehidupan sehari-hari, menggerakkan segala sesuatu mulai dari ponsel pintar dan laptop hingga sistem industri yang kompleks. Seiring dengan meningkatnya ukuran dan kompleksitas aplikasi perangkat lunak, muncul pula tantangan seperti kerentanan keamanan, dan ketidakpuasan pengguna. Konsekuensi *defect* perangkat lunak dapat bervariasi dari ketidaknyamanan kecil hingga peristiwa yang sangat merugikan, tergantung pada jenis aplikasi dan tingkat keparahan masalahnya (Qualetics, 2021).

Untuk mengurangi risiko-risiko ini dan memastikan keandalan serta kualitas produk perangkat lunak, banyak perusahaan menginvestasikan sumber daya yang besar untuk memastikan produk perangkat lunak mereka berkualitas tinggi. Hal ini sejalan dengan nilai-nilai yang diajarkan dalam Alqur'an, sebagaimana Allah SWT berfirman dalam Surah Al-Baqarah Ayat 267

يٰٓاَيُّهَا الَّذِيْنَ اٰمَنُوْٓا اَنْفِقُوْا مِنْ طَيِّبٰتِ مَا كَسَبْتُمْ وَمِمَّآ اَخْرَجْنَا لَكُمْ مِّنَ الْاَرْضِ ۖ وَلَا تَيَمَّمُوا الْخَبِيْثَ مِنْهُ

تُنْفِقُوْنَ وَلَسْتُمْ بِاٰخِذِيْهِ اِلَّآ اَنْ تُغْمِضُوْا فِيْهِ ۗ وَاعْلَمُوْٓا اَنَّ اللّٰهَ غَنِيٌّ حَمِيْدٌ

"*Wahai orang-orang yang beriman, infakkanlah sebagian dari hasil usahamu yang baik-baik dan sebagian dari apa yang Kami keluarkan dari bumi untukmu. Janganlah kamu memilih yang buruk untuk kamu infakkan, padahal kamu tidak mau mengambilnya, kecuali dengan memicingkan mata (enggan) terhadapnya. Ketahuilah bahwa Allah Mahakaya lagi Maha Terpuji.*" *(QS. Al Baqarah: 267).*

Ayat di atas menegaskan pentingnya memberikan yang terbaik dan menjaga kualitas, bukan hanya dalam konteks amal, tetapi juga dalam setiap aspek kehidupan, termasuk dalam menghasilkan produk perangkat lunak yang berkualitas dan andal. Islam mengajarkan bahwa pelayanan yang baik dan berkualitas adalah bentuk tanggung jawab, baik kepada sesama manusia maupun kepada Allah SWT.

Dalam konteks menjaga kualitas, baik dalam pelayanan maupun hasil pekerjaan, prinsip memberikan yang terbaik juga tercermin dalam proses memastikan perangkat lunak bebas dari *defect*. Upaya ini sering kali dilakukan melalui proses manual yang melibatkan pemeriksaan langsung oleh pengembang atau tim penguji untuk memastikan perangkat lunak berfungsi sebagaimana mestinya. Langkah-langkah manual tersebut mencakup pengujian langsung terhadap fungsionalitas, antarmuka, dan pengalaman pengguna, serta tinjauan kode secara mendetail untuk mengidentifikasi kesalahan sintaksis atau logis (Dasari Rathna Nagabhushan, 2024).

Namun, proses manual untuk mengidentifikasi dan memperbaiki *defect* dapat memakan waktu, mahal, dan rentan terhadap kesalahan manusia (Qualetics, 2021). Di sinilah prediksi *defect* perangkat lunak memainkan peran penting, dengan memanfaatkan teknik pembelajaran mesin untuk secara otomatis mengidentifikasi potensi *defect* dalam kode perangkat lunak sebelum muncul sebagai masalah nyata (Janarthanan, 2021; Thomas & Kaliraj, 2024). Dengan cara ini, pengembang dapat mencegah masalah lebih awal, sehingga meningkatkan kualitas dan keandalan perangkat lunak.

Meskipun prediksi *defect* perangkat lunak telah banyak dikembangkan, masih terdapat tantangan dalam ketidakseimbangan kelas (*class imbalance*) pada *dataset* yang dapat mengakibatkan model lebih cenderung pada kelas mayoritas dan mengabaikan kelas minoritas (Arun & Lakshmi, 2022). Selain itu, penelitian sebelumnya juga menyoroti pentingnya evaluasi model yang tepat agar prediksi yang dihasilkan dapat digunakan secara efektif dalam dunia nyata (Moussa & Sarro, 2022).

Penelitian terdahulu telah menggunakan berbagai metode seperti *Random Forest* (RF), *Support Vector Machine* (SVM), dan *deep learning* untuk meningkatkan akurasi prediksi *defect* perangkat lunak. Namun, meskipun metode-metode ini efektif, beberapa studi, seperti Bennin *et al* (2022), menunjukkan bahwa ketidakseimbangan kelas dalam *dataset* masih menjadi tantangan besar yang memengaruhi kinerja prediksi. Hal ini menyebabkan model lebih cenderung pada kelas mayoritas, mengakibatkan performa yang rendah dalam mengidentifikasi *defect*.

Sebagai solusi, penelitian ini menggunakan pendekatan penyeimbangan data yang mempertimbangkan karakteristik fitur, sehingga seleksi sampel tidak dilakukan secara acak, melainkan berdasarkan informasi relevan dari fitur. Beberapa pendekatan penyeimbangan data yang ada seperti SMOTE dan *undersampling* sering kali mengabaikan informasi penting dari fitur dalam *dataset*. Hal ini dapat menyebabkan hilangnya informasi relevan atau bahkan *overfitting*. Oleh karena itu, pendekatan yang mempertimbangkan karakteristik fitur dalam proses penyeimbangan data menjadi penting untuk dijelajahi. Dengan

mempertimbangkan karakteristik fitur dalam proses penyeimbangan data, penelitian ini diharapkan dapat meningkatkan kinerja model *Random Forest* dalam mengidentifikasi *defect* secara lebih seimbang dan akurat.

1.2 Pernyataan Masalah

Berdasarkan penjelasan pada latar belakang, dapat dirumuskan masalah yang dijadikan sebagai fokus penelitian yaitu :

1. Bagaimana algoritma *Random Forest* dapat digunakan secara efektif dalam memprediksi *defect* perangkat lunak?
2. Bagaimana pengaruh penyeimbangan data yang berdasarkan karakteristik distribusi fitur, terhadap kinerja model *Random Forest* dalam melakukan prediksi *defect* perangkat lunak?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah, tujuan dari penelitian yaitu :

1. Membangun model prediksi *defect* perangkat lunak menggunakan model *Random Forest.*
2. Mengevaluasi pengaruh penyeimbangan data berbasis karakteristik fitur terhadap performa model prediksi *defect* perangkat lunak.

1.4 Batasan Masalah

Penelitian ini dibatasi pada pemanfaatan 13 *dataset defect* perangkat lunak dari NASA *Metrics Data Program* (MDP) yang tersedia melalui situs https://www.kaggle.com/datasets/aczy156/software-defect-prediction-nasa/data.

Setiap *dataset* terdiri dari sejumlah fitur numerik serta satu kolom target yang menunjukkan status *defect* perangkat lunak.

Praproses data difokuskan pada dua tahap utama yang berlandaskan karakteristik fitur. Pertama, dilakukan transformasi distribusi menggunakan *logarithmic scaling* terhadap fitur numerik dengan nilai *skewness* > 0,5. Transformasi ini bertujuan untuk mengurangi kemencengan distribusi agar mendekati normal, sehingga fitur menjadi lebih representatif dalam proses pembelajaran.

Kedua, dilakukan penyeimbangan data menggunakan SMOTE berdasarkan distribusi kelas pada fitur target. Penyeimbangan ini penting karena sebagian besar *dataset* memiliki jumlah sampel yang tidak seimbang antara kelas *defect* dan *non-defect*.

Dengan batasan tersebut, penelitian ini difokuskan untuk menganalisis pengaruh penyeimbangan data berbasis karakteristik fitur melalui transformasi distribusi dan *oversampling* terhadap kinerja algoritma *Random Forest* dalam memprediksi *defect* perangkat lunak.

1.5 Manfaat Penelitian

Adapun manfaat yang diharapkan dari penelitian ini adalah dapat dapat membantu dalam mendeteksi *defect* perangkat lunak secara lebih dini dengan menggunakan algoritma *Random Forest* dan pendekatan penyeimbangan data berbasis karakteristik fitur. Deteksi yang lebih awal ini diharapkan dapat mengurangi biaya dan waktu yang dibutuhkan untuk perbaikan perangkat lunak, serta meningkatkan efisiensi dalam proses pengembangan.

Penelitian ini dapat menjadi referensi bagi studi lanjutan di bidang prediksi *defect* perangkat lunak dan *machine learning*, khususnya terkait dengan penanganan masalah ketidakseimbangan data berdasarkan analisis karakteristik fitur. Sementara itu, secara praktis, hasil dari penelitian ini diharapkan dapat membantu perusahaan dalam meningkatkan keandalan perangkat lunak yang dikembangkan serta mengoptimalkan pemanfaatan sumber daya yang dimiliki dalam proses produksi perangkat lunak.

BAB II

STUDI PUSTAKA

2.1 Penelitian Terkait

Dalam rekayasa perangkat lunak, prediksi *defect* perangkat lunak memiliki peran krusial dalam memastikan kualitas dan keandalan produk. Kemajuan dalam pembelajaran mesin, khususnya metode *ensemble* seperti *Random Forest*, telah terbukti efektif dalam meningkatkan prediksi *defect* perangkat lunak. Studi ini bertujuan untuk meninjau penelitian-penelitian yang menunjukkan efektivitas *Random Forest* dalam menangani kompleksitas data perangkat lunak, serta strategi lain yang dapat meningkatkan performa prediksi.

Pada penelitian Malhotra *et al.* (2022), mengungkapkan tren penggunaan metode *ensemble* dalam prediksi *defect* perangkat lunak, dengan *Random Forest* sebagai salah satu teknik yang paling populer. Penelitian ini menekankan bahwa metode *ensemble* seperti *Random Forest*, *boosting*, dan *bagging* dapat meningkatkan akurasi prediksi *defect* perangkat lunak. Temuan ini menunjukkan bahwa *Random Forest*, dengan kekuatan *ensemble*-nya, mampu mengatasi berbagai tantangan dalam prediksi, terutama pada data yang tidak seimbang dan berdimensi tinggi. Kajian ini menjadi landasan penting untuk pengembangan lebih lanjut metodologi prediksi *defect* perangkat lunak.

Dalam ranah prediksi *defect* perangkat lunak, studi terbaru juga menekankan pentingnya teknik pembelajaran mesin yang canggih untuk meningkatkan akurasi dan efisiensi dalam identifikasi *defect*. Thomas *et al.* (2024)

memperkenalkan pendekatan baru yang menggunakan model *Random Forest* yang dioptimalkan, yang secara signifikan meningkatkan prediksi kesalahan perangkat lunak. Penelitian mereka menyoroti dampak serius dari *defect* perangkat lunak, yang dapat menyebabkan kegagalan operasional dan kerugian finansial yang signifikan. Dengan memberikan analisis mendalam tentang berbagai metodologi pembelajaran mesin, para penulis memberikan wawasan berharga mengenai optimasi model prediksi *defect*, sehingga menjawab kebutuhan mendesak dalam jaminan kualitas perangkat lunak.

Selanjutnya, penelitian oleh Alhumam (2022) berfokus pada pemilihan fitur untuk meningkatkan efisiensi dan akurasi *Random Forest*. Alhumam memperkenalkan kerangka kerja berbasis *entropy* untuk pemilihan fitur yang secara efektif menyaring metrik perangkat lunak yang paling relevan. Teknik ini membantu mengurangi dimensionalitas data, sehingga algoritma *Random Forest* dapat bekerja lebih optimal sekaligus mengurangi risiko *overfitting*.

Selain itu, Ali *et al.* (2024) mengusulkan *framework* komprehensif yang mengintegrasikan pemilihan fitur yang ditingkatkan dengan teknik pembelajaran mesin *ensemble*. *Five-stage framework* ini dirancang untuk mengatasi tantangan dalam prediksi *defect* perangkat lunak dengan memanfaatkan *dataset* yang telah dibersihkan dari NASA, memastikan integritas dan relevansi data. Studi ini menekankan pentingnya mengidentifikasi modul yang *defect* sejak awal dalam siklus pengembangan perangkat lunak, sehingga memungkinkan intervensi tepat waktu untuk mengurangi risiko yang terkait dengan *defect* perangkat lunak

Penelitian lain oleh Soe *et al.* (2018) mendukung hasil ini dengan menunjukkan bahwa *Random Forest* mengungguli metode statistik tradisional dalam prediksi *defect* perangkat lunak, terutama pada *dataset* yang besar dan kompleks. Mereka menemukan bahwa algoritma ini sangat andal dalam menangani data yang berdimensi tinggi, yang sering kali menjadi masalah dalam analisis perangkat lunak.

Pendekatan inovatif lain dalam peningkatan akurasi prediksi *defect* perangkat lunak ditawarkan oleh Thapa *et al.* (2020), yang menggabungkan teknik penambangan aturan (*Atomic Rule Mining*) dengan *Random Forest*. Kombinasi ini memungkinkan algoritma tidak hanya meningkatkan akurasi, tetapi juga memberikan wawasan lebih mendalam tentang pola-pola *defect* perangkat lunak. Pendekatan ini penting dalam memberikan pemahaman tambahan yang tidak hanya fokus pada prediksi, tetapi juga pada identifikasi penyebab *defect* perangkat lunak, yang sangat berguna bagi tim rekayasa perangkat lunak.

Secara keseluruhan, penelitian-penelitian ini menunjukkan bahwa *Random Forest* adalah metode yang sangat andal untuk prediksi *defect* perangkat lunak. Oleh karena itu, penelitian ini akan menggunakan *Random Forest* sebagai metode utama dalam prediksi *defect* perangkat lunak, dengan pendekatan SMOTE agar data minoritas seimbang dengan data mayoritas.

Tabel 2. 1 Penelitian terdahulu

No	Nama Peneliti	Judul Penelitian	Metode	Dataset	Hasil Penelitian
1.	Nikhil Saji Thomas, S. Kaliraj	*An Improved and Optimized Random Forest Based Approach to Predict the*	*Random Forest, Feature Selection, Random Search*	NASA JM1	Akurasi 82,96%, *F1-score* 89,53%

No	Nama Peneliti	Judul Penelitian	Metode	Dataset	Hasil Penelitian
		Software Faults			
2.	Abdulaziz Alhumam	*Effective Prediction of Software Defects using Random-tree Entropy based Feature Selection Framework*	*Decision Tree, Random Forest, Naïve Bayes, Radial Basis Function, SVM, K-NN*	NASA PC1	*Random Forest* mencapai akurasi 97,76%
3.	Misbah Ali, Tehseen Mazhar, Amal Al-Rasheed, Tariq Shahzad, Yazeed Yasin Ghadi dan Muhammad Amir Khan	*Enhancing Software Defect Prediction: A Framework with Improved Feature Selection and Ensemble Machine Learning*	*Random Forest, SVM, Naïve Bayes*	NASA CM1, JM1, MC2, MW1, PC1, PC3, dan PC4	*Framework* IECGA membantu *Random Forest* mencapai akurasi maksimum 95,1%
4.	Yan Naung Soe, Paulus Insap Santosa, Rudy Hartanto	*Software Defect Prediction Using Random Forest Algorithm*	*Random Forest*	NASA AR1, AR3, AR4, AR5 , AR6, KC1, KC2, KC3, PC1, PC2, PC3 DAN PC4.	Akurasi tertinggi 85,96% pada *dataset* KC1
5.	Suroj Thapa, Abeer Alsadoon, P.W.C. Prasad, Thair Al-Dala'in, Tarik A. Rashid	*Software Defect Prediction Using Atomic Rule Mining and Random Forest*	*Atomic Rule Mining + Random Forest*	ANT, NASA CM1, JM1, KC3, KC4, MC1, MC2, MW1, PC2, PC4, PC5,	Akurasi 90,09%, waktu pemrosesan berkurang 54,14%
6.	Liangjun Jiang, Zerui Yang, Donghai Wang, Haimei Gong, Juan Li,	*Diabetes prediction model for unbalanced community follow-up data set based on*	Log Transformation, SMOTE, ADASYN, *RandomOverSampler*, *RandomUnderSampler* dan *Near Miss* untuk *balancing, s*eleksi fitur (ANOVA, *Mutual Info*, RFE), model klasifikasi	Data *real-world* dari sistem manajemen layanan komunitas di Distrik Haizhu,	Distribusi fitur yang sangat *skewed* diperbaiki dengan *log transformation*, yang berpengaruh

No	Nama Peneliti	Judul Penelitian	Metode	Dataset	Hasil Penelitian
	Jing Wang and Lei Wang	*optimal feature selection and scorecard*	(*Random Forest* (RF), *Gradient Boosting Decision Tree* (GBDT), *eXtreme Gradient Boosting* (XGB), *K-Nearest Neighbors* (KNN), *Multilayer Perceptron* (MLP), and *Ensemble Learning* (VC) - Pembuatan *scorecard* berbasis WOE dan *Logistic Regression*	Guangzhou, Tiongkok (2016–2023)	signifikan dalam meningkatkan stabilitas model klasifikasi, model *Random Forest* menghasilkan performa tertinggi dibanding model lain (akurasi 91.41%, *F1-score* 90.91%)

2.2 *Defect* Perangkat Lunak

Defect perangkat lunak merujuk pada masalah kualitas yang ditemukan setelah perangkat lunak dirilis dan digunakan oleh pengguna akhir. Dalam konteks rekayasa perangkat lunak, *defect* sering dianggap sebagai kesalahan atau ketidaksesuaian terhadap spesifikasi yang menyebabkan perangkat lunak tidak berfungsi sebagaimana mestinya. Roger S. Pressman (2010) dalam *Software Engineering: A Practitioner's Approach* membedakan antara *error* dan *defect*, di mana *error* adalah kesalahan yang ditemukan sebelum perangkat lunak dirilis, sedangkan *defect* baru teridentifikasi setelah perangkat lunak digunakan. Istilah ini juga sering digunakan secara bergantian dengan *fault* atau *bug*, tergantung pada sudut pandang teknis dan tahapan pengembangannya. Oleh karena itu, pengelolaan *defect* menjadi bagian penting dalam proses jaminan kualitas perangkat lunak untuk memastikan keandalan sistem setelah implementasi.

Defect ini dapat dikategorikan lebih lanjut berdasarkan karakteristik dan tahap kemunculannya. Di tingkat *internal*, *defect* berupa *fault* atau kesalahan yang terjadi selama pengembangan. Contohnya adalah kesalahan logika yang

menghasilkan *output* yang salah meskipun kode berjalan tanpa mengalami *crash*. Di tingkat eksternal, *defect* tampak sebagai pelanggaran terhadap fungsi yang diharapkan, yang mengakibatkan ketidakpuasan pengguna (Taskeen *et al*., 2023).

Contoh *defect* perangkat lunak meliputi kesalahan sintaksis, yang mencegah kompilasi, serta kesalahan *runtime*, seperti pembagian dengan nol (*division by zero*). Kesalahan logika dapat menghasilkan *output* yang tidak terduga meskipun perangkat lunak berfungsi tanpa kendala. Dampak *defect* ini dapat berkisar dari ketidaknyamanan kecil hingga kegagalan sistem besar yang dapat membahayakan data dan keselamatan pengguna (Cai *et al*., 2020; Chen *et al*., 2022).

Selain mengganggu fungsi, *defect* perangkat lunak juga mempengaruhi kualitas keseluruhan produk, meningkatkan biaya pemeliharaan, dan menurunkan kepuasan pengguna. Dalam sistem kritis, seperti di bidang kesehatan atau keuangan, *defect* dapat memiliki konsekuensi serius, termasuk kerugian finansial atau bahkan membahayakan nyawa manusia (Naseem *et al*., 2020). Oleh karena itu, deteksi dan perbaikan dini *defect* merupakan bagian penting dari proses jaminan kualitas dalam pengembangan perangkat lunak (Liu *et al*., 2021).

Secara keseluruhan, *defect* perangkat lunak adalah tantangan yang perlu dikelola dengan baik. Identifikasi dan klasifikasi yang tepat memungkinkan penerapan strategi prediksi dan pencegahan yang efektif, yang pada gilirannya menghasilkan produk perangkat lunak yang lebih berkualitas dan meningkatkan kepuasan pengguna.

2.3 Prediksi *Defect* Perangkat Lunak

Prediksi *defect* perangkat lunak merupakan proses penting dalam rekayasa perangkat lunak yang bertujuan untuk mengidentifikasi *defect* di awal siklus pengembangan guna meningkatkan kualitas perangkat lunak dan mengurangi biaya (Li *et al*., 2018). Proses ini melibatkan analisis data *defect* historis dari rilis sebelumnya, menggunakan metrik seperti kompleksitas kode dan jumlah baris kode, untuk mengidentifikasi bagian perangkat lunak yang berisiko tinggi mengalami *defect* (Cui *et al*., 2022; Taskeen *et al*., 2023). Model-model prediksi *defect* perangkat lunak, yang dibangun menggunakan algoritma pembelajaran mesin seperti *Random Forest*, mampu mengklasifikasikan modul perangkat lunak sebagai *defect* atau bebas *defect* (Zhao *et al*., 2023).

Salah satu tantangan utama dalam prediksi *defect* perangkat lunak adalah ketidakseimbangan kelas, di mana jumlah modul bebas *defect* jauh lebih banyak dibandingkan dengan yang *defect*, yang dapat menyebabkan tingginya tingkat positif palsu. Untuk mengatasi masalah ini, teknik seperti *oversampling, undersampling* atau *cost-sensitive learning* perlu diterapkan untuk meningkatkan akurasi model (Xie *et al*., 2021).

2.4 *Skewness*

Dalam analisis statistik, *skewness* atau kemencengan adalah ukuran untuk mengetahui arah dan derajat ketidaksimetrian suatu distribusi data terhadap distribusi normal. *Skewness* dinyatakan sebagai momen tersentralisasi orde ketiga yang dinormalisasi, yang secara matematis didefinisikan sebagai berikut:

$$\text{Skewness}(X) = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s} \right)^3 \quad (2.1)$$

Keterangan:
n : Jumlah data,
x_i : Nilai data ke-i,
$\bar{x}$: Rata-rata data,
s : Standar deviasi sampel.
$\sum_{i=1}^{n}$: Penjumlahan dari i=1 sampai n

Distribusi normal memiliki *skewness* sebesar 0, yang berarti kurva distribusinya simetris di kedua sisi rata-rata. Jika nilai *skewness* < 0, distribusi disebut *left-skewed* (kemencengan ke kiri), yang menunjukkan bahwa terdapat lebih sedikit data di sisi kiri rata-rata dibandingkan sisi kanan. Secara visual, ini berarti ekor kiri lebih panjang karena adanya sejumlah data dengan nilai yang sangat kecil. Sebaliknya, jika *skewness* > 0, distribusi disebut *right-skewed* (kemencengan ke kanan), dengan lebih sedikit data di sisi kanan rata-rata dan ekor kanan lebih panjang akibat keberadaan data dengan nilai sangat besar (Jiang et al., 2024).

Dalam konteks pembelajaran mesin, fitur dengan *skewness* tinggi dapat menyebabkan bias pada model karena distribusi data yang tidak representatif. Oleh karena itu, penting untuk mengidentifikasi dan mengoreksi *skewness* sebelum melatih model klasifikasi seperti *Random Forest*, terutama jika data akan digunakan dalam proses penyeimbangan kelas.

2.5 *Logarithmic Scaling*

Distribusi data yang tidak simetris atau memiliki kemiringan (*skewness*) tinggi dapat memengaruhi performa model prediktif, terutama dalam menangani nilai-nilai ekstrem. Salah satu pendekatan yang digunakan untuk mengurangi efek *skewness* adalah *logarithmic scaling*, yaitu transformasi nonlinier yang mengubah nilai fitur numerik menjadi bentuk logaritma.

Transformasi ini bertujuan untuk memperlebar jarak antar nilai kecil dan memperkecil jarak antar nilai besar, sehingga distribusi menjadi lebih simetris. Fungsi logaritmik yang umum digunakan adalah sebagai berikut:

$$x_{\text{scaled}} = \log(x + 1) \tag{2.2}$$

Keterangan:
x : Nilai asli dari fitur numerik,
x_{scaled} : Nilai hasil transformasi.

Menurut Jiang (2024), *logarithmic scaling* diterapkan pada fitur dengan nilai *skewness* lebih dari 0,5 dan terbukti efektif dalam mengurangi tingkat *skewness* secara signifikan. Transformasi ini membantu menstabilkan distribusi data dan meningkatkan kualitas input model, khususnya pada algoritma berbasis pohon keputusan seperti *Random Forest*. Oleh karena itu, *logarithmic scaling* merupakan langkah penting dalam *preprocessing* data yang bersifat *skewed*.

2.6 SMOTE

Setelah distribusi fitur diperbaiki, langkah berikutnya dalam menangani masalah ketidakseimbangan kelas adalah menerapkan SMOTE (*Synthetic Minority Over-sampling Technique*). SMOTE merupakan teknik penyeimbangan

data yang bekerja dengan cara menghasilkan sampel sintetis baru dari kelas minoritas, bukan sekadar menduplikasi data yang sudah ada (Hairani et al., 2024).

SMOTE menciptakan contoh baru dengan melakukan interpolasi antara sampel minoritas yang ada dan tetangga terdekatnya. Teknik ini tidak hanya menjaga keragaman data, tetapi juga mengurangi risiko *overfitting* akibat pengulangan data yang sama. SMOTE bermanfaat dalam konteks prediksi *defect* perangkat lunak, di mana jumlah data dari kelas *defect* sering kali jauh lebih sedikit dibandingkan kelas *non-defect*, namun informasi dari kelas ini justru krusial dalam proses pengambilan keputusan. Dengan menyeimbangkan distribusi kelas secara lebih realistis, SMOTE dapat membantu meningkatkan sensitivitas model terhadap pola yang berkaitan dengan *defect* perangkat lunak tanpa mengorbankan data yang ada.

Namun demikian, beberapa penelitian menunjukkan bahwa keseimbangan sempurna (rasio 1:1) tidak selalu menghasilkan model terbaik. Peng *et al.* (2022) mengemukakan bahwa dalam beberapa kondisi, rasio seperti 2:1 atau 3:1 justru dapat meningkatkan stabilitas dan kinerja model, terutama saat data sintetis yang dihasilkan kurang representatif. Oleh karena itu, penentuan rasio kelas dalam penerapan SMOTE perlu mempertimbangkan konteks distribusi data dan kualitas sampel sintetis yang dihasilkan.

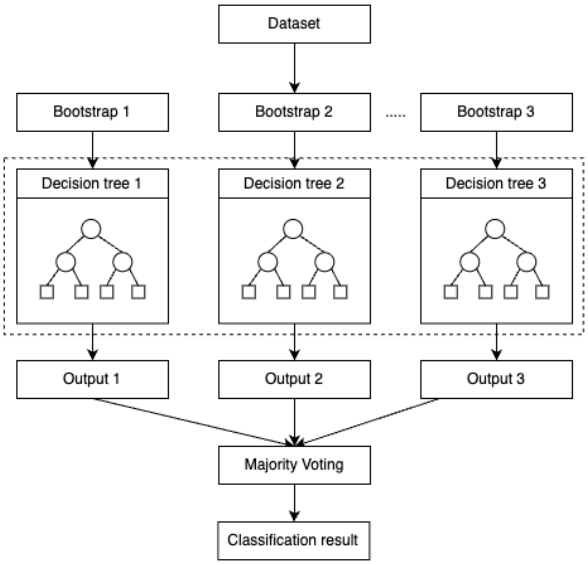
2.7 *Random Forest*

Random Forest (RF) adalah metode pembelajaran *ensemble* yang terutama digunakan untuk tugas klasifikasi dan regresi. Diperkenalkan oleh Leo Breiman pada tahun 2001, RF bekerja dengan membangun banyak pohon keputusan

selama proses pelatihan dan menghasilkan *output* berdasarkan *voting* mayoritas dari setiap pohon untuk klasifikasi, atau rata-rata untuk regresi (Suci Amaliah *et al*., 2022). Metode ini terkenal karena kemampuannya menangani ruang fitur berdimensi tinggi dan memberikan kerangka prediksi yang tangguh bahkan dalam kehadiran data yang bising.

Prinsip utama dari *Random Forest* adalah konsep *bootstrap aggregating* atau *bagging*, yang bertujuan untuk meningkatkan stabilitas dan akurasi algoritma pembelajaran mesin dengan mengurangi varians dan mencegah *overfitting* (Dewi *et al*., 2012). Setiap pohon keputusan dilatih pada subset acak data, yang dipilih dengan penggantian (*bootstrapping*), dan setiap simpul pohon dipisahkan berdasarkan *subset* acak dari fitur (Hajiar Yuliana, 2024). Proses seleksi fitur ini membuat *Random Forest* tahan terhadap *dataset* berdimensi tinggi dan mencegah *overfitting*, yang sering terjadi pada pohon keputusan tunggal.

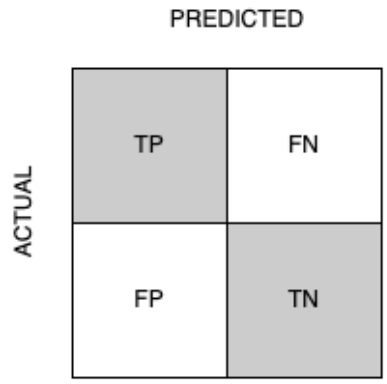
Selain itu, *Random Forest* juga menyediakan informasi mengenai tingkat kepentingan masing-masing fitur (*feature importance*) berdasarkan kontribusinya terhadap proses pemisahan pada tiap pohon. Informasi ini dapat dimanfaatkan dalam proses seleksi fitur maupun untuk analisis lebih lanjut terhadap struktur data. Aguilar-Ruiz (2024) menyatakan bahwa pemilihan fitur berdasarkan *feature importance* sangat membantu dalam meningkatkan akurasi klasifikasi, terutama pada kasus dengan ketidakseimbangan kelas. Pendekatan ini memungkinkan model untuk lebih fokus pada dimensi yang benar-benar relevan, serta mengurangi pengaruh fitur yang tidak informatif terhadap proses klasifikasi.



Gambar 2. 1 Desain sistem *Random Forest*

2.8 *Confusion Matrix*

Confusion matrix adalah alat yang sering digunakan untuk mengevaluasi performa model klasifikasi, termasuk model *Random Forest* dalam memprediksi *defect* perangkat lunak (Istiqamah & Rijal, 2024). Matriks ini menggambarkan hasil prediksi model dalam bentuk tabel, di mana setiap baris mewakili kelas sebenarnya, dan setiap kolom mewakili prediksi model.



Gambar 2. 2 *Confusion Matrix*

Ada empat komponen utama dalam *confusion matrix*:

1. *True Positive* (TP): Kasus di mana model mengklasifikasikan dengan benar bahwa perangkat lunak memiliki *defect* (prediksi positif dan sebenarnya positif) (Ary Prandika Siregar *et al*., 2023).
2. *False Positive* (FP): Kasus di mana model salah mengklasifikasikan perangkat lunak sebagai *defect*, padahal sebenarnya tidak ada *defect* (prediksi positif dan sebenarnya negatif) (Pahlevi *et al*., 2023).
3. *True Negative* (TN): Kasus di mana model mengklasifikasikan dengan benar bahwa perangkat lunak tidak memiliki *defect* (prediksi negatif dan sebenarnya negatif).
4. *False Negative* (FN): Kasus di mana model salah mengklasifikasikan perangkat lunak tidak memiliki *defect*, padahal sebenarnya ada *defect* (prediksi negatif dan sebenarnya positif).

Metrik penting yang dihasilkan dari *confusion matrix* meliputi:

1. *Accuracy*: Proporsi prediksi yang benar terhadap total data.

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.3)$$

2. *F1-Score*: Rata-rata harmonik antara *precision* dan *recall*, digunakan untuk mengevaluasi kinerja model saat ada ketidakseimbangan kelas.

$$F1 - Score = \frac{2 \times \text{Presisi} \times \text{Recall}}{\text{Presisi} + \text{Recall}} \quad (2.4)$$

2.9 *K-fold Cross Validation*

K-Fold Cross Validation adalah teknik evaluasi model yang digunakan untuk menilai performa model pada data yang tidak terlihat selama pelatihan. Teknik ini melibatkan pembagian *dataset* ke dalam K bagian (*folds*). Pada setiap iterasi, satu bagian digunakan sebagai data uji (Hafid, 2023), sementara sisanya digunakan sebagai data latih. Proses ini diulang sebanyak K kali, sehingga setiap bagian *dataset* digunakan sekali sebagai data uji.

Keunggulan *K-Fold Cross Validation* adalah kemampuannya untuk mengurangi variabilitas hasil evaluasi model dengan memberikan perkiraan yang lebih akurat dari performa model pada data yang belum pernah dilihat. Teknik ini juga membantu memastikan bahwa model tidak hanya bekerja baik pada satu *subset* data (Azis *et al*., 2020) tetapi dapat bekerja secara umum pada berbagai *subset* data.

Proses *K-Fold* yang umum digunakan meliputi:

1. Memecah data menjadi K *subset*.
2. Melatih model sebanyak K kali, menggunakan K-1 *subset* sebagai data latih dan satu subset sebagai data uji.
3. Menghitung rata-rata performa model dari setiap iterasi.

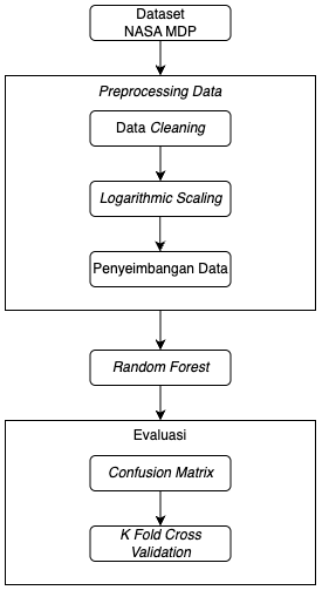
Hasil dari *K-Fold Cross Validation* umumnya digunakan untuk mengukur akurasi dan *f1-score*, serta membandingkan performa berbagai model (Azis *et al*., 2020).

BAB III

METODE PENELITIAN

3.1 Desain Penelitian

Desain penelitian ini menggambarkan proses yang dilakukan untuk membangun model prediksi *defect* perangkat lunak menggunakan metode *Random Forest*. Setiap langkah dalam penelitian ini dirancang untuk memastikan hasil prediksi yang optimal melalui pengolahan data. Gambar berikut menunjukkan alur sistem yang mencakup seluruh proses mulai dari *input* data hingga evaluasi akhir model.



Gambar 3. 1 Desain Penelitian

3.2 *Dataset*

Dataset yang digunakan untuk prediksi *defect* perangkat lunak berasal dari NASA MDP *(Metric Data Program)*. *Dataset* ini berisi berbagai metrik perangkat lunak yang berkaitan dengan kompleksitas kode dan jumlah *defect* perangkat

lunak. Pada penelitian ini menggunakan 13 *dataset* NASA MDP yaitu CM1, JM1, KC1, KC3, KC4, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5. Berikut merupakan fitur dari *dataset* KC1:

Tabel 3. 1 Penjelasan fitur *dataset KC1*

No	Fitur	Keterangan
1.	LOC_BLANK	Jumlah baris kosong dalam kode program.
2.	BRANCH_COUNT	Jumlah percabangan dalam kode program.
3.	LOC_CODE_AND COMMENT	Jumlah baris yang berisi kode dan komentar.
4.	LOC_COMMENTS	Jumlah baris yang hanya berisi komentar.
5.	CYCLOMATIC_COMPLEXITY	Kompleksitas siklomatik berdasarkan metode McCabe, yang mengukur jumlah jalur independen dalam kode.
6.	DESIGN_COMPLEXITY	Kompleksitas desain berdasarkan metode McCabe, mengukur kompleksitas struktural dari desain program.
7.	ESSENTIAL_COMPLEXITY	Kompleksitas esensial berdasarkan metode McCabe, yang menunjukkan seberapa sulit kode dapat direstrukturisasi untuk menyederhanakan logika.
8.	LOC_EXECUTABLE	Jumlah baris kode yang benar-benar dapat dieksekusi.
9.	HALSTEAD_CONTENT	Metrik kompleksitas *Halstead* yang mengukur jumlah informasi dalam program.
10.	HALSTEAD_DIFFICULTY	Tingkat kesulitan dalam memahami atau memodifikasi kode berdasarkan teori Halstead.
11.	HALSTEAD_EFFORT	Perkiraan upaya atau kerja yang diperlukan untuk menulis atau memahami program, menurut teori *Halstead*.
12.	HALSTEAD_ERROR_EST	Perkiraan jumlah kesalahan dalam kode berdasarkan kompleksitas *Halstead*.
13.	HALSTEAD_LENGTH	Panjang program yang dihitung dari jumlah total operator dan operand dalam kode.
14.	HALSTEAD_LEVEL	Tingkat abstraksi kode berdasarkan perbandingan antara panjang dan kesulitan program.
15.	HALSTEAD_PROG_TIME	Perkiraan waktu yang dibutuhkan untuk menulis kode berdasarkan metrik *Halstead*.
16.	HALSTEAD_VOLUME	Ukuran kompleksitas program yang dihitung berdasarkan jumlah token unik dan total dalam kode.
17.	NUM_OPERANDS	Total jumlah operand dalam kode program.
18.	NUM_OPERATORS	Total jumlah operator dalam kode program.
19	NUM_UNIQUE_OPERANDS	Jumlah operand unik dalam kode program.
20.	NUM_UNIQUE_OPERATORS	Jumlah operator unik dalam kode program.
21.	LOC_TOTAL	Jumlah total baris kode dalam program.
22.	Defective	Indikator apakah kode memiliki *defect* atau tidak, dengan nilai {*false, true*} (false: *non defect*, *true*: *defect*).

Sumber: *Kaggle* (2020)

Tabel 3. 2 Contoh *dataset KC1*

LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	… …	NUM_ OPERATORS	LOC_ TOTAL	Defective
7	9	1	…	46	31	N
2	7	1	…	59	32	N
1	1	0	…	14	9	N
1	13	2	…	99	48	Y
0	1	0	…	5	4	Y
0	3	0	…	7	2	N
0	1	0	…	5	4	N
8	41	5	…	236	123	N
2	1	1	…	39	25	Y
…	…	…	…	…	…	…

Sumber: *Kaggle* (2020)

3.3 *Preprocessing Data*

Preprocessing data merupakan tahap penting dalam membangun model prediksi yang akurat. Pada penelitian ini, *preprocessing* dilakukan dengan beberapa langkah yaitu, data *cleaning,* pengukuran *skewness, logarithmic scaling* dan SMOTE. Berikut merupakan contoh data sebelum dilakukan *preprocessing.*

Tabel 3. 3 Contoh *dataset* sebelum *preprocessing*

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	2	1	0	7	N
2.	0	1	0	16	N
3.	0	1	0	0	N
4.	3	1	0	9	N
5.	1	1	0	18	Y
6.	0	1	0	11	N
7.	1	10	1	71	Y
8.	1	12	1	78	Y
9.	0	1	0	12	N
10.	0	1	0	28	Y
11.	1	1	0	14	N
12.	3	5	0	30	Y
13.	2	5	0	15	N
14.	2	1	0	13	N
15.	7	17	4	50	N
16.	0	1	0	1	N
17.	3	7	0	20	Y
18.	1	1	0	4	N
19.	1	1	0	4	N
20.	0	1	0	1	N

Sumber: diolah peneliti (2025)

3.3.1 Data *Cleaning*

Pada tahap *data cleaning*, dilakukan beberapa proses untuk memastikan kualitas dan konsistensi data sebelum digunakan dalam pemodelan. Langkah-langkah yang dilakukan meliputi:

1. Pengecekan dan penanganan duplikasi. Data diperiksa untuk mendeteksi keberadaan baris yang duplikat. Baris duplikat yang ditemukan kemudian dihapus untuk menghindari bias dalam proses pelatihan model.
2. Pengecekan *missing values*. *Dataset* diperiksa untuk mengetahui apakah terdapat nilai yang hilang (*missing values*). Jika ditemukan, dilakukan penanganan yang sesuai, seperti penghapusan baris yang tidak lengkap atau imputasi nilai berdasarkan strategi tertentu, tergantung pada konteks dan sebaran datanya.
3. Transformasi tipe data. Pada proses transformasi data, dilakukan perubahan tipe data pada fitur *defective* yang semula berbentuk *boolean/string* menjadi *integer*. Fitur ini merepresentasikan kondisi *defect* atau tidak *defect* pada perangkat lunak, di mana nilai *False* menunjukkan bahwa perangkat lunak tidak memiliki *defect*, dan *True* menunjukkan adanya *defect*. Untuk mempermudah pemrosesan oleh model, nilai *False* dikonversi menjadi 0 (tidak *defect*) dan *True* dikonversi menjadi 1 (*defect*) (Thapa et al., 2020). Transformasi ini diperlukan agar model prediksi dapat mengenali label target secara numerik, sehingga memfasilitasi proses pembelajaran dan evaluasi.

Tabel 3. 4 Contoh *dataset* setelah ditransformasi

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	2	1	0	7	0
2.	0	1	0	16	0
3.	0	1	0	0	0
4.	3	1	0	9	0
5.	1	1	0	18	1
6.	0	1	0	11	0
7.	1	10	1	71	1
8.	1	12	1	78	1
9.	0	1	0	12	0
10.	0	1	0	28	1
11.	1	1	0	14	0
12.	3	5	0	30	1
13.	2	5	0	15	0
14.	2	1	0	13	0
15.	7	17	4	50	0
16.	0	1	0	1	0
17.	3	7	0	20	1
18.	1	1	0	4	0
19.	1	1	0	4	0
20.	0	1	0	1	0

Sumber: diolah peneliti (2025)

3.3.2 Pengukuran *Skewness*

Skewness atau kemencengan adalah ukuran untuk mengetahui sejauh mana distribusi data simetris terhadap nilai rata-ratanya. Untuk mengetahui tingkat *skewness* pada suatu fitur, kita dapat menggunakan rumus *skewness* pada persamaan 2.1

Langkah-langkah perhitungan menggunakan contoh dari kolom LOC_EXECUTABLE = 7, 16, 0, 9, 18, 11, 71, 78, 12, 28, 14, 30, 15, 13, 50, 1, 20, 4, 4, 1

1. Hitung rata-rata

$$\bar{x} = \frac{\sum x_i}{n} \tag{3.1}$$

Keterangan:
$\bar{x}$: Rata-rata dari seluruh data
x_i : Nilai data ke-i

n : Jumlah total data
$\sum x_i$: Penjumlahan seluruh nilai data

$$\bar{x} = \frac{402}{20} = 20.1$$

2. Hitung standar deviasi sampel

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (3.2)$$

Keterangan:
s : Simpangan baku (standar deviasi) dari sampel
n : Jumlah data (ukuran sampel)
x_i : Nilai data ke-i dalam sampel
$\bar{x}$: Rata-rata dari seluruh data sampel

Menghitung selisih kuadrat dari tiap nilai terhadap rata-rata, berikut adalah tabel nilai $(x_i - \bar{x})^2$:

Tabel 3. 5 Hasil perhitungan selisih kuadrat tiap nilai pada rata-rata

No	x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
1.	7	-13,10	171,61
2.	16	-4,10	16,81
3.	0	-20,10	404,01
4.	9	-11,10	123,21
5.	18	-2,10	4,41
6.	11	-9,10	82,81
7.	71	50,90	2590,81
8.	78	57,90	3352,41
9.	12	-8,10	65,61
10.	28	7,90	62,41
11.	14	-6,10	37,21
12.	30	9,90	98,01
13.	15	-5,10	26,01
14.	13	-7,10	50,41
15.	50	29,90	894,01
16.	1	-19,10	364,81
17.	20	-0,10	0,01
18.	4	-16,10	259,21
19.	4	-16,10	259,21
20.	1	-19,10	364,81
Jumlah			9227,8

Sumber: diolah peneliti (2025)

$$s = \sqrt{\frac{9227.8}{20-1}} = \sqrt{\frac{9227.8}{19}} = \sqrt{485.68} \approx 22.04$$

3. Hitung koefisien awal

$$\frac{n}{(n-1)(n-2)} = \frac{20}{(19)(18)} = 0.0585$$

4. Hitung *Z-Score* dan Z^3 untuk masing-masing nilai

Tabel 3. 6 Hasil perhitungan *Z-score* dan Z^3 pada fitur LOC_EXECUTABLE

No	*Z-Score*	Z^3
1.	-0,5944	-0,2100
2.	-0,1860	-0,0064
3.	-0,9120	-0,7587
4.	-0,5036	-0,1277
5.	-0,0952	-0,0008
6.	-0,4129	-0,0704
7.	2,3096	12,320
8.	2,6272	18,135
9.	-0,3675	-0,0496
10.	0,3584	0,0460
11.	-0,2767	-0,0212
12.	0,4492	0,0906
13.	-0,2314	-0,0123
14.	-0,3221	-0,0334
15.	1,3567	2,4974
16.	-0,8666	-0,6510
17.	-0,0045	0,0000
18.	-0,7305	-0,3899
19.	-0,7305	-0,3899
20.	-0,8666	-0,6510

Sumber: diolah peneliti (2025)

5. Jumlahkan seluruh Z^3

$$\sum Z^3 = 29{,}717$$

6. Hitung *skewness*

$$Skewness = 0.0585 \times 29.717 = 1.738$$

Nilai tersebut menunjukkan bahwa distribusi fitur LOC_EXECUTABLE memiliki *skewness* positif (*right-skewed*) dan perlu dilakukan transformasi menggunakan *logarithmic scaling*. Perhitungan serupa dilakukan terhadap fitur

lainnya seperti LOC_BLANK, BRANCH_COUNT, dan LOC_COMMENTS, untuk menentukan apakah fitur tersebut mengalami *skewness* dan apakah perlu dilakukan transformasi lebih lanjut.

Tabel 3. 7 Hasil perhitungan *skewness*

LOC_BLANK	BRANCH_COUNT	LOC_COMMENTS	LOC_EXECUTABLE
2,0171	1,9179	3,7792	1,7378

Sumber: diolah peneliti (2025)

3.3.3 *Logarithmic Scaling*

Berdasarkan hasil analisis pada Tabel 3.7, diketahui bahwa beberapa fitur memiliki nilai *skewness* lebih dari 0,5, yaitu LOC_BLANK, BRANCH_COUNT, LOC_COMMENTS, dan LOC_EXECUTABLE. Oleh karena itu, dilakukan transformasi *logarithmic scaling* terhadap fitur-fitur tersebut guna memperbaiki distribusi data yang miring. Pada subbab ini, ditunjukkan langkah-langkah perhitungan manual *logarithmic scaling* menggunakan persamaan 2.2

Transformasi ini bertujuan untuk memperbesar jarak antar nilai kecil dan mengecilkan jarak antar nilai besar, sehingga distribusi data menjadi lebih simetris dan stabil. Berikut ditampilkan contoh perhitungan manual pada salah satu fitur yang mengalami *skewness* tinggi.

Misalkan kita ambil baris pertama dari *dataset*, yaitu:

Tabel 3. 8 Contoh data baris pertama

LOC_BLANK	BRANCH_COUNT	LOC_COMMENTS	LOC_EXECUTABLE
2	1	0	7

Sumber: diolah peneliti (2025)

Langkah-langkahnya:

1. Transformasi LOC_BLANK:

   $log(1 + 2) = log(3) \approx 1.0986$

2. Transformasi BRANCH_COUNT:

   $log(1 + 1) = log(2) \approx 0.6931$

3. Transformasi LOC_COMMENTS:

   $log(1 + 0) = log(1) = 0$

4. Transformasi LOC_EXECUTABLE:

   $log(1 + 7) = log(8) \approx 2.0794$

Hasil *logarithmic scaling* untuk data baris pertama:

Tabel 3. 9 Hasil *logarithmic scaling* data baris pertama

Fitur	Data Asli	Data setelah *logarithmic scaling*
LOC_BLANK	2	1,0986
BRANCH_COUNT	1	0,6931
LOC_COMMENTS	0	0
LOC_EXECUTABLE	7	2,0794

Sumber: diolah peneliti (2025)

Langkah ini dilakukan untuk seluruh data pada setiap fitur yang memenuhi kriteria *skewness* > 0,5. Hasil transformasi ini kemudian digunakan sebagai data masukan untuk pelatihan model *Random Forest*. Setelah transformasi, distribusi data menjadi lebih simetris, sehingga diharapkan dapat meningkatkan performa model prediksi *defect* perangkat lunak.

Tabel 3. 10 Hasil setelah *logarithmic scaling*

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	1,0986	0,6931	0	2,0794	0
2.	0	0,6931	0	2,8332	0
3.	0	0,6931	0	0	0
4.	1,3862	0,6931	0	2,3025	0
5.	0,6931	0,6931	0	2,9444	1
6.	0	0,6931	0	2,4849	0
7.	0,6931	2,3978	0,6931	4,2766	1

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
8.	0,6931	2,5649	0,6931	4,3694	1
9.	0	0,6931	0	2,5649	0
10.	0	0,6931	0	3,3672	1
11.	0,6931	0,6931	0	2,7080	0
12.	1,3862	1,7917	0	3,4339	1
13.	1,0986	1,7917	0	2,7725	0
14.	1,0986	0,6931	0	2,6390	0
15.	2,0794	2,8903	1,6094	3,9318	0
16.	0	0,6931	0	0,6931	0
17.	1,3862	2,0794	0	3,0445	1
18.	0,6931	0,6931	0	1,6094	0
19.	0,6931	0,6931	0	1,6094	0
20.	0	0,6931	0	0,6931	0

Sumber: diolah peneliti (2025)

3.3.4 SMOTE

Setelah dilakukan transformasi *logarithmic scaling* terhadap fitur numerik, tahap selanjutnya dalam proses *preprocessing* data adalah penyeimbangan kelas. Salah satu metode yang digunakan dalam penelitian ini adalah SMOTE (*Synthetic Minority Over-sampling Technique*), yaitu teknik yang menghasilkan data sintetis baru untuk kelas minoritas guna menyamakan jumlahnya dengan kelas mayoritas.

Berbeda dengan *random oversampling* yang hanya menduplikasi sampel minoritas secara acak, SMOTE menciptakan sampel baru dengan melakukan interpolasi linier antara setiap sampel minoritas dan beberapa tetangga terdekatnya (biasanya ditentukan menggunakan algoritma *k-nearest neighbors*). Teknik ini memungkinkan penambahan variasi pada data sintetis, sehingga dapat mengurangi risiko *overfitting* yang sering terjadi akibat pengulangan data yang sama (Chawla et al., 2002).

Menurut Jiang (2024), SMOTE dirancang untuk memperkaya representasi kelas minoritas secara lebih realistis dibanding metode duplikasi sederhana. Proses pembentukan data sintetis SMOTE dijelaskan pada persamaan 3.3

$$x_{\text{new}} = x_i + \text{rand}(0,1) \times (x_{ij} - x_i) \quad (3.3)$$

Keterangan:
x_i : sampel dari kelas minoritas,
x_{ij} : salah satu tetangga terdekat dari x_i,
$rand(0,1)$: jumlah target sampel kelas minoritas setelah *oversampling*,
x_{new} : sampel sintetis yang dihasilkan.

Contoh perhitungan manual SMOTE, menggunakan data yang diberikan melalui transformasi *logarithmic scaling* pada Tabel 3.10. Fokus diberikan pada data minoritas ke-5 dengan label *Defective = 1*, dan dilakukan pencarian jarak *Euclidean* terhadap sesama data minoritas lainnya, yaitu data ke-7, ke-8, dan ke-10. Data-data tersebut disajikan pada Tabel 3.11.

Tabel 3. 11 Contoh data minoritas

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
5.	0,6931	0,6931	0	2,9444	1
7.	0,6931	2,3978	0,6931	4,2766	1
8.	0,6931	2,5649	0,6931	4,3694	1
10.	0	0,6931	0	3,3672	1

Sumber: diolah peneliti (2025)

1. Menghitung jarak *euclidean*

$$d(x_5, x_j) = \sqrt{\sum_{k=1}^{n} (x_5 - x_j)^2} \quad (3.4)$$

Keterangan :
x_5 : vektor fitur dari sampel minoritas ke-5,
x_j : vektor fitur dari salah satu tetangga minoritas terdekat dari x_5,
d : jarak *Euclidean*

Jarak data ke-5 dan data ke-7

$$d(x_5, x_7) =$$

$$\sqrt{(0.6931 - 0.6931)^2 + (0.6931 - 2.3979)^2 + (0 - 0.6931)^2 + (2.9444 - 4.2767)^2}$$

$$= \sqrt{0 + 2.9120 + 0.4804 + 1.7701}$$

$$= \sqrt{5.1625} \approx 2.272$$

Jarak data ke-5 dan data ke-8

$$d(x_5, x_8) =$$

$$\sqrt{(0.6931 - 0.6931)^2 + (0.6931 - 2.5649)^2 + (0 - 0.6931)^2 + (2.9444 - 4.3694)^2}$$

$$= \sqrt{0 + 3.5058 + 0.4804 + 2.0301}$$

$$= \sqrt{6.0163} \approx 2.453$$

Jarak data ke-5 dan data ke-10

$$d(x_5, x_{10}) =$$

$$\sqrt{(0.6931 - 0)^2 + (0.6931 - 0.6931)^2 + (0 - 0)^2 + (2.9444 - 3.3673)^2}\backslash$$

$$= \sqrt{0.4804 + 0 + 0 + 0.1784}$$

$$= \sqrt{0.6588} \approx 0.812$$

2. Buat data sintetis

Interpolasi dilakukan antara data ke-5 dan data ke-10 menggunakan parameter $\lambda = 0.5$ dengan persamaan 3.3. Komponen-komponen vektor hasil interpolasi:

$\text{synthetic}[0] = 0.6931 + 0.5 \cdot (0 - 0.6931) = 0.6931 - 0.3466 = 0.3465$

$\text{synthetic}[1] = 0.6931 + 0.5 \cdot (0.6931 - 0.6931) = 0.6931 + 0 = 0.6931$

$\text{synthetic}[2] = 0 + 0.5 \cdot (0 - 0) = 0$

$\text{synthetic}[3] = 2.9444 + 0.5 \cdot (3.3673 - 2.9444) = 2.9444 + 0.21 = 3.1558$

Masing-masing komponen hasil interpolasi, yaitu $\text{synthetic}[0]$ hingga $\text{synthetic}[3]$, merepresentasikan nilai fitur yang dihasilkan dari proses interpolasi

linier antara fitur-fitur yang sesuai pada data ke-5 dan data ke-10. Secara berurutan, synthetic[0] merupakan hasil interpolasi pada fitur LOC_BLANK, synthetic[1] mewakili fitur BRANCH_COUNT, synthetic[2]} adalah fitur LOC_COMMENTS, synthetic[3] merupakan nilai interpolasi untuk fitur LOC_EXECUTABLE.

Nilai-nilai ini digunakan untuk membentuk satu sampel sintetis baru yang memiliki karakteristik serupa dengan sampel minoritas asli, tetapi dengan sedikit variasi. Dengan cara ini, SMOTE membantu memperkaya representasi data minoritas tanpa melakukan duplikasi langsung.

Tabel 3. 12 Contoh *dataset* setelah diseimbangkan

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	1,0986	0,6931	0	2,0794	0
2.	0	0,6931	0	2,8332	0
3.	0	0,6931	0	0	0
4.	1,3862	0,6931	0	2,3025	0
5.	0	0,6931	0	2,4849	0
6.	0	0,6931	0	2,5649	0
7.	0,6931	0,6931	0	2,7080	0
8.	1,0986	1,7917	0	2,7725	0
9.	1,0986	0,6931	0	2,6390	0
10.	2,0794	2,8903	1,6094	3,9318	0
11.	0	0,6931	0	0,6931	0
12.	0,6931	0,6931	0	1,6094	0
13.	0,6931	0,6931	0	1,6094	0
14.	0	0,6931	0	0,6931	0
15.	0,6931	0,6931	0	2,9444	1
16.	0,6931	2,3978	0,6931	4,2766	1
17.	0,6931	2,5649	0,6931	4,3694	1
18.	0	0,6931	0	3,3672	1
19.	1,3862	1,7917	0	3,4339	1
20.	1,3862	2,0794	0	3,0445	1
21.	0,6931	2,4814	0,6931	4,3230	1
22.	1,3863	1,9356	0	3,2392	1
23.	0,3465	0,6931	0	3,1558	1
24.	1,0397	1,2424	0	3,1892	1
25.	1,0397	2,0948	0,34655	3,8553	1
26.	1,0397	1,3862	0	2,9944	1
27.	1,0397	2,1783	0,34655	3,9017	1
28.	1,0397	2,2386	0,34655	3,6606	1

Sumber: diolah peneliti (2025)

3.4 *Random Forest*

Random Forest adalah salah satu algoritma pembelajaran mesin yang digunakan untuk klasifikasi dan regresi. Algoritma ini merupakan jenis *ensemble learning* yang bekerja dengan menggabungkan banyak *decision tree* untuk meningkatkan akurasi prediksi. Setiap *decision tree* dihasilkan dari *subset* data yang berbeda (menggunakan teknik *bootstrap sampling*) dan dihasilkan dengan memilih fitur secara acak untuk memecah simpul-simpul dalam pohon tersebut.

Keunggulan utama *Random Forest* adalah kemampuannya untuk mengatasi masalah *overfitting* yang sering terjadi pada *decision tree* tunggal, serta memberikan prediksi yang lebih stabil dan akurat. Dalam *Random Forest,* prediksi akhir dibuat berdasarkan hasil *voting* mayoritas dari masing-masing *decision tree* (untuk klasifikasi) atau rata-rata (untuk regresi).

Contoh melakukan perhitungan manual *Random Forest* pada *dataset* tabel 3.12 dengan empat fitur (LOC_BLANK, BRANCH_COUNT, LOC_COMMENTS, LOC_EXECUTABLE) dan *defects* (kelas 0 atau 1). Berikut langkah-langkahnya :

a. Membagi data (*Bootstrap*)

Ini adalah langkah pertama di mana setiap pohon dalam *Random Forest* dilatih pada *dataset* yang berbeda. Di sini penulis menggunakan tiga estimator (n_estimator = 3) yang membagi *dataset* asli menjadi beberapa *subset* secara acak, sebagai berikut:

1. *Bootstrap sampling* 1

Tabel 3. 13 *Bootstrap sampling* 1

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	0	0,6931	0	2,5649	0
2.	0,6931	0,6931	0	1,6094	0
3.	0,6931	0,6931	0	1,6094	0
4.	1,0986	0,6931	0	2,639	0
5.	2,0794	2,8903	1,6094	3,9318	0
6.	0,6931	0,6931	0	1,6094	0
7.	0	0,6931	0	2,5649	0
8.	0,6931	2,3978	0,6931	4,2766	1
9.	1,0986	0,6931	0	2,0794	0
10.	0,6931	2,5649	0,6931	4,3694	1
11.	0	0,6931	0	2,8332	0
12.	0,6931	0,6931	0	1,6094	0
13.	1,0986	1,7917	0	2,7725	0
14.	0	0,6931	0	0,6931	0
15.	0,6931	0,6931	0	2,708	0
16.	1,0397	1,3862	0	2,9944	1
17.	1,3862	1,7917	0	3,4339	1
18.	0,6931	2,4814	0,6931	4,323	1
19.	0	0,6931	0	2,5649	0
20.	1,3862	1,7917	0	3,4339	1
21.	0,6931	2,4814	0,6931	4,323	1
22.	0,6931	0,6931	0	1,6094	0
23.	0	0,6931	0	0,6931	0
24.	0,6931	0,6931	0	2,9444	1
25.	1,3862	1,7917	0	3,4339	1
26.	0	0,6931	0	2,4849	0
27.	1,0397	1,2424	0	3,1892	1
28.	1,0397	1,2424	0	3,1892	1

Sumber: dig*enerate* penulis dari hasil program *python* (2025)

2. *Bootstrap sampling* 2

Tabel 3. 14 *Bootstrap sampling* 2

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	1,0986	0,6931	0	2,639	0
2.	0,6931	2,3978	0,6931	4,2766	1
3.	0	0,6931	0	0,6931	0
4.	1,0986	0,6931	0	2,639	0
5.	0,3465	0,6931	0	3,1558	1
6.	0,6931	0,6931	0	1,6094	0
7.	1,3862	1,7917	0	3,4339	1
8.	0,6931	0,6931	0	1,6094	0
9.	1,0986	0,6931	0	2,639	0
10.	1,0986	1,7917	0	2,7725	0
11.	0	0,6931	0	0	0

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
12.	0	0,6931	0	3,3672	1
13.	0,6931	0,6931	0	1,6094	0
14.	1,3863	1,9356	0	3,2392	1
15.	0,6931	2,3978	0,6931	4,2766	1
16.	1,0397	2,1783	0,34655	3,9017	1
17.	0,6931	2,4814	0,6931	4,323	1
18.	0,6931	2,4814	0,6931	4,323	1
19.	0	0,6931	0	2,5649	0
20.	1,0986	1,7917	0	2,7725	0
21.	1,3862	0,6931	0	2,3025	0
22.	0,6931	0,6931	0	2,708	0
23.	0	0,6931	0	2,4849	0
24.	0	0,6931	0	0,6931	0
25.	0,6931	0,6931	0	1,6094	0
26.	1,3862	2,0794	0	3,0445	1
27.	1,0986	1,7917	0	2,7725	0
28.	0,6931	0,6931	0	2,708	0

Sumber: di*generate* penulis dari hasil program *python* (2025)

3. *Bootstrap sampling* 3

Tabel 3. 15 *Bootstrap Sampling* 3

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
1.	0	0,6931	0	0,6931	0
2.	1,0397	2,0948	0,34655	3,8553	1
3.	1,0397	1,3862	0	2,9944	1
4.	1,3862	0,6931	0	2,3025	0
5.	1,0397	2,0948	0,34655	3,8553	1
6.	1,0986	0,6931	0	2,639	0
7.	1,0986	0,6931	0	2,0794	0
8.	1,3863	1,9356	0	3,2392	1
9.	1,3862	2,0794	0	3,0445	1
10.	0	0,6931	0	0,6931	0
11.	0,6931	0,6931	0	1,6094	0
12.	1,0397	1,3862	0	2,9944	1
13.	2,0794	2,8903	1,6094	3,9318	0
14.	0	0,6931	0	0,6931	0
15.	1,3863	1,9356	0	3,2392	1
16.	1,0397	1,2424	0	3,1892	1
17.	0,6931	0,6931	0	2,708	0
18.	1,0986	0,6931	0	2,0794	0
19.	0,6931	2,4814	0,6931	4,323	1
20.	0,6931	0,6931	0	1,6094	0
21.	1,0986	1,7917	0	2,7725	0
22.	0,6931	0,6931	0	2,9444	1
23.	1,0397	2,0948	0,34655	3,8553	1
24.	1,0397	2,1783	0,34655	3,9017	1
25.	0	0,6931	0	3,3672	1
26.	1,0397	2,1783	0,34655	3,9017	1

No	LOC_ BLANK	BRANCH_ COUNT	LOC_ COMMENTS	LOC_ EXECUTABLE	Defective
27.	0,3465	0,6931	0	3,1558	1
28.	0	0,6931	0	0	0

Sumber: di*generate* penulis dari hasil program *python* (2025)

b. Membentuk pohon keputusan (*decision tree*)

Langkah berikutnya dalam algoritma *Random Forest* adalah membentuk pohon keputusan (*decision tree*) dari setiap *bootstrap sample*. Pada tahap ini, proses dimulai dengan menentukan fitur terbaik yang akan digunakan sebagai akar pohon (*root node*). Pemilihan fitur terbaik dilakukan dengan menghitung nilai *Gini Impurity* dan *Gini Split* dari setiap fitur yang tersedia. Fitur dengan nilai *Gini Split* paling rendah akan dipilih sebagai *node* karena menghasilkan pemisahan kelas (*split*) yang paling "bersih" atau homogen.

Gini Impurity digunakan untuk mengukur tingkat ketidakmurnian (*impurity*) suatu *node*. Semakin rendah nilai *Gini Impurity*, semakin homogen data pada *node* tersebut. Adapun *Gini Split* mengukur *impurity* total dari dua *subset* data yang dihasilkan setelah pemisahan berdasarkan *threshold* pada suatu fitur.

Pada bagian ini, proses perhitungan *Gini Impurity* dan *Gini Split* akan dijelaskan secara manual menggunakan fitur LOC_EXECUTABLE dari *bootstrap sampling 1*. Nilai *threshold* ditentukan berdasarkan salah satu nilai tengah dari data sebagai batas untuk membagi data menjadi dua *subset*, yaitu *subset* kiri (nilai fitur ≤ *threshold*) dan *subset* kanan (nilai fitur > *threshold*). Setelah pembagian data, masing-masing *subset* dihitung nilai *Gini Impurity*-nya, lalu dihitung nilai *Gini Split* totalnya.

Rumus *Gini Impurity*

$$Gini\ Impurity = 1 - \sum_{i=0}^{c-1}(p_i^2) \quad (3.5)$$

Keterangan:
p_i : Probabilitas *Node*
c : Jumlah kelas yang ada pada *node* tersebut.

Rumus *Gini Split* atau kriteria gini

$$Gini\ Split = \sum_{i=0}^{k-1}(\frac{n_i}{n})Gini\ Impurity \quad (3.6)$$

Keterangan:
k : Jumlah kelas yang ada pada *node* tersebut.
n_i : Total data pada suatu kelas
n : Total seluruh data

Contoh perhitungan pada fitur LOC_EXECUTABLE dengan *threshold* = 2,8888 yang ada di *bootstrap sample* 1.

1. Bagi data berdasarkan *threshold*:

   Subset kiri (≤ 2,7402):

   Nilai LOC_EXECUTABLE : 0.6931, 0.6931, 0.6931, 1.6094, 1.6094, 1.6094, 1.6094, 2.0794, 2.4849, 2.5649, 2.5649, 2.5649, 2.6390, 2.7080

   Total = 14 data

   Subset kanan (> 2,7402):

   Nilai LOC_EXECUTABLE : 2.7725, 2.8332, 2.9444, 2.9944, 3.1892, 3.1892, 3.4339, 3.4339, 3.4339, 3.9318, 4.2766, 4.3230, 4.3230, 4.3694

   Total = 14 data

2. Hitung *Gini Impurity* masing-masing *subset*

   Subset kiri (> 2,7402)

   Data *Defective* (kelas): 0 = 14 dan 1 = 0

   Menghitung *Gini* kiri dan kanan menggunakan persamaan 3.7

$$Gini_{\text{kiri}} = 1 - \left(\frac{14}{14}\right)^2 - \left(\frac{0}{14}\right)^2 = 1 - 1 - 0 = 0$$

   Subset kanan (> 2,7402)

   Data *Defective* (kelas) : 0 = 3 dan 1 = 11

$$Gini_{\text{kanan}} = 1 - \left(\frac{3}{14}\right)^2 - \left(\frac{11}{14}\right)^2 \approx 0.3367$$

3. Hitung *Gini split* masing-masing *subset*

   Menghitung *Gini* split menggunakan persamaan 3.8

$$Gini_{\text{total}} = \frac{14}{28} \times Gini_{\text{kiri}} + \frac{14}{28} \times Gini_{\text{kanan}}$$

$$= \frac{14}{28} \times 0 + \frac{14}{28} \times 0.3367$$

$$= 0 + 0.16835$$

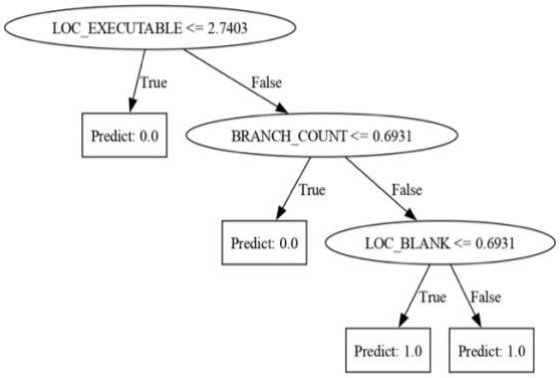
$$= 0.16835$$

Berikut hasil perhitungan *gini impurity* tiap *bootstrap sample* beserta *decision tree* yang terbentuk

Tabel 3. 16 *Gini impurity bootstrap sampling 1*

Fitur	*Threshold*	*Subset* Kiri		*Subset* Kanan		*Gini* Kiri	*Gini* Kanan	*Gini* Total
		0	1	0	1			
LOC_BLANK	0,6931	13	5	4	6	0,4012	0,48	0,4293
BRANCH_COUNT	0,6931	15	1	2	10	0,1171	0,2777	0,1860
LOC_COMMENTS	0	16	7	1	4	0,4234	0,32	0,4049
LOC_EXECUTABLE	2,7402	14	0	3	11	0	0,3367	0,1683

Sumber: diolah peneliti (2025)

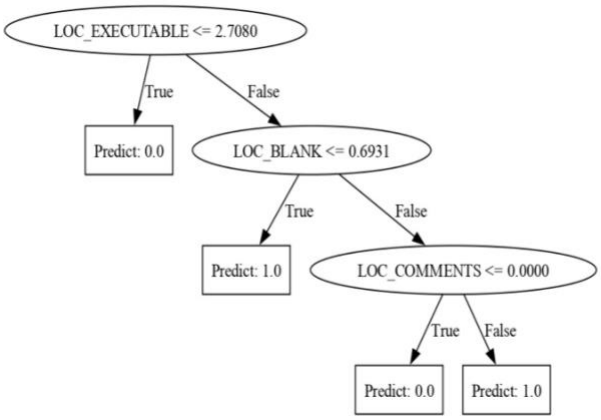


Gambar 3. 2 *Bootstrap sampling 1*
Sumber: diolah peneliti (2025)

Tabel 3. 17 *Gini impurity bootstrap sampling* 2

Fitur	*Threshold*	*Subset* Kiri		*Subset* Kanan		*Gini* Kiri	*Gini* Kanan	*Gini* Total
		0	1	0	1			
LOC_BLANK	0,6931	11	6	7	4	0,4567	0,4628	0,4591
BRANCH_COUNT	0,6931	15	2	3	8	0,2076	0,3966	0,2818
LOC_COMMENTS	0	18	5	0	5	0,3402	0	0,2795
LOC_EXECUTABLE	2,708	15	0	3	10	0	0,3550	0,1648

Sumber: diolah peneliti (2025)

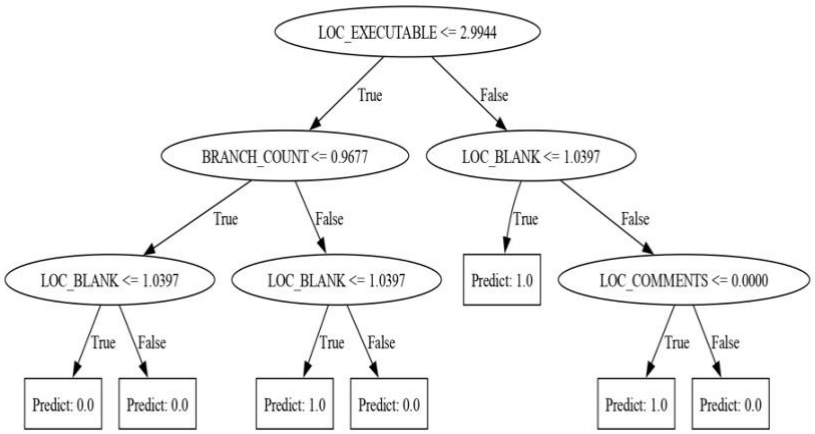


Gambar 3. 3 *Bootstrap sampling 2*
Sumber: diolah peneliti (2025)

Tabel 3. 18 *Gini impurity bootstrap sampling 3*

Fitur	*Threshold*	*Subset* Kiri		*Subset* Kanan		*Gini* Kiri	*Gini* Kanan	*Gini* Total
		0	1	0	1			
LOC_BLANK	1,0397	7	4	6	11	0,4628	0,4567	0,4591
BRANCH_COUNT	0,9677	11	3	2	12	0,3367	0,2448	0,2908
LOC_COMMENTS	0	12	9	1	6	0,4897	0,2448	0,4285
LOC_EXECUTABLE	2,9944	11	0	2	15	0	0,20761	0,1260

Sumber: diolah peneliti (2025)



Gambar 3. 4 *Bootstrap sampling* 3
Sumber: diolah peneliti (2025)

3.5 Hasil Prediksi

Setelah model *Random Forest* dibangun, tahap berikutnya adalah menghasilkan prediksi *defect* perangkat lunak pada *dataset* yang digunakan. Hasil dari prediksi ini bukan hanya berupa nilai akurasi model, tetapi juga informasi terkait dengan potensi *defect* pada bagian-bagian tertentu dari perangkat lunak. Model *Random Forest* memberikan *output* berupa klasifikasi *biner* pada setiap entitas perangkat lunak yang dianalisis, apakah berisiko memiliki *defect* atau tidak.

3.6 Evaluasi

Evaluasi dalam penelitian ini menggunakan *Confusion Matrix* dan *K-fold Cross Validation* sebagai metode untuk menilai kinerja model prediksi *defect* perangkat lunak.

3.6.1 Confusion Matrix

Confusion Matrix digunakan sebagai alat utama untuk mengukur performa model klasifikasi secara terperinci. Tabel ini memetakan hasil prediksi model ke dalam empat kategori, yaitu:

1. *True Positive* (TP) : Data positif yang diprediksi benar sebagai positif
2. *True Negative* (TN) : Data negatif yang diprediksi benar sebagai negatif
3. *False Positive* (FP) : Data negatif yang salah diprediksi sebagai positif
4. *False Negative* (FN) : Data positif yang salah diprediksi sebagai negatif

Dengan menggunakan *confusion matrix*, beberapa metrik evaluasi seperti *Accuracy* dan *F1-Score* dapat dihitung untuk mengevaluasi kinerja model secara menyeluruh. Namun, penting untuk dicatat bahwa hanya mengandalkan satu kali pembagian data uji dapat menghasilkan evaluasi yang kurang *robust*, karena performa model dapat bervariasi tergantung pada bagaimana data terbagi.

Contoh perhitungan manual *confusion matrix*, misalkan setelah model dijalankan, diperoleh hasil sebagai berikut untuk 117 data uji, dengan hasil sebagai berikut:

1. 50 data benar terklasifikasi sebagai *defect* (TP = 50)
2. 50 data benar terklasifikasi sebagai tidak *defect* (TN = 50)
3. 7 data seharusnya *defect* tetapi diprediksi tidak *defect* (FN = 7)

4. 10 data seharusnya tidak *defect* tetapi diprediksi *defect* (FP = 10)

Sehingga, *Confusion Matrix* yang terbentuk adalah:

Tabel 3. 19 Contoh *confusion matrix*

		Nilai Prediksi	
		Defect (1)	Tidak *Defect* (0)
Nilai Aktual	*Defect* (1)	50 (TP)	7 (FN)
	Tidak *Defect* (0)	10 (FP)	50 (TN)

Dari tabel ini, dapat dihitung metrik evaluasi sebagai berikut:

1. Akurasi (*Accuracy*)

Menggunakan persamaan 2.3

$$Accuracy = \frac{50 + 50}{50 + 50 + 10 + 7}$$
$$= \frac{100}{117} \approx 85{,}47\%$$

2. *F1-Score*

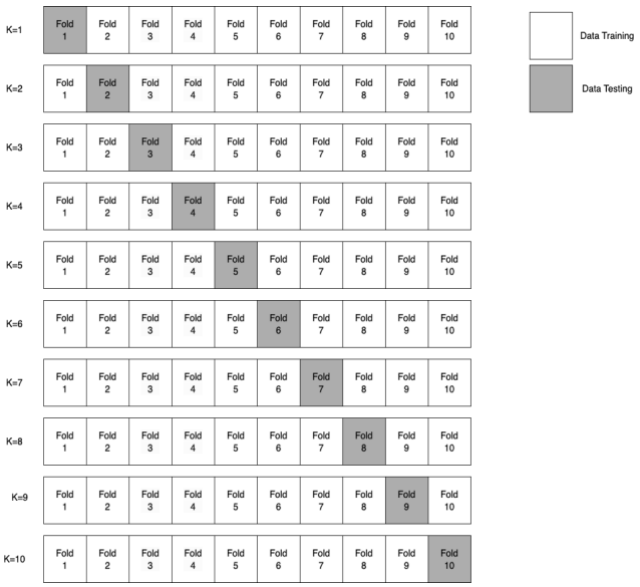
Menggunakan persamaan 2.4

$$F1 - Score = \frac{2 \times Presisi \times Recall}{Presisi + Recall}$$
$$= \frac{2 \times 0.8333 \times 0.8772}{0.8333 + 0.8772}$$
$$= \frac{2 \times 0.7221}{1.7105} \approx 85{,}43\%$$

3.6.2 *K-fold Cross Validation*

Untuk mengatasi pengujian yang dapat mengakibatkan evaluasi kurang *robust*, *K-fold Cross Validation* digunakan. Metode ini membagi *dataset* menjadi beberapa bagian (*fold*) yang memastikan bahwa model diuji pada berbagai *subset* data yang berbeda, sehingga menghasilkan evaluasi yang lebih komprehensif. Pada setiap iterasi, model dilatih pada sejumlah *fold* tertentu dan diuji pada *fold*

yang tersisa. Proses ini diulang sebanyak *k* kali, sehingga setiap bagian *dataset* digunakan sebagai data uji sebanyak satu kali.



Gambar 3. 5 *K-fold Cross Validation*

Dalam penelitian ini, metode *K-fold Cross Validation* diterapkan dengan nilai *k* yang disesuaikan dengan rasio pembagian data. Untuk rasio 90:10 digunakan *10-fold Cross Validation*, rasio 80:20 menggunakan *5-fold*, rasio 75:25 menggunakan *4-fold*, dan rasio 70:30 menggunakan *3-fold*. Penyesuaian ini dilakukan untuk menjaga proporsi data latih dan uji dalam setiap *fold*, terutama pada *dataset* dengan jumlah data yang terbatas, agar hasil evaluasi tetap representatif dan tidak memicu *overfitting*. Penggunaan *cross validation* ini memastikan bahwa model diuji pada berbagai skenario dan memberikan gambaran yang lebih menyeluruh mengenai kinerja model dalam memprediksi *defect* perangkat lunak. Studi sebelumnya juga menunjukkan efektivitas kombinasi *Random Forest* dan *K-fold Cross Validation* dalam berbagai domain.

Contoh perhitungan manual *K-Fold Cross Validation* (K=5). Misalkan kita memiliki 100 data dan ingin menggunakan 5-*Fold Cross Validation* (K=5), maka *dataset* dibagi menjadi 5 bagian masing-masing berisi 20 data.

Tabel 3. 20 Contoh *k-fold cross validation*

Fold	Data Latih (80 data)	Data Uji (20 data)	Akurasi
Fold 1	*Fold* 2, 3, 4, 5	*Fold* 1	78%
Fold 2	*Fold* 1, 3, 4, 5	*Fold* 2	80%
Fold 3	*Fold* 1, 2, 4, 5	*Fold* 3	82%
Fold 4	*Fold* 1, 2, 3, 5	*Fold* 4	79%
Fold 5	*Fold* 1, 2, 3, 4	*Fold* 5	81%

Sumber: diolah peneliti (2025)

Setelah semua 5 *fold* diuji, kita menghitung rata-rata akurasi:

$$\text{Akurasi rata} - \text{rata} = \frac{78\% + 80\% + 82\% + 79\% + 81\%}{5}$$

$$= \frac{400}{5} = 80\%$$

Jadi, berdasarkan *5-Fold Cross Validation*, model memiliki akurasi rata-rata sebesar 80%.

Untuk memastikan evaluasi model lebih komprehensif, penelitian ini juga mengombinasikan *Confusion Matrix* dengan *K-fold Cross Validation*. *Confusion Matrix* memberikan informasi detail mengenai akurasi prediksi di setiap kelas, memungkinkan analisis lebih mendalam terhadap distribusi kesalahan. Sementara itu, *K-fold Cross Validation* memastikan bahwa evaluasi dilakukan secara menyeluruh pada berbagai *subset* data, sehingga mengurangi risiko bias dan *overfitting*. Dengan pendekatan ini, diharapkan hasil prediksi *defect* perangkat lunak menjadi lebih andal dan dapat digunakan sebagai dasar pengambilan keputusan dalam pengembangan perangkat lunak.

3.7 Skenario Uji Coba

Dalam skenario pengujian ini, model *Random Forest* akan dievaluasi berdasarkan tiga pendekatan berbeda terhadap kondisi data sebelum proses pelatihan, yaitu:

1. Skenario *Baseline*

   Model dilatih menggunakan data asli tanpa proses penyeimbangan kelas maupun transformasi fitur. Pendekatan ini berfungsi sebagai pembanding awal (*baseline*) terhadap skenario lainnya.

2. Skenario SMOTE

   Data yang tidak seimbang diseimbangkan menggunakan teknik *Synthetic Minority Over-sampling Technique* (SMOTE), di mana sampel sintetis dari kelas minoritas dibuat untuk menyeimbangkan jumlah dengan kelas mayoritas. Tidak dilakukan transformasi pada fitur.

3. Skenario SMOTE dengan *Filtering*

   Sebelum proses penyeimbangan, fitur-fitur numerik yang memiliki distribusi tidak normal ditransformasi menggunakan *logarithmic scaling* untuk mengurangi *skewness*. Selanjutnya, data diseimbangkan dengan SMOTE. Setelah proses *oversampling*, dilakukan *filtering* terhadap sampel sintetis berdasarkan rasio tetangga dari kelas minoritas menggunakan algoritma *k-nearest neighbors*. Hanya sampel sintetis yang memiliki ≥25% tetangga dari kelas minoritas yang dipertahankan, guna mengurangi *noise* pada distribusi data.

Setiap pendekatan di atas akan diuji pada empat skenario pembagian data latih dan data uji yang telah ditetapkan, yaitu:

1. 90% data latih dan 10% data uji
2. 80% data latih dan 20% data uji
3. 75% data latih dan 25% data uji
4. 70% data latih dan 30% data uji

Hasil evaluasi dari masing-masing skenario akan dibandingkan untuk menilai pengaruh teknik penyeimbangan data dan transformasi fitur terhadap kinerja model, khususnya dalam meningkatkan kemampuan model dalam memprediksi *defect* perangkat lunak. Analisis ini akan membantu menentukan kombinasi pendekatan dan konfigurasi data yang menghasilkan performa terbaik.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Hasil Penyeimbangan Data

Pada penelitian ini, penyeimbangan data dilakukan menggunakan dua pendekatan, yaitu SMOTE dan SMOTE yang dikombinasikan dengan *filtering* berbasis kemiripan data sintetis terhadap data mayoritas. Kedua pendekatan ini bertujuan untuk meningkatkan representasi kelas minoritas (*defect*) agar model pembelajaran mesin tidak bias terhadap kelas mayoritas (*non-defect*).

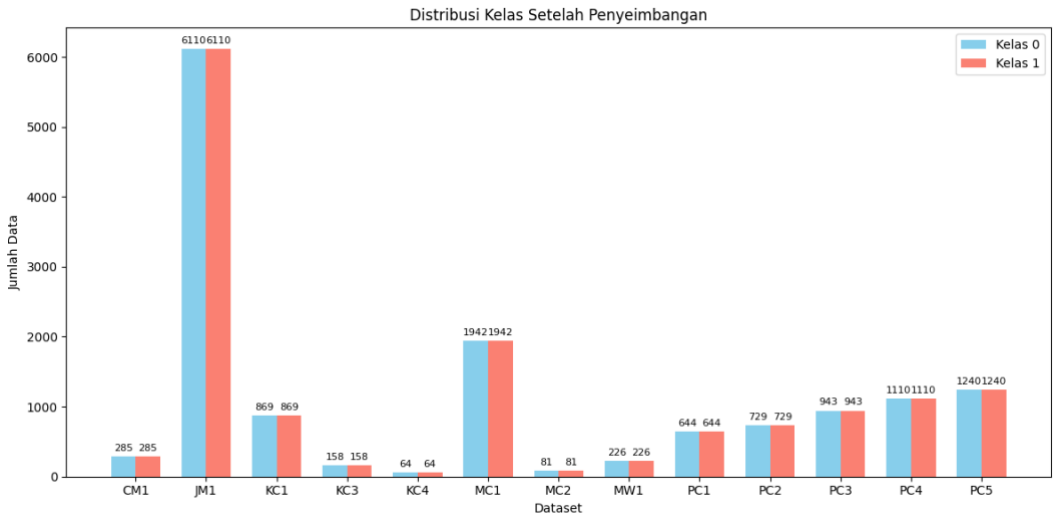
Penerapan SMOTE menghasilkan penambahan sampel sintetis pada kelas minoritas hingga jumlahnya sama dengan kelas mayoritas. Pendekatan ini sangat penting, terutama untuk *dataset* berukuran kecil, agar karakteristik data asli tetap tercermin dalam proses pelatihan model. Tabel berikut menunjukkan distribusi jumlah sampel sebelum dan sesudah penyeimbangan menggunakan SMOTE:

Tabel 4. 1 Distribusi data sebelum dan sesudah penyeimbangan

No	Dataset	Jumlah Fitur	Jumlah data sebelum penyeimbangan		Jumlah data setelah penyeimbangan	
			0	1	0	1
1.	CM1	37	285	42	285	285
2.	JM1	21	6110	1672	6110	6110
3.	KC1	21	869	314	869	869
4.	KC3	39	158	36	158	158
5.	KC4	40	64	61	64	64
6.	MC1	38	1942	46	1942	1942
7.	MC2	39	81	44	81	81
8.	MW1	37	226	27	226	226
9.	PC1	37	644	61	644	644
10.	PC2	36	729	16	729	729
11.	PC3	37	943	134	943	943
12.	PC4	31	1110	177	1110	1110
13.	PC5	38	1240	471	1240	1240

Sumber: diolah peneliti (2025)

Untuk memberikan gambaran visual mengenai dampak dari proses penyeimbangan ini disajikan distribusi kelas hasil SMOTE pada masing-masing *dataset*.



Gambar 4. 1 Distribusi kelas setelah penyeimbangan

Meskipun SMOTE secara efektif meningkatkan proporsi kelas minoritas, terdapat potensi bahwa sebagian data sintetis yang dihasilkan memiliki karakteristik yang terlalu mirip dengan data kelas mayoritas. Oleh karena itu, dilakukan tahap tambahan berupa *filtering*, yaitu penyaringan terhadap data sintetis berdasarkan kedekatannya dengan sampel dari kelas mayoritas, menggunakan pendekatan berbasis kemiripan fitur.

Filtering ini bertujuan untuk mempertahankan keberagaman data kelas minoritas hasil sintesis dan menghindari bias model akibat data sintetis yang terlalu serupa dengan kelas mayoritas. Hasil akhirnya adalah jumlah data minoritas tetap bertambah, namun sebagian data sintetis yang terlalu mirip telah dihilangkan.

Tabel 4. 2 Distribusi data sebelum dan sesudah penyeimbangan dengan *filtering*

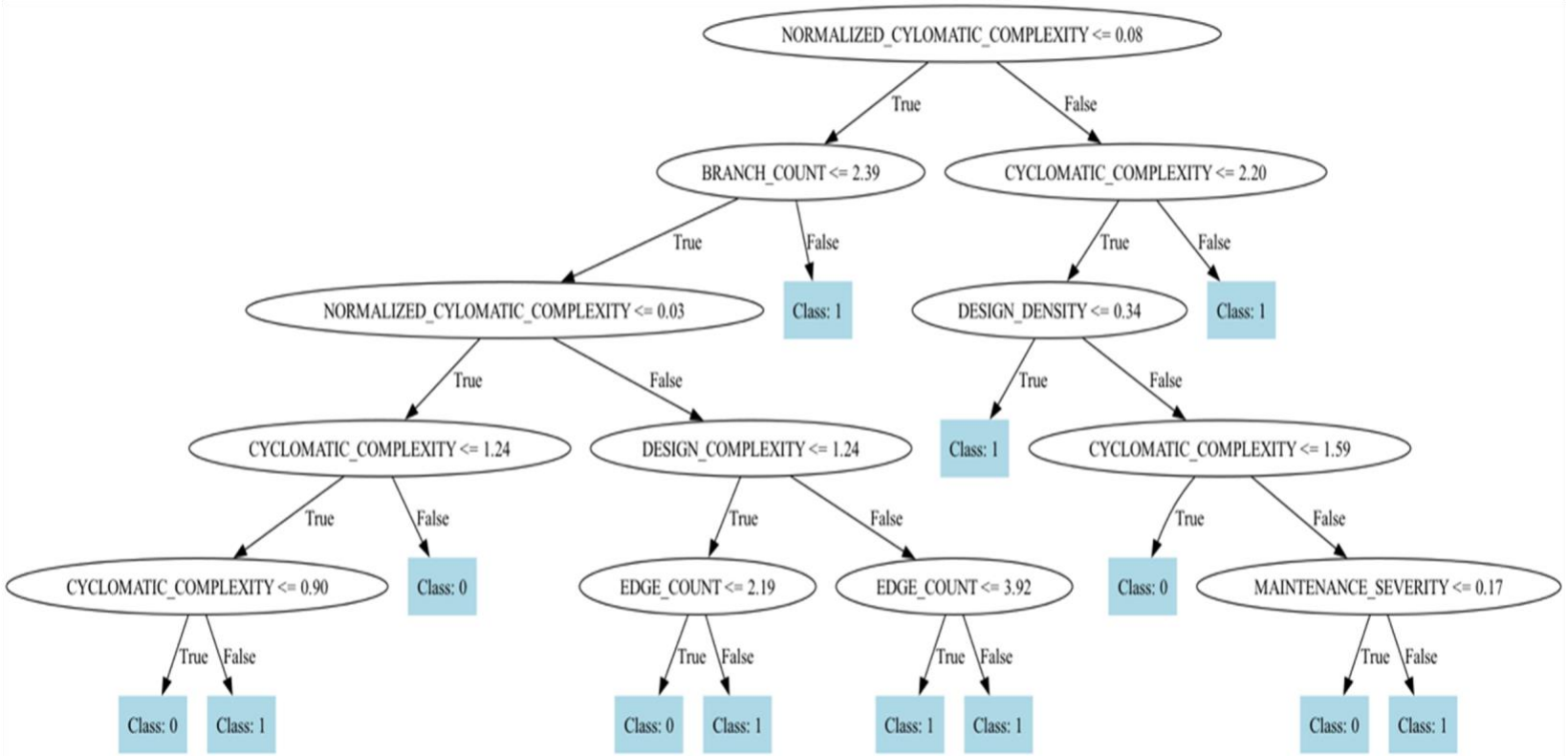
No	Dataset	Jumlah Fitur	Jumlah data sebelum penyeimbangan		Jumlah data setelah penyeimbangan dengan *filtering*	
			0	1	0	1
1.	CM1	37	285	42	285	283
2.	JM1	21	6110	1672	6110	6052
3.	KC1	21	869	314	869	856
4.	KC3	39	158	36	158	156
5.	KC4	40	64	61	64	64
6.	MC1	38	1942	46	1942	1928
7.	MC2	39	81	44	81	81
8.	MW1	37	226	27	226	221
9.	PC1	37	644	61	644	643
10.	PC2	36	729	16	729	729
11.	PC3	37	943	134	943	942
12.	PC4	31	1110	177	1110	1109
13.	PC5	38	1240	471	1240	1229

Sumber: diolah peneliti (2025)

Dengan distribusi kelas yang telah diseimbangkan melalui dua pendekaran tersebut, langkah selanjutnya adalah melatih model *Random Forest* untuk mengevaluasi pengaruh penyeimbangan ini terhadap performa prediksi *defect* perangkat lunak.

4.2 Hasil *Training Random Forest*

Pada tahap pelatihan, algoritma *Random Forest* membentuk sejumlah pohon keputusan, masing-masing dibangun dari *subset* acak data latih (*bootstrap sample*). Setiap pohon memisahkan data berdasarkan fitur-fitur yang dipilih secara rekursif, dengan tujuan meminimalkan nilai *gini impurity* di setiap simpul. Gambar 4.2 menampilkan salah satu pohon keputusan yang dihasilkan dari proses pelatihan, digunakan sebagai representasi untuk menggambarkan bagaimana model menentukan pemisahan data secara bertahap berdasarkan nilai *gini*



Gambar 4. 2 Visualisasi salah satu pohon pada *dataset* KC4
Sumber: di*generate* penulis dari hasil program *python* (2025)

Pohon tersebut dimulai dari simpul akar yang memilih fitur NORMALIZED_CYCLOMATIC_COMPLEXITY <= 0,08, karena pemisahan ini menghasilkan penurunan *gini* paling besar pada langkah pertama. Jika suatu data memiliki nilai fitur tersebut lebih kecil atau sama dengan 0,08, maka data tersebut diarahkan ke cabang kiri. Sebaliknya, jika lebih besar, diarahkan ke cabang kanan.

Contohnya, dari simpul akar, cabang kiri menuju simpul dengan pemisahan BRANCH_COUNT <= 2,39. Artinya, hanya data dengan NORMALIZED_CYCLOMATIC_COMPLEXITY <= 0,08 yang akan diperiksa lebih lanjut menggunakan BRANCH_COUNT. Pada setiap langkah, proses pembelahan dilakukan berdasarkan fitur dan *threshold* yang mengoptimalkan pengurangan *impurity*.

Node daun (*leaf node*) pada pohon menunjukkan hasil akhir klasifikasi. Sebagai contoh, sebuah simpul daun yang menampilkan *Class*: 0 menunjukkan bahwa data yang mencapai *node* tersebut diprediksi sebagai *non-defect*, sedangkan *Class*: 1 menunjukkan *defect*.

```python
# Kelas Random Forest
class RandomForest:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, n_estimators=10, max_depth=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = []

    Tabnine | Edit | Test | Explain | Document
    def fit(self, X, y):
        for _ in range(self.n_estimators):
            indices = np.random.choice(len(X), len(X), replace=True)
            X_sample, y_sample = X.iloc[indices], y.iloc[indices]
            tree = DecisionTree(max_depth=self.max_depth)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    Tabnine | Edit | Test | Explain | Document
    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        return np.squeeze(np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=0, arr=tree_preds))

    # Tambahkan get_params agar kompatibel dengan scikit-learn
    Tabnine | Edit | Test | Explain | Document
    def get_params(self, deep=True):
        return {"n_estimators": self.n_estimators, "max_depth": self.max_depth}
✓ 0.0s
```

```python
# Membuat dan melatih model Random Forest
rf_manual = RandomForest(n_estimators=100, max_depth=5)
rf_manual.fit(X_train, y_train)
✓ 3.9s
```

Gambar 4. 3 Kode implementasi *Random Forest*

Gambar 4.3 menunjukkan potongan kode implementasi dari algoritma *Random Forest* yang digunakan dalam penelitian ini. Kelas *Random Forest* dirancang untuk mereplikasi prinsip *Random Forest,* yaitu membentuk sejumlah pohon keputusan (*decision tree*) secara independent dari subset acak data latih yang diperoleh melalui teknik *bootstrap sampling.* Prediksi akhir dilakukan melalui mekanisme *majority voting* terhadap hasil prediksi dari setiap pohon.

Selain fungsi *fit* dan *predict*, implementasi ini juga menyertakan metode *get_params,* yang bertujuan agar model dapat digunakan secara langsung dalam proses evaluasi seperti *K-Fold Cross Validation* menggunakan *library scikit-learn.* Sehingga model dapat divalidasi secara sistematis untuk mengukur kestablan dan kinerjanya terhadap data yang tidak terlihat selama pelatihan.

4.2 Hasil *Testing Random Forest*

Setelah proses pelatihan model selesai dilakukan, langkah selanjutnya adalah menguji kinerja model *Random Forest* terhadap data pengujian (*testing set*) yang telah dipisahkan sebelumnya. Pada tahap ini, model yang telah dilatih digunakan untuk memprediksi kelas (*defect* atau *non-defect*) dari masing-masing *instance* dalam data uji, dan hasil prediksi dibandingkan dengan nilai aktual untuk menilai akurasi dan efektivitas model.

Tabel 4.2 menampilkan sebagian hasil prediksi model *Random Forest* terhadap data uji. Beberapa fitur input yang digunakan antara lain BRANCH_COUNT, CALL_PAIRS, CYCLOMATIC_COMPLEXITY, dan lain sebagainya. Kolom *Actual* menunjukkan label asli (0 = non-defect, 1 = defect), sedangkan kolom *Predicted* menunjukkan hasil prediksi. Karena banyaknya fitur dan nama yang panjang, tabel dalam dokumen utama disederhanakan untuk menjaga keterbacaan, sementara versi lengkapnya tersedia di lampiran.

Tabel 4. 3 Contoh hasil *testing* pada *dataset* KC4

BC	CP	CC	CD	DC	…..	NC	NL	Actual	Predicted	Kesesuaian
1.0	1.0	1.0	0.0	1.0	…	6.0	79.0	0	0	Sesuai
1.0	1.0	1.0	0.0	1.0	…	9.0	405.0	0	0	Tidak Sesuai
1.0	3.0	1.0	0.0	1.0	…	23.0	34.0	1	1	Sesuai
1.0	1.0	1.0	0.0	1.0	…	10.0	24.0	0	0	Sesuai
1.0	3.0	1.0	0.0	1.0	…	10.0	22.0	0	0	Sesuai
23.0	10.0	12.0	0.0	12.0	…	111.0	124.0	1	1	Sesuai
1.0	1.0	1.0	0.0	1.0	…	6.0	142.0	0	0	Sesuai
51.0	11.0	26.0	0.0	20.0	…	209.0	228.0	1	1	Sesuai
7.0	2.0	4.0	0.0	1.0	…	33.0	34.0	0	1	Tidak Sesuai
15.0	14.0	8.0	0.0	7.0	…	113.0	138.0	1	1	Sesuai
9.0	2.0	5.0	0.0	2.0	…	26.0	41.0	1	1	Sesuai
127.0	17.0	64.0	0.0	42.0	…	583.0	634.0	1	1	Sesuai
1.0	1.0	1.0	0.0	1.0	…	8.0	2064.0	0	0	Sesuai
1.0	1.0	1.0	0.0	1.0	…	6.0	79.0	0	0	Sesuai
1.0	1.0	1.0	0.0	1.0	…	9.0	405.0	0	0	Sesuai

Sumber: dig*enerate* penulis dari hasil program *python* (2025)

Berikut penjelasan singkatan kolom pada tabel:
BC : BRANCH_COUNT
CP : CALL_PAIRS
CC : CYCLOMATIC_COMPLEXITY
CD : CYCLOMATIC_DENSITY
DC : DESIGN_COMPLEXITY
… : Kolom fitur lain yang disingkat untuk menjaga keterbacaan
NC : NODE_COUNT
NL : NUMBER_OF_LINES
Actual : Label sebenarnya (0 = *non-defect*, 1 = *defect*)
Predicted : Hasil prediksi model *Random Forest*

Berdasarkan hasil tersebut, secara umum model *Random Forest* mampu mengklasifikasikan sebagian besar data dengan tepat. Namun, terdapat pula beberapa kasus misklasifikasi. Pada salah satu *instance* dengan karakteristik BRANCH_COUNT = 6.99, MAINTENANCE_SEVERITY = 0.284, dan LOC_TOTAL = 34.0, model memprediksi sebagai *defect* (1), padahal nilai aktualnya adalah *non-defect* (0). Kesalahan seperti ini dapat terjadi karena kemiripan nilai fitur dengan *instance defect* lainnya atau karena keterbatasan model dalam membedakan pola yang sangat mirip antar kelas.

Untuk mengevaluasi performa keseluruhan model, dilakukan pengukuran metrik evaluasi yang dijelaskan lebih lanjut pada subbab berikutnya, seperti akurasi dan *f1-score*. Hasil pengujian ini memberikan gambaran awal tentang kemampuan generalisasi model terhadap data baru yang belum pernah dilihat sebelumnya.

4.3 Evaluasi Skenario *Baseline*

Pada skenario ini, model *Random Forest* dilatih menggunakan data dalam kondisi aslinya, tanpa penerapan teknik penyeimbangan data. Artinya, distribusi kelas pada *dataset* tetap tidak seimbang sebagaimana kondisi awal masing-masing

dataset. Tujuan dari skenario *baseline* ini adalah untuk mengevaluasi kinerja model ketika dihadapkan pada ketidakseimbangan kelas yang umum ditemukan dalam data dunia nyata, khususnya pada domain prediksi *defect* perangkat lunak.

Skenario ini berfungsi sebagai tolok ukur (*benchmark*) untuk menilai efektivitas dari metode penyeimbangan data yang diusulkan, yaitu SMOTE dan SMOTE dengan *filtering*. Dengan membandingkan hasil performa model pada scenario ini terhadap dua skenario lainnya, dapat dianalisis sejauh mana pengaruh distribusi kelas terhadap metrik evaluasi seperti akurasi dan *F1-score.*

Untuk menjaga validitas dan generalisasi hasil, setiap *dataset* diuji dengan beberapa variasi rasio pembagian data latih dan data uji, yaitu 90:10, 80:20, 75:25, dan 70:30. Hasil evaluasi pada masing-masing rasio tersebut akan disajikan dan dianalisis lebih lanjut pada subbab berikutnya

4.3.1 Hasil Uji Coba dengan Pembagian 90:10

Pada pengujian ini, model *Random Forest* dilatih menggunakan data tanpa penyeimbangan (tanpa SMOTE), dengan distribusi kelas dibiarkan dalam kondisi aslinya. Data dibagi dengan rasio 90% untuk pelatihan dan 10% untuk pengujian, agar model dapat mempelajari pola dari jumlah data latih yang besar meskipun dalam kondisi ketidakseimbangan. Evaluasi dilakukan menggunakan metrik akurasi dan *F1-score* untuk menilai kemampuan model, khususnya dalam mengenali kelas minoritas (*defect*). Tabel berikut menyajikan hasil pengujian terhadap 13 *dataset*.

Tabel 4. 4 Hasil uji coba skenario 1 dengan rasio 90:10

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8182	0,0000
2.	JM1	0,8036	0,1257
3.	KC1	0,7311	0,1579
4.	KC3	0,6500	0,0000
5.	KC4	0,6923	0,6667
6.	MC1	0,9749	0,0000
7.	MC2	08462	0,5000
8.	MW1	0,8846	0,0000
9.	PC1	0,9296	0,2857
10.	PC2	0,9733	0,0000
11.	PC3	0,8519	0,0000
12.	PC4	0,8527	0,0000
13.	PC5	0,7035	0,2154

Sumber: di*generate* penulis dari hasil program *python* (2025)

Berdasarkan hasil tersebut, dapat diamati bahwa model menghasilkan akurasi yang tinggi pada sebagian besar *dataset*. MC1 mencapai akurasi sebesar 97,49%, PC2 sebesar 97,33%, dan PC1 sebesar 92,96%. Namun, tingginya akurasi ini tidak selalu mencerminkan kemampuan model dalam mendeteksi kelas *defect*. Hal ini tercermin dari *F1-score* yang sangat rendah, bahkan 0, pada banyak *dataset*. Sebagai contoh, CM1, KC3, MC1, MW1, PC2, PC3, dan PC4 semuanya memiliki *F1-score* sebesar 0. Hal ini menunjukkan bahwa model gagal mengidentifikasi *instance* kelas minoritas, dan cenderung hanya memprediksi kelas mayoritas.

Di sisi lain, terdapat beberapa *dataset* yang menunjukkan performa yang relatif lebih seimbang. *Dataset* KC4 menghasilkan *F1-score* sebesar 0,6667, menunjukkan kemampuan model yang baik dalam mengenali kelas *defect* meskipun data tidak seimbang. *Dataset* MC2 juga memperlihatkan performa yang cukup baik dengan *F1-score* sebesar 0,5000. *Dataset* KC1 dan PC5 menunjukkan

performa yang sedang, dengan *F1-score* masing-masing sebesar 0,1579 dan 0,2154.

Sementara itu, meskipun beberapa *dataset* seperti PC1 mencatat akurasi tinggi (92,96%), *F1-score*-nya rendah (0,2857), yang menandakan ketidakseimbangan pengenalan antar kelas masih menjadi masalah serius.

Secara keseluruhan, hasil uji coba pada skenario ini menegaskan bahwa akurasi yang tinggi tidak dapat dijadikan satu-satunya indikator untuk menilai kinerja model pada data tidak seimbang. Nilai *F1-score* yang rendah pada sebagian besar *dataset* menunjukkan bahwa model cenderung bias terhadap kelas mayoritas dan mengalami kesulitan dalam mengenali kelas *defect*. Oleh karena itu, penerapan metode penyeimbangan data tetap diperlukan untuk meningkatkan kemampuan model dalam mendeteksi kelas minoritas secara lebih adil dan akurat.

4.3.2 Hasil Uji Coba dengan Pembagian 80:20

Pada pengujian ini, model *Random Forest* dilatih menggunakan data tanpa penyeimbangan (tanpa SMOTE), dengan distribusi kelas mayoritas dan minoritas dibiarkan dalam kondisi aslinya. Data dibagi dengan rasio 80% untuk pelatihan dan 20% untuk pengujian, guna memberikan proporsi data uji yang lebih representatif dalam menilai kinerja model terhadap data tidak seimbang. Evaluasi dilakukan menggunakan metrik akurasi dan *F1-score* untuk mengukur performa keseluruhan dan kemampuan mendeteksi kelas minoritas (*defect*).

Tabel 4. 5 Hasil uji coba skenario 1 dengan rasio 80:20

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8333	0,0000
2.	JM1	0,7958	0,1264
3.	KC1	0,7553	0,2162
4.	KC3	0,7692	0,1818

No	Dataset	Accuracy	F1-Score
5.	KC4	0,7958	0,1264
6.	MC1	0,9749	0,0000
7.	MC2	0,8400	0,6000
8.	MW1	0,8039	0,0000
9.	PC1	0,9078	0,0000
10.	PC2	0,9732	0,0000
11.	PC3	0,8380	0,0000
12.	PC4	0,8798	0,0606
13.	PC5	0,6968	0,2464

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pada Tabel 4.5 menunjukkan bahwa akurasi model secara umum cukup tinggi, dengan nilai berkisar antara 69% hingga 97%. Namun, tingginya akurasi ini tidak selalu mencerminkan performa yang baik dalam mengenali kelas *defect*. Pada sebagian besar *dataset* seperti CM1, MC1, MW1, PC1, PC2, dan PC3 model mencatat *F1-score* sebesar 0, yang mengindikasikan kegagalan total dalam mengidentifikasi kelas minoritas. Hal ini menunjukkan bahwa model cenderung bias terhadap kelas mayoritas.

Beberapa pengecualian terlihat pada *dataset* MC2, yang mencapai *F1-score* tertinggi sebesar 0,6000, menunjukkan kemampuan model yang relatif baik dalam mengenali kelas *defect*. *Dataset* KC1 dan PC5 juga menunjukkan *F1-score* yang lebih baik dibandingkan yang lain. KC1 dengan *F1-score* sebesar 0,2162 dan PC5 sebesar 0.2464, meskipun masih tergolong rendah.

Secara keseluruhan, hasil ini menunjukkan bahwa meskipun rasio 80:20 memberikan data latih yang cukup besar, ketidakseimbangan kelas tetap menjadi hambatan utama. Nilai *F1-score* yang rendah pada sebagian besar *dataset* menegaskan pentingnya penerapan teknik penyeimbangan data untuk meningkatkan kemampuan model dalam mengenali kelas *defect* secara adil dan efektif.

4.3.3 Hasil Uji Coba dengan Pembagian 75:25

Pada pengujian ini, model *Random Forest* dilatih menggunakan data yang tidak diseimbangkan, tanpa penerapan metode SMOTE. Distribusi antara kelas mayoritas dan minoritas dibiarkan sebagaimana kondisi awal *dataset*. Data kemudian dibagi dengan skema 75% untuk pelatihan dan 25% untuk pengujian.

Rasio ini dipilih untuk mengevaluasi sejauh mana performa model tetap konsisten ketika porsi data uji lebih besar dibandingkan skema sebelumnya. Dengan jumlah data uji yang lebih banyak, diharapkan hasil evaluasi dapat memberikan gambaran yang lebih akurat terhadap kemampuan generalisasi model dalam kondisi distribusi kelas yang tidak seimbang. Metrik yang digunakan meliputi akurasi dan *f1-score*. Tabel berikut menyajikan hasil pengujian model terhadap 13 *dataset* pada skenario ini:

Tabel 4. 6 Hasil uji coba skenario 1 dengan rasio 75:25

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8537	0,0000
2.	JM1	0,7955	0,1076
3.	KC1	0,7601	0,2680
4.	KC3	0,7959	0,1667
5.	KC4	0,8125	0,7692
6.	MC1	0,9738	0,0000
7.	MC2	0,7812	0,2222
8.	MW1	0,7969	0,0000
9.	PC1	0,9153	0,1176
10.	PC2	0,9733	0,0000
11.	PC3	0,8519	0,0000
12.	PC4	0,8882	0,0526
13.	PC5	0,7103	0,2706

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pengujian model pada skenario uji coba tanpa penyeimbangan dengan pembagian data sebesar 75% untuk pelatihan dan 25% untuk pengujian menunjukkan variasi performa yang cukup signifikan di antara 13 *dataset* yang

diuji. Secara umum, sebagian besar *dataset* menghasilkan akurasi tinggi dengan rentang antara 71% hingga 97%. *Dataset* dengan akurasi tertinggi meliputi MC1 (97,38%), PC2 (97,33%), dan PC1 (91,53%).

Namun, seperti halnya pada skenario sebelumnya (rasio 80:20), tingginya nilai akurasi tidak selalu mencerminkan kemampuan model dalam mendeteksi kelas minoritas. Beberapa *dataset*, seperti CM1, MC1, PC2, dan PC3, memiliki nilai *F1-score* sebesar nol, menandakan bahwa model sepenuhnya gagal mengidentifikasi data *defect* pada data uji. Hal ini menunjukkan bahwa distribusi kelas yang tidak seimbang masih menjadi tantangan signifikan, di mana model cenderung hanya fokus pada kelas mayoritas.

Meski demikian, beberapa *dataset* menunjukkan hasil yang lebih menjanjikan. KC4 menjadi *dataset* dengan performa terbaik, mencatat *F1-score* sebesar 0,7692, yang mengindikasikan kemampuan model dalam mengenali kedua kelas secara lebih proporsional. Selain itu, *dataset* KC1 dan PC5 juga menunjukkan performa yang relatif baik dengan *F1-score* di atas 0,26, meskipun masih jauh dari ideal.

Secara keseluruhan, hasil pada skenario ini kembali menegaskan bahwa pembagian data latih sebesar 75% memang memberi ruang yang cukup bagi model untuk belajar, namun tidak serta-merta mampu mengatasi tantangan ketidakseimbangan kelas. Oleh karena itu, intervensi tambahan seperti teknik penyeimbangan kelas tetap diperlukan agar performa model menjadi lebih menyeluruh dan adil terhadap semua kelas.

4.3.4 Hasil Uji Coba dengan Pembagian 70:30

Dalam pengujian ini, data pelatihan dan pengujian dibagi dengan skema 70% untuk pelatihan dan 30% untuk pengujian, tanpa dilakukan SMOTE. Artinya, distribusi kelas mayoritas dan minoritas dibiarkan tidak seimbang seperti kondisi awal *dataset*.

Skema ini digunakan untuk mengamati performa model saat data latih relatif lebih sedikit dan porsi data uji lebih besar, yang berpotensi menantang kemampuan model dalam mengenali pola, terutama pada kelas minoritas. Evaluasi tetap dilakukan menggunakan metrik akurasi dan *f1-score* guna menilai efektivitas klasifikasi model dalam kondisi ketidakseimbangan data. Tabel berikut menyajikan hasil pengujian model terhadap 13 *dataset* pada skenario ini:

Tabel 4. 7 Hasil uji coba skenario 1 dengan rasio 70:30

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8485	0,0000
2.	JM1	0,7979	0,1061
3.	KC1	0,7493	0,2992
4.	KC3	0,7797	0,1333
5.	KC4	0,7632	0,7273
6.	MC1	0,9732	0,0000
7.	MC2	0,7105	0,4211
8.	MW1	0,8158	0,0000
9.	PC1	0,9245	0,0000
10.	PC2	0,9643	0,0000
11.	PC3	0,8611	0,0000
12.	PC4	0,8760	0,1111
13.	PC5	0,7276	0,3137

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pada Tabel 4.7 menunjukkan bahwa meskipun sebagian besar *dataset* mencatat akurasi tinggi (antara 71% hingga 97%), nilai *F1-score* pada banyak *dataset* tetap sangat rendah. Beberapa di antaranya, seperti CM1, MW1, MC1,

PC1, PC2, dan PC3, mencatat *F1-score* sebesar 0, yang berarti model gagal total mengenali kelas minoritas dan cenderung hanya memprediksi kelas mayoritas.

Namun, terdapat beberapa pengecualian. *Dataset* KC4 menunjukkan performa terbaik dengan *F1-score* sebesar 0,7273. *Dataset* MC2 dan PC5 juga mencatat *F1-score* yang cukup baik, masing-masing 0,4211 dan 0,3137, menunjukkan kemampuan model dalam mengenali sebagian kasus *defect*. KC1 juga mencatat *F1-score* sebesar 0,2992, lebih tinggi dibanding banyak *dataset* lainnya.

Secara keseluruhan, meskipun skema 70:30 memperbesar ruang evaluasi model, ketidakseimbangan kelas tetap menjadi hambatan utama. Rendahnya *F1-score* pada sebagian besar *dataset* memperkuat pentingnya penerapan strategi penyeimbangan data untuk meningkatkan deteksi kelas minoritas secara efektif.

4.4 Evaluasi Skenario SMOTE

Skenario ini mengevaluasi kinerja model *Random Forest* setelah dilakukan penyeimbangan data menggunakan metode *Synthetic Minority Over-sampling Technique* (SMOTE). Teknik SMOTE digunakan untuk mengatasi permasalahan ketidakseimbangan kelas yang sering kali menurunkan performa model klasifikasi dalam mendeteksi kelas minoritas, khususnya pada domain prediksi *defect* perangkat lunak. Berbeda dengan pendekatan *baseline*, pada skenario ini data minoritas diperbanyak secara sintetis hingga jumlahnya setara dengan data mayoritas.

Tujuan dari skenario ini adalah untuk mengamati sejauh mana teknik *oversampling* dapat meningkatkan performa model dalam mengenali kelas

minoritas tanpa mengubah struktur data secara drastis. Evaluasi dilakukan berdasarkan metrik-metrik utama seperti akurasi dan *F1-score*.

Sama seperti skenario *baseline*, pengujian dilakukan terhadap setiap *dataset* dengan empat variasi rasio pembagian data latih dan data uji: 90:10, 80:20, 75:25, dan 70:30. Hasil evaluasi dari masing-masing rasio akan disajikan pada subbab berikut dan dibandingkan dengan skenario lainnya untuk menilai efektivitas SMOTE dalam konteks klasifikasi data tidak seimbang.

4.4.1 Hasil Uji Coba dengan Pembagian 90:10

Pada pengujian ini, model *Random Forest* dievaluasi menggunakan data yang telah melalui proses penyeimbangan sebagaimana dijelaskan pada Subbab 4.5. Setelah data diseimbangkan menggunakan SMOTE, pembagian data dilakukan dengan skema 90% untuk pelatihan dan 10% untuk pengujian. Rasio ini dipilih agar model memiliki peluang lebih besar dalam mempelajari pola dari data pelatihan, khususnya pada kelas minoritas yang sebelumnya kurang terwakili. Evaluasi dilakukan pada data uji yang belum pernah dilihat oleh model, menggunakan dua metrik yaitu akurasi dan *f1-score*. Tabel 4.8 menyajikan hasil evaluasi model terhadap 13 *dataset* dalam skenario ini.

Tabel 4. 8 Hasil uji coba skenario 2 dengan rasio 90:10

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8246	0,8387
2.	JM1	0,6236	0,6048
3.	KC1	0,6897	0,6932
4.	KC3	0,7188	0,7097
5.	KC4	0,9231	0,9231
6.	MC1	0,8509	0,8505
7.	MC2	0,7059	0,7059
8.	MW1	0,7391	0,7600
9.	PC1	0,8217	0,8369
10.	PC2	0,8904	0,9012
11.	PC3	0,7831	0,8019

No	Dataset	Accuracy	F1-Score
12.	PC4	0,7793	0,7860
13.	PC5	0,6774	0,6774

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pengujian menunjukkan bahwa pada sebagian besar *dataset*, model *Random Forest* mampu mencapai kinerja yang baik setelah diterapkan metode SMOTE. Hal ini terlihat dari nilai *F1-score* yang tinggi dan selaras dengan nilai akurasi. Pada *dataset* KC4, model memperoleh nilai akurasi dan *F1-score* sebesar 0,9231, yang menunjukkan keseimbangan klasifikasi antara kedua kelas. Hal serupa juga terlihat pada dataset PC2 (*F1-score* = 0,9012) dan MC1 (*F1-score* = 0,8505), yang mengindikasikan bahwa data sintetis yang ditambahkan mampu memperkuat representasi kelas minoritas dan meningkatkan performa prediksi.

Meskipun terdapat peningkatan performa pada beberapa *dataset*, masih ditemukan variabilitas hasil pada *dataset* lainnya. Contohnya, *dataset* JM1 dan PC5 menunjukkan performa yang relatif rendah, dengan *F1-score* masing-masing sebesar 0,6048 dan 0,6774. Hal ini menandakan bahwa meskipun distribusi kelas telah diseimbangkan, kualitas data sintetis yang dihasilkan oleh SMOTE belum sepenuhnya mampu menangkap kompleksitas atau distribusi fitur kelas minoritas pada dataset tersebut. Dengan kata lain, efektivitas SMOTE sangat dipengaruhi oleh karakteristik intrinsik dari masing-masing *dataset*.

Sebagian besar dataset menunjukkan perbedaan yang kecil antara nilai akurasi dan *F1-score*. Hal ini menunjukkan bahwa model tidak hanya fokus pada prediksi kelas mayoritas, tetapi juga berhasil mempelajari pola kelas minoritas secara proporsional. Sebagai contoh, *dataset* KC3 memperoleh akurasi sebesar 0,7188 dan *F1-score* sebesar 0,7097, yang mencerminkan bahwa model bekerja

secara seimbang dalam mengklasifikasikan kedua kelas. Fenomena ini mengindikasikan bahwa SMOTE dapat memperbaiki bias model terhadap kelas mayoritas, terutama ketika digunakan bersamaan dengan proporsi pelatihan yang tinggi.

Secara keseluruhan, hasil dari skenario ini menunjukkan bahwa penerapan SMOTE pada rasio pelatihan sebesar 90% cukup efektif dalam meningkatkan kinerja model. Semakin besar jumlah data pelatihan yang tersedia, semakin besar pula peluang model dalam menyerap informasi dari data sintetis yang ditambahkan. Oleh karena itu, rasio 90:10 dalam skenario ini dapat dikatakan sebagai konfigurasi yang menguntungkan, khususnya untuk meningkatkan deteksi terhadap kelas minoritas.

4.4.2 Hasil Uji Coba dengan Pembagian 80:20

Pengujian pada skenario ini dilakukan untuk mengevaluasi kinerja model *Random Forest* setelah dilakukan penyeimbangan data menggunakan metode SMOTE dan kemudian dibagi dengan rasio 80% untuk data pelatihan dan 20% untuk data pengujian. Rasio ini dipilih karena merupakan salah satu konfigurasi yang umum digunakan dalam proses pembelajaran mesin, yang memungkinkan model memperoleh cukup data pelatihan untuk mengenali pola, sekaligus menyisakan proporsi data uji yang memadai untuk mengevaluasi performa secara adil.

Proses penyeimbangan data dilakukan sebelum proses pelatihan, dengan tujuan agar distribusi kelas tidak lagi condong terhadap kelas mayoritas. Evaluasi performa model dilakukan dengan menggunakan dua metrik utama, yaitu akurasi

dan *F1-score*. Akurasi menggambarkan proporsi keseluruhan prediksi yang benar, sedangkan *F1-score* memberikan penilaian yang lebih adil pada ketidakseimbangan kelas, karena memperhitungkan *precision* dan *recall*. Tabel 4.9 berikut menyajikan hasil evaluasi kinerja model terhadap ketiga belas *dataset* dalam skenario ini:

Tabel 4. 9 Hasil uji coba skenario 2 dengan rasio 80:20

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8246	0,8387
2.	JM1	0,6154	0,5959
3.	KC1	0,6379	0,6337
4.	KC3	0,8125	0,8125
5.	KC4	0,8462	0,8333
6.	MC1	0,8378	0,8368
7.	MC2	0,7576	0,7143
8.	MW1	0,8462	0,8478
9.	PC1	0,8178	0,8315
10.	PC2	0,8973	0,9062
11.	PC3	0,8069	0,8215
12.	PC4	0,7658	0,7778
13.	PC5	0,6673	0,6784

Sumber: dig*enerate* penulis dari hasil program *python* (2025)

Secara umum, hasil pengujian dengan rasio 80:20 menunjukkan performa model yang cukup stabil setelah diterapkan SMOTE. Beberapa *dataset* seperti CM1, MC1, MW1, dan PC2 menunjukkan nilai *F1-score* yang tinggi, masing-masing 0,8387, 0,8368, 0,8478, dan 0,9062. Hasil ini menunjukkan bahwa proses penyeimbangan berhasil meningkatkan kemampuan model dalam mengklasifikasikan kelas minoritas, serta menjaga presisi dan recall dalam batas yang seimbang.

Dataset-dataset seperti KC3 dan PC3 juga menunjukkan performa yang memuaskan, dengan *F1-score* 0,8125 dan 0,8215, yang menandakan bahwa data sintetis yang dihasilkan oleh SMOTE dapat merepresentasikan distribusi kelas

minoritas dengan baik, khususnya ketika didukung oleh kualitas data asli yang memadai dan tidak terlalu terdistorsi.

Namun demikian, model masih menunjukkan performa yang terbatas pada beberapa dataset, seperti JM1 (*F1-score* = 0,5959) dan KC1 (*F1-score* = 0,6337). Hal ini menunjukkan bahwa pada *dataset* dengan kompleksitas atau *skewness* fitur yang tinggi, SMOTE belum sepenuhnya efektif dalam menghasilkan data sintetis yang berkualitas. Hal ini dapat disebabkan oleh *overlap* antar kelas yang tinggi atau pola distribusi fitur yang tidak linier, sehingga data sintetis yang ditambahkan tidak sepenuhnya membantu model dalam meningkatkan performa klasifikasinya.

Sebagian besar hasil menunjukkan kesesuaian antara nilai akurasi dan *F1-score*, yang menandakan bahwa model tidak hanya mengandalkan prediksi terhadap kelas mayoritas, tetapi juga mempertimbangkan prediksi kelas minoritas dengan baik. Pada *dataset* MC2, nilai akurasi sebesar 0,7576 disertai dengan *F1-score* 0,7143, yang mengindikasikan bahwa model dapat menangani kedua kelas secara proporsional meskipun terdapat sedikit penurunan dibandingkan rasio pelatihan yang lebih besar (90%).

4.4.3 Hasil Uji Coba dengan Pembagian 75:25

Pengujian dalam skenario ini dilakukan dengan menggunakan rasio pembagian data sebesar 75% untuk pelatihan dan 25% untuk pengujian. Sebagaimana pada skenario sebelumnya, data telah diseimbangkan terlebih dahulu menggunakan metode SMOTE. Tujuan dari pengujian ini adalah untuk mengetahui sejauh mana performa model *Random Forest* tetap stabil dan efektif saat jumlah data pelatihan dikurangi secara bertahap, namun tetap

mempertahankan proporsi data uji yang lebih besar untuk evaluasi yang lebih representatif.

Evaluasi dilakukan dengan menggunakan dua metrik utama, yaitu akurasi dan *F1-score*. Metrik akurasi digunakan untuk mengukur proporsi total prediksi yang benar terhadap seluruh data uji, sedangkan *F1-score* digunakan untuk memberikan gambaran yang lebih adil dalam konteks klasifikasi data tidak seimbang, dengan memperhitungkan *trade-off* antara *precision* dan *recall*. Tabel 4.10 berikut menyajikan hasil pengujian model terhadap ketiga belas dataset dalam skenario ini:

Tabel 4. 10 Hasil uji coba skenario 2 dengan rasio 75:25

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8601	0,8734
2.	JM1	0,6213	0,6014
3.	KC1	0,6368	0,6343
4.	KC3	0,7342	0,7407
5.	KC4	0,7812	0,7742
6.	MC1	0,8393	0,8382
7.	MC2	0,6829	0,6667
8.	MW1	0,7699	0,7833
9.	PC1	0,8137	0,8286
10.	PC2	0,9041	0,9123
11.	PC3	0,8093	0,8221
12.	PC4	0,7622	0,7692
13.	PC5	0,6629	0,6645

Sumber: di*generate* penulis dari hasil program *python* (2025)

Sebagian besar *dataset* menunjukkan performa yang cukup baik dalam skenario ini, dengan nilai *F1-score* yang relatif stabil dan tinggi. *Dataset* seperti CM1 (0,8734), MC1 (0,8382), MW1 (0,7833), dan PC2 (0,9123) mempertahankan performa klasifikasi yang unggul. Hal ini menunjukkan bahwa model masih mampu belajar secara efektif meskipun jumlah data pelatihan sedikit berkurang dibandingkan skenario sebelumnya. Keberhasilan ini juga

mengindikasikan bahwa data sintetis yang ditambahkan melalui SMOTE tetap memberikan kontribusi positif terhadap kemampuan generalisasi model.

Terdapat beberapa kasus di mana *F1-score* lebih tinggi dibandingkan akurasi, seperti pada *dataset* PC1 (*Accuracy* = 0.8137, *F1-score* = 0.8286) dan PC3 (*Accuracy* = 0.8093, *F1-score* = 0.8221). Ini menunjukkan bahwa model lebih baik dalam menyeimbangkan *precision* dan *recall* untuk kelas minoritas, yang menjadi fokus utama dalam studi ini. Hasil ini memperkuat efektivitas SMOTE dalam memperbaiki sensitivitas model terhadap data minoritas.

Namun, performa model masih belum optimal pada beberapa *dataset* seperti JM1 (*F1-score* = 0.6014), KC1 (*F1-score* = 0.6343), dan PC5 (*F1-score* = 0.6645). Rendahnya skor ini mengindikasikan bahwa data sintetis yang dihasilkan mungkin belum cukup representatif untuk mencerminkan distribusi kelas minoritas secara akurat. Kemungkinan lain adalah struktur data yang kompleks atau noise tinggi pada fitur-fitur input, yang menyebabkan model kesulitan dalam mempelajari pola klasifikasi yang jelas.

Meskipun jumlah data pelatihan berkurang dibanding skenario 90:10 dan 80:20, penurunan performa tidak terlalu signifikan. Bahkan, pada beberapa *dataset* seperti CM1 dan PC2, *F1-score* meningkat dibandingkan skenario sebelumnya. Hal ini menunjukkan bahwa meskipun data pelatihan sedikit lebih kecil, data uji yang lebih besar memberikan evaluasi yang lebih robust dan mendorong model untuk berperforma lebih stabil.

4.4.4 Hasil Uji Coba dengan Pembagian 70:30

Pengujian pada skenario ini menggunakan rasio pembagian data sebesar 70% untuk pelatihan dan 30% untuk pengujian. Sebelumnya, seluruh dataset telah melalui proses penyeimbangan menggunakan metode SMOTE guna meningkatkan representasi kelas minoritas. Rasio ini dipilih untuk mengevaluasi performa model ketika proporsi data uji diperbesar, yang berarti tantangan evaluasi model menjadi lebih tinggi karena lebih banyak data yang belum pernah dilihat digunakan dalam pengujian. Skenario ini penting untuk menilai kestabilan generalisasi model dalam kondisi evaluasi yang ketat.

Model dievaluasi dengan dua metrik utama, yaitu akurasi dan *F1-score*, untuk memberikan gambaran menyeluruh mengenai kemampuan model dalam mengklasifikasikan data secara benar serta sensitivitasnya terhadap ketidakseimbangan kelas. Berikut ini adalah hasil evaluasi model *Random Forest* terhadap ketiga belas dataset pada skenario ini:

Tabel 4. 11 Hasil uji coba skenario 2 dengan rasio 70:30

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8304	0,8466
2.	JM1	0,6192	0,5935
3.	KC1	0,6494	0,6376
4.	KC3	0,7789	0,7835
5.	KC4	0,7179	0,6667
6.	MC1	0,8611	0,8634
7.	MC2	0,7755	0,7660
8.	MW1	0,8088	0,8169
9.	PC1	0,8140	0,8326
10.	PC2	0,9132	0,9202
11.	PC3	0,8127	0,8279
12.	PC4	0,7808	0,7884
13.	PC5	0,6599	0,6675

Sumber: di*generate* penulis dari hasil program *python* (2025)

Secara umum, hasil pada skenario ini menunjukkan bahwa model masih mampu mempertahankan performa yang baik meskipun proporsi data pengujian diperbesar. *Dataset* seperti PC2 (*F1-score* = 0,9202), MC1 (0,8634), dan MW1 (0,8169) menunjukkan bahwa kualitas model tetap konsisten dan *robust* dalam mengenali pola pada data yang belum pernah dilihat.

Seperti pada skenario sebelumnya, *F1-score* tetap menjadi indikator utama untuk menilai keberhasilan model dalam menghadapi data tidak seimbang. Beberapa dataset menunjukkan *F1-score* yang lebih tinggi dibandingkan akurasi, seperti PC1 (*Accuracy* = 0,8140, *F1-score* = 0,8326) dan PC3 (*Accuracy* = 0,8127, *F1-score* = 0,8279). Hal ini menunjukkan bahwa hasil prediksi pada kelas minoritas tetap optimal meskipun evaluasi dilakukan pada lebih banyak data uji.

Namun, tidak semua *dataset* menunjukkan performa optimal. *Dataset* seperti JM1 (*F1-score* = 0.5935) dan KC4 (*F1-score* = 0.6667) mengalami penurunan performa jika dibandingkan dengan rasio pembagian sebelumnya. Penurunan ini mengindikasikan bahwa data sintetis yang ditambahkan belum cukup mewakili pola sebenarnya, atau kemungkinan noise pada data asli memengaruhi hasil generalisasi model.

Dataset seperti CM1, MC1, dan PC2 terus menunjukkan kinerja yang unggul di semua skenario, termasuk pada skenario ini. Hal ini mengindikasikan bahwa kombinasi struktur data yang jelas, serta representasi data minoritas yang berhasil diperbaiki oleh SMOTE, mendukung pembelajaran model secara konsisten meskipun proporsi data uji bertambah.

4.5 Evaluasi Skenario SMOTE dengan *Filtering*

Skenario ini merupakan pengembangan dari pendekatan sebelumnya, di mana data diseimbangkan menggunakan metode SMOTE, namun disertai dengan proses *filtering* terhadap data sintetis. Tujuan dari *filtering* ini adalah untuk meningkatkan kualitas sampel sintetis yang digunakan dalam pelatihan model, dengan mengeliminasi sampel yang berpotensi mengganggu proses klasifikasi karena terlalu dekat dengan wilayah kelas mayoritas.

Filtering dilakukan menggunakan pendekatan *k-nearest neighbors* dengan menghitung proporsi tetangga dari kelas minoritas di sekitar setiap sampel sintetis. Hanya sampel sintetis yang memiliki minimal 25% tetangga dari kelas minoritas yang dipertahankan. Pendekatan ini didasarkan pada asumsi bahwa tidak semua data sintetis yang dihasilkan oleh SMOTE memiliki kualitas yang baik dalam mewakili distribusi kelas minoritas. Dengan kata lain, *filtering* berperan sebagai langkah seleksi kedua untuk memastikan hanya sampel yang benar-benar representatif dari kelas minoritas yang digunakan dalam pelatihan model.

Untuk meningkatkan ketepatan proses *filtering*, dilakukan seleksi fitur menggunakan nilai *feature importance* dari model *Random Forest*. Model dilatih terlebih dahulu pada data hasil SMOTE (gabungan data asli dan sintetis), kemudian dihitung tingkat kepentingan tiap fitur. Hanya fitur-fitur penting yang digunakan dalam perhitungan kedekatan antar sampel, agar evaluasi tetangga terdekat hanya mempertimbangkan dimensi yang paling relevan. Pendekatan ini sejalan dengan Aguilar-Ruiz (2024), yang menekankan pentingnya

pemanfaatan fitur-fitur spesifik kelas dalam proses klasifikasi**.** Dengan mengurangi pengaruh fitur yang tidak informatif, *filtering* menjadi lebih tajam dan terhindar dari *noise,* sehingga meningkatkan keandalan proses seleksi data sintetis.

Selain itu, proses ini juga mempertimbangkan temuan dari studi Peng *et al* (2022) yang menunjukkan bahwa rasio kelas optimal tidak selalu harus seimbang 1:1. Dalam beberapa kondisi, rasio 2:1 atau 3:1 justru menghasilkan model yang lebih stabil, terutama saat *oversampling* menghasilkan data sintetis yang kurang representatif. Berdasarkan hal tersebut, penelitian ini tidak secara ketat menargetkan keseimbangan sempurna, melainkan menekankan pada kualitas dan kemiripan structural data sintetis terhadap data minoritas asli, yang diukur melalui kedekatan pada fitur-fitur penting.

Sebelum proses SMOTE dan *filtering* diterapkan, seluruh fitur numerik terlebih dahulu ditransformasi menggunakan *logarithmic scaling* untuk mengurangi efek *skewness*. Transformasi ini bertujuan agar pembangkitan dan penyaringan data sintetis berlangsung dalam ruang fitur yang lebih terstandarisasi dari segi distribusi, sehingga menghindari bias akibat ketidakseimbangan skala antar fitur.

Evaluasi pada skenario ini bertujuan untuk menilai sejauh mana kombinasi SMOTE dan *filtering* berbasis kemiripan dapat memberikan peningkatan performa model dibandingkan dua skenario sebelumnya (*baseline* dan SMOTE tanpa *filtering*). Kinerja model diukur menggunakan metrik akurasi dan *F1-score*, serta diuji pada keempat rasio pembagian data latih dan uji yang sama, yaitu

90:10, 80:20, 75:25, dan 70:30. Hasil evaluasi dari tiap rasio akan dipaparkan pada subbab-subbab berikut sebagai dasar perbandingan antar skenario.

4.5.1 Hasil Uji Coba dengan Pembagian 90:10

Pada pengujian ini, model *Random Forest* dievaluasi menggunakan data yang telah melalui proses penyeimbangan sebagaimana dijelaskan pada Subbab 4.6. Setelah data diseimbangkan menggunakan SMOTE dan melalui proses *filtering*, pembagian data dilakukan dengan skema 90% untuk pelatihan dan 10% untuk pengujian. Rasio ini dipilih agar model memiliki peluang lebih besar dalam mempelajari pola dari data pelatihan, khususnya pada kelas minoritas yang sebelumnya kurang terwakili. Evaluasi dilakukan pada data uji yang belum pernah dilihat oleh model, menggunakan dua metrik yaitu akurasi dan *f1-score*. Tabel 4.12 menyajikan hasil evaluasi model terhadap 13 *dataset* dalam skenario ini.

Tabel 4. 12 Hasil uji coba skenario 3 rasio 90:10

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8070	0,8136
2.	JM1	0,6360	0,6138
3.	KC1	0,7514	0,7514
4.	KC3	0,8125	0,8000
5.	KC4	0,9231	0,9231
6.	MC1	0,8605	0,8615
7.	MC2	0,7647	0,7500
8.	MW1	0,8222	0,8333
9.	PC1	0,8295	0,8429
10.	PC2	0,8630	0,8734
11.	PC3	0,8095	0,8235
12.	PC4	0,7568	0,7652
13.	PC5	0,7206	0,7113

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pada Tabel 4.12 menunjukkan bahwa penggabungan SMOTE dengan *filtering* berbasis tetangga minoritas menghasilkan kinerja yang kompetitif dan stabil, baik dari sisi akurasi maupun *F1-score*. Mayoritas *dataset* menunjukkan

F1-score di atas 0,75, yang menandakan bahwa model mampu mengklasifikasikan kelas minoritas dengan lebih akurat dibandingkan skenario baseline maupun skenario SMOTE tanpa *filtering*.

Secara khusus, *dataset* seperti KC4, MC1, PC2, dan MW1 mencatatkan *F1-score* yang sangat tinggi (di atas 0,85), yang berarti *filtering* tidak mengurangi efektivitas SMOTE, bahkan memperbaikinya dengan cara menghapus data sintetis yang berisiko menimbulkan *noise*. *Dataset* KC1 dan KC3 juga menunjukkan performa yang sangat baik dan seimbang antara akurasi dan *F1-score*, menandakan bahwa *filtering* tetap menjaga kualitas data sintetis yang dipertahankan.

Dataset JM1, yang secara umum memiliki distribusi kelas yang sangat tidak seimbang dan kompleksitas fitur yang tinggi, mencatatkan perbaikan *F1-score* menjadi 0,6138, lebih tinggi dibandingkan *baseline* dan sedikit meningkat dibanding SMOTE murni. Ini menunjukkan bahwa *filtering* memberikan dampak positif meskipun dalam kondisi ekstrim, walaupun peningkatannya belum setinggi pada *dataset* lain.

Secara keseluruhan, kombinasi SMOTE dan *filtering* terbukti membantu memperbaiki kelemahan model dalam mengenali kelas *defect*, dengan tetap menjaga nilai akurasi yang kompetitif. Pendekatan ini berhasil menyaring data sintetis yang kurang representatif dan mempertahankan hanya sampel-sampel yang diyakini memiliki dukungan struktur lokal dari kelas minoritas, yang kemudian berdampak pada peningkatan kualitas prediksi model.

4.5.2 Hasil Uji Coba dengan Pembagian 80:20

Pada skenario ini, model *Random Forest* dievaluasi menggunakan data yang telah melalui proses penyeimbangan sebagaimana dijelaskan pada Subbab 4.6. Setelah data diseimbangkan menggunakan SMOTE dan melalui proses *filtering*, data dibagi menjadi 80% data latih dan 20% data uji dipilih untuk menjaga keseimbangan antara kapasitas pelatihan model dan ruang pengujian yang cukup luas, sehingga performa model dapat dinilai lebih akurat terhadap data yang belum pernah dilihat sebelumnya. Evaluasi dilakukan dengan menggunakan metrik akurasi dan *f1-score* sebagai indikator utama. Tabel berikut menyajikan hasil pengujian model terhadap 13 *dataset* pada skenario ini:

Tabel 4. 13 Hasil uji coba skenario 3 rasio 80:20

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8509	0,8571
2.	JM1	0,6371	0,6122
3.	KC1	0,7159	0,7200
4.	KC3	0,8413	0,8214
5.	KC4	0,8462	0,8462
6.	MC1	0,8592	0,8608
7.	MC2	0,6970	0,6667
8.	MW1	0,7667	0,7529
9.	PC1	0,8411	0,8520
10.	PC2	0,8836	0,8917
11.	PC3	0,7905	0,8010
12.	PC4	0,7613	0,7623
13.	PC5	0,6802	0,6762

Sumber: di*generate* penulis dari hasil program *python* (2025)

Hasil pada Tabel 4.13 menunjukkan bahwa penerapan kombinasi SMOTE dan filtering berhasil memberikan hasil yang baik pada sebagian besar dataset. Terlihat bahwa *F1-score* berada pada kisaran tinggi (0,72 ke atas) untuk sebagian besar *dataset*, yang menandakan peningkatan kemampuan model dalam mengidentifikasi kelas minoritas secara lebih tepat.

Beberapa *dataset* seperti CM1, KC3, MC1, PC1, dan PC2 menampilkan *F1-score* di atas 0,85. Hal ini menunjukkan bahwa model tidak hanya unggul dalam mempelajari pola dari data pelatihan, tetapi juga mampu mengeneralisasi pengetahuan tersebut ke data uji dengan cukup efektif. Khususnya *dataset* PC2, yang memperoleh *F1-score* sebesar 0,8917, merupakan performa terbaik dalam skenario ini, menandakan bahwa distribusi kelas dan kualitas data sintetis yang dihasilkan sangat mendukung klasifikasi yang akurat.

Dataset KC1 dan KC4 juga menunjukkan kinerja yang stabil, dengan *F1-score* sebesar 0,72 dan 0,8462, menandakan bahwa *filtering* tidak menurunkan kualitas hasil SMOTE, melainkan membantu dalam menyaring *outlier* sintetis. Sementara itu, *dataset* JM1 yang dikenal memiliki ketidakseimbangan kelas ekstrem masih mencatat *F1-score* sekitar 0,6122, lebih baik dari skenario *baseline* dan SMOTE tanpa *filtering*, meskipun peningkatannya belum signifikan. Hal ini menunjukkan bahwa pendekatan *filtering* tetap memberikan manfaat meskipun pada data dengan tingkat kesulitan tinggi.

Beberapa *dataset* seperti MC2 dan PC5 memiliki *F1-score* di bawah 0,70, yaitu masing-masing 0,6667 dan 0,6762. Hal ini bisa disebabkan oleh pola distribusi fitur yang kurang ideal untuk interpolasi SMOTE, atau karena keberadaan minoritas yang terlalu sedikit sehingga *filtering* justru mengurangi jumlah data sintetis yang dapat digunakan.

Secara keseluruhan, skenario ini menunjukkan bahwa rasio pelatihan 80% masih memberikan hasil yang baik dalam hal kemampuan model mempelajari pola, serta cukup representatif dalam mengevaluasi performa terhadap data baru.

Filtering terhadap data hasil SMOTE terbukti mampu meningkatkan ketepatan klasifikasi pada kelas minoritas tanpa mengorbankan akurasi keseluruhan, dan tetap memberikan hasil yang konsisten di berbagai *dataset*.

4.5.3 Hasil Uji Coba dengan Pembagian 75:25

Pada skenario ini, model *Random Forest* dievaluasi menggunakan data yang telah melalui proses penyeimbangan sebagaimana dijelaskan pada Subbab 4.2. Data hasil penyeimbangan kemudian dibagi dengan skema 75% data latih dan 25% data uji.

Rasio ini memberikan proporsi pelatihan yang lebih moderat, dengan porsi data uji yang lebih besar dibandingkan rasio sebelumnya. Pendekatan ini memungkinkan pengujian yang lebih ketat terhadap kemampuan generalisasi model. Evaluasi dilakukan menggunakan metrik akurasi dan *F1-score*. Tabel berikut menyajikan hasil pengujian model terhadap 13 *dataset* pada skenario ini:

Tabel 4. 14 Hasil uji coba skenario 3 rasio 75:25

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8521	0,8645
2.	JM1	0,6439	0,6217
3.	KC1	0,6852	0,6699
4.	KC3	0,8987	0,8919
5.	KC4	0,8438	0,8387
6.	MC1	0,8781	0,8776
7.	MC2	0,6585	0,6316
8.	MW1	0,8214	0,8182
9.	PC1	0,8261	0,8418
10.	PC2	0,8959	0,9031
11.	PC3	0,7775	0,7904
12.	PC4	0,7784	0,7751
13.	PC5	0,7039	0,7063

Sumber: dig*enerate* penulis dari hasil program *python* (2025)

Hasil pengujian pada Tabel 4.14 menunjukkan bahwa kombinasi SMOTE dengan *filtering* tetap memberikan performa klasifikasi yang tinggi meskipun

dengan proporsi data uji yang lebih besar. Secara umum, *F1-score* mengalami peningkatan atau tetap stabil jika dibandingkan dengan rasio sebelumnya (80:20), menandakan bahwa model mampu mempertahankan kualitas prediksi meskipun ruang pengujian diperluas.

Beberapa *dataset* menampilkan performa luar biasa, seperti KC3 dengan *F1-score* sebesar 0,8919, serta PC2 dengan nilai tertinggi *F1-score* yaitu 0,9031. Ini menunjukkan bahwa pola dari kelas minoritas telah berhasil ditangkap dengan baik dan diterapkan secara efektif saat pengujian. Hal ini menjadi indikasi keberhasilan *filtering* dalam mempertahankan kualitas data sintetis.

Dataset seperti MC1, CM1, dan PC1 juga menunjukkan performa stabil dengan *F1-score* di atas 0,84, mendukung kesimpulan bahwa strategi penyeimbangan berbasis karakteristik fitur tidak hanya efektif saat data latih mendominasi (seperti pada rasio 90:10), tetapi juga ketika ruang evaluasi lebih besar.

Namun demikian, masih terdapat beberapa *dataset* dengan *F1-score* di bawah 0,70, seperti JM1 (0,6217), MC2 (0,6316), dan KC1 (0,6699). Hal ini menunjukkan bahwa meskipun pendekatan *filtering* berhasil mengurangi *noise* dari SMOTE, beberapa *dataset* tetap menantang karena karakteristik distribusi fiturnya atau ketimpangan internal yang tidak bisa diatasi sepenuhnya hanya dengan interpolasi berbasis tetangga.

Dataset PC5, meskipun termasuk yang memiliki *F1-score* lebih rendah (0,7063), tetap menunjukkan performa yang meningkat dibandingkan *baseline*. Ini

membuktikan bahwa metode ini mampu memperbaiki sensitivitas model terhadap kelas minoritas, bahkan dalam kondisi distribusi yang kurang ideal.

Secara keseluruhan, rasio 75:25 menghasilkan model dengan generalisasi yang baik dan tetap mempertahankan performa klasifikasi terhadap kelas minoritas. Strategi penyeimbangan yang diterapkan terbukti *robust* di berbagai tingkat pembagian data, dan *filtering* berkontribusi penting dalam menjaga kualitas sampel sintetis yang digunakan.

4.5.4 Hasil Uji Coba dengan Pembagian 70:30

Pada skenario ini, model *Random Forest* dievaluasi menggunakan data yang telah melalui proses penyeimbangan sebagaimana dijelaskan pada Subbab 4.2. Data hasil penyeimbangan kemudian dibagi dengan skema 70% data latih dan 30% data uji.

Rasio ini digunakan untuk memberikan tantangan yang lebih tinggi pada model, karena jumlah data pelatihan lebih terbatas sementara data uji lebih besar. Hal ini membantu mengevaluasi sejauh mana model mampu mempertahankan performa dengan data pelatihan yang lebih sedikit. Evaluasi dilakukan menggunakan metrik akurasi dan *f1-score*. Tabel berikut menyajikan hasil pengujian model terhadap 13 *dataset* pada skenario ini:

Tabel 4. 15 Hasil uji coba skenario 3 rasio 70:30

No	Dataset	Accuracy	F1-Score
1.	CM1	0,8538	0,8634
2.	JM1	0,6399	0,6178
3.	KC1	0,6815	0,6733
4.	KC3	0,8632	0,8602
5.	KC4	0,7692	0,7429
6.	MC1	0,8562	0,8569
7.	MC2	0,7143	0,6818
8.	MW1	0,8444	0,8489
9.	PC1	0,8398	0,8510

No	Dataset	Accuracy	F1-Score
10.	PC2	0,8995	0,9072
11.	PC3	0,7968	0,8067
12.	PC4	0,7778	0,7764
13.	PC5	0,7004	0,6951

Sumber: di*generate* penulis dari hasil program *python* (2025)

Meskipun data pelatihan pada skenario ini lebih terbatas (70%), hasil uji coba menunjukkan bahwa performa model tetap stabil dan kompetitif, baik dari segi akurasi maupun *F1-score*. Hal ini membuktikan bahwa strategi penyeimbangan data dan filtering yang digunakan cukup efektif, bahkan ketika kapasitas pelatihan dikurangi.

Dataset seperti PC2 dan CM1 menunjukkan performa terbaik dengan *F1-score* masing-masing 0,9072 dan 0,8634, membuktikan bahwa pola kelas minoritas tetap berhasil dipelajari dan diaplikasikan dengan baik. *Dataset* lain yang juga menunjukkan hasil sangat baik adalah MW1 (0,8489), PC1 (0,8510), KC3 (0,8602), dan MC1 (0,8569), yang mencerminkan ketahanan model terhadap penurunan proporsi data latih.

Meskipun demikian, beberapa *dataset* seperti JM1 (0,6178), MC2 (0,6818), dan KC1 (0,6733) masih menunjukkan *F1-score* di bawah 0,70. Hal ini bisa jadi disebabkan oleh kompleksitas pola pada kelas minoritas atau masih adanya ketimpangan fitur yang tidak seluruhnya teratasi oleh teknik SMOTE-*filtering*. Meski *F1-score* pada *dataset* ini tidak ideal, tetap terlihat adanya peningkatan signifikan dibandingkan *baseline* tanpa penyeimbangan.

Dataset PC5 juga menunjukkan peningkatan performa yang konsisten dibandingkan skenario *baseline*, dengan *F1-score* 0,6951, yang menunjukkan

bahwa model telah belajar lebih baik dalam mengenali kelas minoritas pada skenario ini.

Secara keseluruhan, skenario 70:30 ini menunjukkan bahwa model *Random Forest* dengan penyeimbangan berbasis karakteristik fitur tetap mampu menjaga akurasi dan sensitivitas model, meskipun porsi data pelatihan dikurangi. Dengan demikian, pendekatan ini terbukti efektif dan *robust* terhadap variasi skema pembagian data.

4.6 Analisa Hasil Uji Coba

Subbab ini menyajikan analisa terhadap hasil uji coba model klasifikasi menggunakan algoritma *Random Forest* pada data yang telah melalui proses penyeimbangan dan data asli tanpa penyeimbangan. Analisis dilakukan berdasarkan empat skenario pembagian data latih dan data uji, yaitu 90:10, 80:20, 75:25, dan 70:30. Tujuan dari analisa ini adalah untuk mengevaluasi pengaruh strategi penyeimbangan data terhadap performa model dalam mendeteksi kelas minoritas pada data *defect* perangkat lunak.

Penilaian performa model dilakukan dengan mempertimbangkan metrik evaluasi seperti akurasi dan *f1-score*. Selain itu, perbandingan dilakukan antar berbagai *dataset* untuk melihat konsistensi hasil, serta untuk mengidentifikasi kondisi di mana pendekatan tertentu lebih efektif. Analisis ini menjadi dasar dalam menarik kesimpulan terkait efektivitas penyeimbangan data dan dampaknya terhadap kemampuan generalisasi model.

4.6.1 Analisa Hasil Uji Coba Skenario *Baseline*

Pengujian Random Forest tanpa strategi penyeimbangan data dilakukan pada empat rasio data latih dan uji: 90:10, 80:20, 75:25, dan 70:30. Hasil menunjukkan akurasi tinggi, yaitu antara 65% hingga 97%. Namun tidak mencerminkan kemampuan mendeteksi kelas minoritas. Pada beberapa *dataset* seperti CM1, MC1, MW1, PC1, PC2, dan PC3, *F1-score* bernilai nol, menandakan model gagal mengenali kelas *defect*.

Gambar 4. 4 Perbandingan *Accuracy* dan *F1-Score* tanpa penyeimbangan
Sumber: di*generate* penulis dari hasil program *python* (2025)

Gambar 4.4 yang membandingkan akurasi dan *F1-score* memperjelas adanya kesenjangan signifikan di antara kedua metrik tersebut. Fenomena ini menegaskan bahwa metrik akurasi saja tidak cukup representative dalam konteks data tidak seimbang, karena model cenderung memprioritaskan kelas mayoritas. Ketika kelas minoritas tidak memiliki representasi memadai dalam proses pelatihan, kemampuan model dalam mengenali pola dari kelas tersebut menjadi sangat terbatas.

Temuan ini menyoroti bahwa ketidakseimbangan kelas memberikan dampak langsung terhadap kegagalan model dalam mempelajari dan memprediksi kelas minoritas. Tanpa intervensi seperti teknik *oversampling* atau *undersampling,* model akan bergantung pada distribusi kelas dominan dan menghasilkan prediksi yang tidak adil serta bias terhadap kelas mayoritas.

Meskipun demikian, terdapat beberapa pengecualian menarik. *Dataset* KC4 menunjukkan performa yang relatif konsisten, dengan *F1-score* tertinggi mencapai 0,7692, bahkan tanpa penyeimbangan eksplisit. Hal ini mengindikasikan bahwa distribusi fitur atau proporsi kelas dalam *dataset* tersebut relatif seimbang atau lebih informatif untuk kedua kelas.

Dataset MC2 juga mencatat performa yang cukup baik, khususnya pada rasio 90:10, 80:20, dan 70:30, dengan *F1-score* antara 0,2222 hingga 0,4211. Hal ini menunjukkan bahwa keberadaan sampel minoritas yang cukup pada data uji tetap dapat membantu model mengenali pola klasifikasi dari kelas *defect*, meskipun representasinya dalam data latih terbatas.

Selain itu, *dataset* KC1 dan PC5 juga menunjukkan nilai *F1-score* yang relatif kompetitif (sekitar 0,27 hingga 0,31) pada beberapa rasio, menandakan bahwa model tidak sepenuhnya mengabaikan kelas minoritas dalam kasus tertentu. Namun demikian, performa ini masih belum cukup optimal dan tidak konsisten di seluruh variasi rasio.

Secara keseluruhan, hasil ini menunjukkan bahwa pengaturan rasio data latih dan uji saja tidak cukup untuk mengatasi masalah ketidakseimbangan kelas. Tanpa adanya intervensi berupa teknik penyeimbangan data berbasis fitur atau strategi lainnya, model *Random Forest* akan tetap bias terhadap kelas mayoritas dan tidak mampu mendeteksi *defect* secara efektif. Oleh karena itu, penerapan strategi penyeimbangan tetap diperlukan untuk mencapai performa klasifikasi yang lebih akurat dan adil.

4.6.2 Analisa Hasil Uji Coba Skenario SMOTE

Skenario kedua dalam penelitian ini menerapkan teknik SMOTE (*Synthetic Minority Over-sampling Technique*) untuk mengatasi ketidakseimbangan kelas pada setiap *dataset* sebelum dilakukan pelatihan model *Random Forest*. Evaluasi dilakukan dengan rasio pembagian data latih dan uji sebesar 90:10, 80:20, 75:25, dan 70:30, dengan tujuan menilai sejauh mana SMOTE mampu meningkatkan kinerja model terutama dalam mengenali kelas *defect*.

Berdasarkan hasil uji coba, terlihat bahwa secara umum terjadi peningkatan *F1-score* yang signifikan dibandingkan skenario *baseline*. Hampir seluruh *dataset* menunjukkan nilai *F1-score* yang lebih tinggi dan stabil dapat dilihat pada

Gambar 4.5, menandakan bahwa SMOTE berhasil meningkatkan representasi kelas minoritas sehingga model dapat mempelajari pola dengan lebih seimbang.

Gambar 4. 5 Perbandingan *Accuracy* dan *F1-Score* dengan SMOTE
Sumber: di*generate* penulis dari hasil program *python* (2025)

Pada *dataset* CM1, PC1, PC2, dan PC3, yang sebelumnya gagal mendeteksi kelas minoritas (*F1-score* = 0), kini mencatatkan *F1-score* di atas 0,80 pada hampir semua rasio. Ini menunjukkan bahwa SMOTE mampu mengubah distribusi kelas menjadi lebih seimbang secara efektif, dan hasilnya berdampak

langsung terhadap kemampuan model dalam mengklasifikasikan kedua kelas dengan lebih adil.

Dataset JM1, yang memiliki distribusi awal sangat timpang, mengalami peningkatan kinerja meskipun tidak sekuat dataset lainnya. Nilai *F1-score* berkisar antara 0,59 hingga 0,60, sedikit lebih baik dari skenario *baseline*, tetapi tetap menunjukkan tantangan dalam menangani ketimpangan ekstrem, terutama pada dataset dengan jumlah fitur dan ukuran besar.

Sementara itu, *dataset* seperti KC4, MC1, dan MW1, yang sebelumnya sudah menunjukkan performa baik tanpa penyeimbangan, tetap mempertahankan kinerjanya bahkan cenderung meningkat pada beberapa rasio. Hal ini menunjukkan bahwa penerapan SMOTE tidak mengganggu struktur data yang sudah seimbang secara alami, dan justru dapat memperkuat generalisasi model.

Jika ditinjau dari segi rasio pembagian data, skenario 90:10 dan 75:25 tampaknya memberikan hasil *F1-score* yang cukup stabil dan tinggi di banyak *dataset*, menandakan bahwa memberikan lebih banyak data latih kepada model dapat memperkuat pemahaman pola dari kelas minoritas hasil SMOTE. Rasio 70:30 juga memberikan hasil yang baik, meskipun pada beberapa *dataset* seperti KC4 dan JM1, performa sedikit menurun dibanding rasio yang lebih kecil.

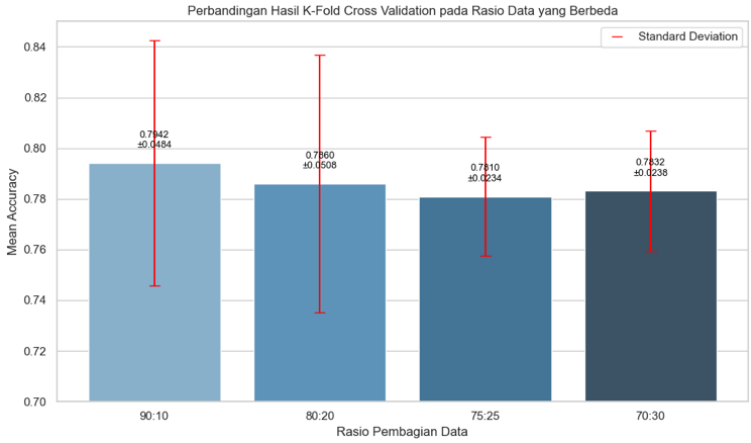
Secara keseluruhan, penerapan SMOTE menunjukkan efektivitasnya dalam memperbaiki bias terhadap kelas mayoritas yang muncul pada skenario *baseline*. Nilai *F1-score* yang meningkat secara signifikan membuktikan bahwa model mampu mendeteksi kelas *defect* dengan lebih baik, sekaligus menjaga nilai akurasi tetap kompetitif. Temuan ini memperkuat urgensi penggunaan teknik

penyeimbangan data sebagai bagian dari *pipeline* dalam prediksi *defect* perangkat lunak, terutama pada kondisi data yang sangat tidak seimbang.

4.6.3 Analisa Hasil Uji Coba Skenario SMOTE dengan *filtering*

Skenario ketiga dalam penelitian ini menerapkan strategi penyeimbangan data menggunakan SMOTE berbasis karakteristik fitur, yang dikombinasikan dengan proses *filtering* berdasarkan kedekatan jarak antar data minoritas. Tujuan utama dari pendekatan ini adalah untuk menghasilkan data sintetis yang tidak hanya mewakili kelas minoritas, tetapi juga relevan dan berkualitas, sehingga menghindari *noise* dan *overfitting* yang umum terjadi pada metode penyeimbangan konvensional.

Eksperimen dilakukan pada empat rasio pembagian data latih dan uji (90:10, 80:20, 75:25, dan 70:30) dengan hasil *k-fold cross validation* menunjukkan rata-rata *accuracy* tertinggi pada rasio 90:10 (*mean* = 0,7942; std = 0,0484). Rasio 90:10 unggul dalam rata-rata akurasi, sedangkan 75:25 dan 70:30 lebih stabil karena memiliki standar deviasi yang lebih rendah.



Gambar 4. 6 Hasil *k-fold cross validation* pada skenario 3
Sumber: di*generate* penulis dari hasil program *python* (2025)

Gambar 4. 7 Perbandingan *Accuracy* dan *F1-Score* pada SMOTE + *filtering*
Sumber: di*generate* penulis dari hasil program *python* (2025)

Secara umum, terdapat peningkatan signifikan pada nilai *F1-score* dibandingkan dengan skenario *baseline* (tanpa penyeimbangan data) maupun skenario SMOTE biasa. Pada skenario *baseline*, sejumlah *dataset* seperti CM1, PC1, PC2, dan PC3 menunjukkan *F1-score* = 0, yang menandakan bahwa model gagal total dalam mengenali kelas minoritas. Setelah diterapkan SMOTE biasa, *F1-score* meningkat, namun belum stabil pada semua rasio. Beberapa *dataset*

seperti MC2 dan JM1 masih menunjukkan performa yang fluktuatif atau rendah akibat sampel sintetis yang tidak selektif.

Dengan penerapan *filtering* setelah SMOTE, peningkatan kinerja model menjadi lebih konsisten dan signifikan. Pada *dataset* PC1, nilai *F1-score* meningkat dari 0 pada *baseline* menjadi 0,8326 pada SMOTE biasa, dan lebih tinggi lagi menjadi 0,8510 pada SMOTE dengan *filtering*. Hal yang sama terjadi pada *dataset* KC3, di mana *F1-score* meningkat dari 0,1333 (*baseline*) menjadi 0,7835 (SMOTE biasa), dan mencapai 0,8602 pada skenario *filtering*.

Peningkatan yang konsisten juga terlihat pada *dataset* lain seperti KC4, MW1, PC4 dan PC5, yang menunjukkan bahwa proses *filtering* membantu menjaga kualitas data sintetis dengan mempertahankan kesesuaian distribusi spasial data minoritas. Sementara itu, pada *dataset* JM1 dan KC1 yang memiliki tingkat kompleksitas dan *noise* lebih tinggi, peningkatan tetap terjadi, meskipun nilainya tidak setinggi *dataset* lain, menunjukkan keterbatasan metode ini dalam kondisi distribusi yang ekstrem.

Secara keseluruhan, hasil ini mengonfirmasi bahwa metode SMOTE berbasis karakteristik fitur yang diikuti dengan proses *filtering* memberikan keunggulan yang nyata dibandingkan dua skenario sebelumnya. Teknik ini tidak hanya meningkatkan sensitivitas model terhadap kelas minoritas, tetapi juga menjaga stabilitas performa lintas rasio pembagian data, yang membuktikan efektivitasnya dalam mengatasi ketidakseimbangan kelas secara lebih optimal.

4.7 Perbandingan Kinerja Model

Bagian ini membahas perbandingan kinerja model *Random Forest* dalam tiga skenario pengujian berbeda, yaitu tanpa penyeimbangan data (*baseline*), dengan penyeimbangan menggunakan SMOTE, dan dengan SMOTE yang dikombinasikan dengan *filtering* berbasis karakteristik fitur. Evaluasi dilakukan pada empat variasi rasio pembagian data latih dan uji: 90:10, 80:20, 75:25, dan 70:30. Kinerja model diukur menggunakan metrik akurasi dan *F1-score*, yang masing-masing mewakili akurasi umum dan kemampuan model dalam mendeteksi kelas minoritas.

Pengujian terhadap model *Random Forest* tanpa strategi penyeimbangan dilakukan untuk mengevaluasi kemampuannya dalam mengklasifikasikan data dengan distribusi kelas tidak seimbang. Hasil evaluasi menunjukkan bahwa nilai akurasi secara umum tinggi, berada di kisaran 65% hingga 97%. Namun, tingginya akurasi ini tidak mencerminkan kemampuan deteksi kelas minoritas secara akurat.

Pada sejumlah *dataset*, seperti CM1, MC1, MW1, PC1, PC2, dan PC3, metrik *F1-score* tercatat bernilai nol di seluruh rasio, menandakan bahwa model gagal total dalam mendeteksi kelas *defect*. Hal ini mengindikasikan bahwa model cenderung bias terhadap kelas mayoritas dan mengabaikan kelas minoritas yang tidak cukup terwakili dalam proses pelatihan.

Beberapa pengecualian muncul pada *dataset* KC4, yang mencatat *F1-score* hingga 0,7692 tanpa penyeimbangan eksplisit, serta MC2, KC1, dan PC5, yang menunjukkan *F1-score* berkisar antara 0,22 hingga 0,31 pada beberapa rasio.

Namun, performa tersebut tidak konsisten dan belum mencerminkan keberhasilan model dalam menangani ketidakseimbangan kelas secara menyeluruh.

Secara keseluruhan, skenario ini menunjukkan bahwa tanpa intervensi penyeimbangan, model cenderung menghasilkan klasifikasi yang bias dan tidak adil terhadap kelas *defect*. Oleh karena itu, dibutuhkan strategi penyeimbangan untuk meningkatkan sensitivitas model terhadap kelas minoritas.

Skenario kedua menerapkan teknik SMOTE (*Synthetic Minority Over-sampling Technique*) sebelum pelatihan model *Random Forest*. Hasil pengujian menunjukkan peningkatan *F1-score* yang signifikan dibandingkan skenario baseline. Hampir seluruh *dataset* menunjukkan peningkatan yang stabil dan substansial dalam kemampuan mendeteksi kelas minoritas.

Dataset seperti CM1, PC1, PC2, dan PC3, yang sebelumnya memiliki *F1-score* nol, kini menunjukkan nilai di atas 0,80 pada sebagian besar rasio. *Dataset* JM1, meskipun sangat tidak seimbang, mengalami peningkatan *F1-score* hingga kisaran 0,59–0,60, menunjukkan bahwa SMOTE mampu memperbaiki distribusi kelas secara moderat meskipun tidak sempurna.

Dataset yang sebelumnya sudah cukup seimbang seperti KC4, MC1, dan MW1 tidak mengalami penurunan performa setelah penerapan SMOTE. Bahkan, performanya cenderung meningkat, menandakan bahwa SMOTE tidak merusak distribusi data yang sudah baik, dan justru memperkuat generalisasi model.

Rasio 90:10 dan 75:25 menghasilkan performa *F1-score* yang tinggi dan stabil, menunjukkan pentingnya proporsi data latih yang mencukupi untuk mempelajari pola dari kelas minoritas hasil *oversampling*. Hasil ini menegaskan

bahwa SMOTE merupakan strategi yang efektif dalam meningkatkan sensitivitas model terhadap kelas *defect*.

Skenario ketiga menggabungkan SMOTE dengan *filtering* berbasis karakteristik fitur, yang bertujuan menghasilkan data sintetis yang relevan dan berkualitas. *Filtering* dilakukan berdasarkan kedekatan jarak antar data minoritas, sehingga hanya sampel sintetis yang konsisten secara spasial yang dipertahankan. Hasil *k-fold cross validation* menunjukkan rata-rata akurasi tertinggi pada rasio 90:10 (mean = 0,7942; std = 0,0484). Rasio 75:25 dan 70:30 menunjukkan performa yang lebih stabil karena memiliki standar deviasi akurasi yang lebih rendah, menandakan bahwa filtering berkontribusi terhadap kestabilan hasil pelatihan model.

Dari sisi *F1-score*, skenario ini menunjukkan performa terbaik di antara ketiga pendekatan. Pada *dataset* PC1, *F1-score* meningkat dari 0 (*baseline*) menjadi 0,8326 (SMOTE) dan memperoleh 0,8510 (SMOTE dengan *filtering*)**.** Demikian pula pada KC3, yang meningkat dari 0,1333 menjadi 0,7835 dan 0,8602 pada smote dengan *filtering*. Hampir semua *dataset* mengalami peningkatan *F1-score* yang lebih tinggi dan stabil dibandingkan skenario sebelumnya.

Filtering membantu menghindari *noise* yang dihasilkan oleh data sintetis yang tidak relevan, sehingga menghasilkan distribusi kelas minoritas yang lebih akurat dan mudah dipelajari oleh model. Beberapa *dataset* seperti KC4, MW1, PC4, dan PC5 mencatatkan performa optimal, sementara JM1 dan KC1 tetap menunjukkan peningkatan meskipun terbatas, mencerminkan tantangan dari

distribusi data yang sangat kompleks. Tabel berikut merangkum keunggulan relatif dari masing-masing skenario:

Tabel 4. 16 Ringkasan perbandingan

Skenario	Akurasi	F1-score	Stabilitas	Deteksi Kelas Minoritas
Tanpa Penyeimbangan	Tinggi tapi menyesatkan	Sangat rendah	Tidak stabil	Gagal mendeteksi
SMOTE	Tinggi dan kompetitif	Meningkat signifikan	Moderat	Umumnya berhasil
SMOTE + Filtering	Tinggi dan konsisten	Tertinggi dan stabil	Paling stabil	Paling efektif

Dari hasil ini, dapat disimpulkan bahwa SMOTE dengan *filtering* memberikan hasil terbaik secara keseluruhan, baik dalam hal akurasi, kestabilan model, maupun sensitivitas terhadap kelas minoritas. Pendekatan ini efektif dalam menangani ketidakseimbangan kelas tanpa menimbulkan *noise* berlebih, menjadikannya pilihan utama dalam skenario prediksi *defect* perangkat lunak berbasis data tidak seimbang.

4.8 Integrasi Penelitian

Penelitian ini tidak hanya berupaya menghasilkan solusi ilmiah terhadap permasalahan ketidakseimbangan kelas dalam data, tetapi juga mencerminkan nilai-nilai Islam yang luhur, khususnya dalam hal keadilan dan keteraturan ciptaan.

Dalam QS. Al-Mulk ayat 3, Allah SWT berfirman:

ٱلَّذِى خَلَقَ سَبۡعَ سَمَٰوَٰتٖ طِبَاقٗاۖ مَّا تَرَىٰ فِى خَلۡقِ ٱلرَّحۡمَٰنِ مِن تَفَٰوُتٖۖ فَٱرۡجِعِ ٱلۡبَصَرَ هَلۡ تَرَىٰ مِن فُطُورٖ

"Yang menciptakan tujuh langit berlapis-lapis. Tidak akan kamu lihat sesuatu yang tidak seimbang pada ciptaan Tuhan Yang Maha Pengasih. Maka lihatlah sekali lagi, adakah kamu lihat sesuatu yang cacat?" (QS. Al-Mulk: 3)

Tafsir Al-Madinah Al-Munawwarah menjelaskan bahwa ayat ini menunjukkan kesempurnaan ciptaan Allah SWT yang bebas dari cacat, retakan, maupun ketidakseimbangan. Langit sebagai salah satu makhluk besar diciptakan dengan struktur yang rapi, harmonis, dan tidak mengandung kejanggalan sedikit pun, sebuah wujud nyata dari keteraturan dan keadilan ilahiah.

Nilai-nilai tersebut menjadi pedoman dalam pendekatan ilmiah yang dilakukan peneliti, khususnya dalam menganalisis distribusi data dan menangani ketidakseimbangan kelas. Upaya untuk mendeteksi distribusi yang timpang serta melakukan balancing melalui teknik statistik atau machine learning sejatinya merupakan bentuk ikhtiar manusia dalam meneladani keteraturan yang ditekankan dalam ayat ini.

Nilai-nilai Islam yang luhur juga tercermin dalam QS. An-Nahl ayat 90, di mana Allah SWT menekankan pentingnya keadilan dalam seluruh aspek kehidupan, termasuk dalam pengambilan keputusan dan perlakuan terhadap sesama:

إِنَّ ٱللَّهَ يَأۡمُرُ بِٱلۡعَدۡلِ وَٱلۡإِحۡسَٰنِ وَإِيتَآيِٕ ذِي ٱلۡقُرۡبَىٰ وَيَنۡهَىٰ عَنِ ٱلۡفَحۡشَآءِ وَٱلۡمُنكَرِ وَٱلۡبَغۡيِۚ يَعِظُكُمۡ لَعَلَّكُمۡ تَذَكَّرُونَ

"Sesungguhnya Allah menyuruh (kamu) berlaku adil dan berbuat kebajikan, memberi bantuan kepada kerabat, dan Dia melarang (melakukan) perbuatan keji, kemungkaran dan permusuhan. Dia memberi pengajaran kepadamu agar kamu dapat mengambil pelajaran." (QS. An-Nahl: 90)

Dalam Tafsir Al-Mukhtashar, disebutkan bahwa perintah Allah SWT untuk berlaku adil mencakup kewajiban menunaikan hak-hak baik terhadap Allah SWT maupun terhadap sesama manusia, serta menolak perlakuan yang berat sebelah kecuali atas dasar yang benar. Ayat ini juga mengajarkan pentingnya

menempatkan segala sesuatu pada tempatnya dan melarang kezaliman serta arogansi dalam perlakuan terhadap orang lain.

Ketika dikaitkan dengan konteks penelitian ini, prinsip keadilan sangat relevan dengan permasalahan ketidakseimbangan kelas dalam data. Ketimpangan distribusi data antara kelas mayoritas dan minoritas dapat menyebabkan bias dalam hasil model prediksi. Oleh karena itu, langkah-langkah yang diambil untuk menyeimbangkan data merupakan wujud nyata dari pengamalan nilai 'adl (keadilan) dalam penelitian ilmiah yaitu dengan tidak memihak atau mengabaikan satu kelas tertentu, tetapi memberikan representasi yang proporsional dan objektif sesuai kebutuhan analisis.

Dengan demikian, penyelesaian masalah ilmiah seperti *class imbalance* tidak hanya menjadi tanggung jawab teknis, tetapi juga mencerminkan kesadaran etis dan spiritual untuk menjunjung tinggi prinsip keadilan sebagaimana yang ditekankan dalam ajaran Islam.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan, berikut adalah kesimpulan yang dapat diambil:

1. Algoritma *Random Forest terbukti* efektif dalam memprediksi *defect* perangkat lunak. Model ini mampu menghasilkan kinerja yang baik secara umum. Namun, performanya cenderung menurun ketika digunakan pada data yang tidak seimbang, terutama dalam mengenali kelas minoritas.
2. Strategi penyeimbangan data berdasarkan karakteristik distribusi fitur berdampak positif terhadap kinerja model *Random Forest*. Penggunaan teknik SMOTE mampu meningkatkan nilai *F1-score* secara signifikan. Peningkatan yang paling konsisten diperoleh dari kombinasi SMOTE dengan *filtering* berdasarkan karakteristik data minoritas, yang membantu meningkatkan sensitivitas model tanpa mengorbankan akurasi secara keseluruhan. Hal ini menunjukkan pentingnya pemilihan teknik penyeimbangan yang mempertimbangkan distribusi fitur untuk menghasilkan model prediksi yang lebih adil dan andal.

5.2 Saran

Berdasarkan hasil penelitian, berikut beberapa saran untuk pengembangan penelitian selanjutnya:

1. Eksplorasi algoritma lain seperti XGBoost atau LightGBM dapat dilakukan untuk membandingkan efektivitas pendekatan penyeimbangan pada model yang berbeda.
2. Penggunaan metrik evaluasi tambahan seperti *G-Mean* atau MCC disarankan agar analisis performa model lebih menyeluruh pada kondisi data tidak seimbang.
3. Uji coba pada *dataset* lain di luar NASA MDP, baik dari proyek industri maupun *open source*, diperlukan untuk menguji generalisasi pendekatan ini secara lebih luas.
4. Optimasi parameter pada model *Random Forest* maupun metode penyeimbangan dapat membantu memperoleh kinerja yang lebih optimal dan konsisten.

DAFTAR PUSTAKA

Aguilar-Ruiz, J. S. (2024). *Class-specific feature selection for classification explainability*. http://arxiv.org/abs/2411.01204

Alhumam, A. (2022). Effective Prediction of Software Defects using Random-tree Entropy based Feature Selection Framework. *International Journal of Advanced Computer Science and Applications*, *13*(5), 348–354. https://doi.org/10.14569/IJACSA.2022.0130541

Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Ghadi, Y. Y., & Khan, M. A. (2024). Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. In *PeerJ Computer Science* (Vol. 10). peerj.com. https://doi.org/10.7717/peerj-cs.1860

Arun, C., & Lakshmi, C. (2022). Genetic algorithm-based oversampling approach to prune the class imbalance issue in software defect prediction. In *Soft Computing* (Vol. 26, Issue 23, pp. 12915–12931). Springer. https://doi.org/10.1007/s00500-021-06112-6

Ary Prandika Siregar, Dwi Priyadi Purba, Jojor Putri Pasaribu, & Khairul Reza Bakara. (2023). Implementasi Algoritma Random Forest Dalam Klasifikasi Diagnosis Penyakit Stroke. *Jurnal Penelitian Rumpun Ilmu Teknik*, *2*(4), 155–164. https://doi.org/10.55606/juprit.v2i4.3039

Azis, H., Purnawansyah, P., Fattah, F., & Putri, I. P. (2020). Performa Klasifikasi K-NN dan Cross Validation Pada Data Pasien Pengidap Penyakit Jantung. *ILKOM Jurnal Ilmiah*, *12*(2), 81–86. https://doi.org/10.33096/ilkom.v12i2.507.81-86

Bennin, K. E., Tahir, A., MacDonell, S. G., & Börstler, J. (2022). An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction. *IET Software*, *16*(2), 185–199. https://doi.org/10.1049/sfw2.12052

Cai, X., Niu, Y., Geng, S., Zhang, J., Cui, Z., Li, J., & Chen, J. (2020). An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. *Concurrency and Computation: Practice and Experience*, *32*(5). https://doi.org/10.1002/cpe.5478

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*(1), 321–357. https://doi.org/10.1613/jair.953

Chen, L. qiong, Wang, C., & Song, S. long. (2022). Software defect prediction based on nested-stacking and heterogeneous feature selection. In *Complex and Intelligent Systems* (Vol. 8, Issue 4, pp. 3333–3348). Springer.

https://doi.org/10.1007/s40747-022-00676-y

Cui, C., Liu, B., & Wang, S. (2022). WIFLF: An approach independent of the target project for cross-project defect prediction. *Journal of Software: Evolution and Process*, *34*(12). https://doi.org/10.1002/smr.2497

Dasari Rathna Nagabhushan. (2024). Manual Testing Vital Role: A Research Perspective. *International Journal of Advanced Research in Science, Communication and Technology*, 183–189. https://doi.org/10.48175/ijarsct-15229

Dewi, N. K., Mulyadi, S. Y., & Syafitri, U. D. (2012). Penerapan Metode Random Forest Dalam Driver Analysis. *Forum Statistika Dan Komputasi*, *16*(1), 35–43. http://journal.ipb.ac.id/index.php/statistika/article/view/5443

Hafid, H. (2023). Penerapan K-Fold Cross Validation untuk Menganalisis Kinerja Algoritma K-Nearest Neighbor pada Data Kasus Covid-19 di Indonesia. *Journal of Mathematics*, *6*(2), 161–168. http://www.ojs.unm.ac.id/jmathcos

Hairani, H., Widiyaningtyas, T., & Dwi Prasetya, D. (2024). Addressing Class Imbalance of Health Data: A Systematic Literature Review on Modified Synthetic Minority Oversampling Technique (SMOTE) Strategies. *JOIV: International Journal on Informatics Visualization*, *8*(3), 1310–1318.

Hajiar Yuliana. (2024). Hyperparameter Optimization of Random Forest for 5G Coverage Prediction. *Buletin Pos Dan Telekomunikasi*, *22*(1), 75–90. https://doi.org/10.17933/bpostel.v22i1.390

Istiqamah, N., & Rijal, M. (2024). Klasifikasi Ulasan Konsumen Menggunakan Random Forest dan SMOTE. *Journal of System and Computer Engineering (JSCE)*, *5*(1), 66–77. https://doi.org/10.61628/jsce.v5i1.1061

Janarthanan, R. (2021). *Software Defect Prediction based on Random Forest Classifier with Artificial Neural Networks*. *14*(1), 1105–1119.

Jiang, L., Yang, Z., Wang, D., Gong, H., Li, J., Wang, J., & Wang, L. (2024). Diabetes prediction model for unbalanced community follow-up data set based on optimal feature selection and scorecard. *Digital Health*, *10*. https://doi.org/10.1177/20552076241236370

Li, Z., Jing, X.-Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. *IET Software*, *12*(3), 161–175. https://doi.org/10.1049/iet-sen.2017.0148

Liu, W., Wang, B., & Wang, W. (2021). Deep Learning Software Defect Prediction Methods for Cloud Environments Research. *Scientific Programming*, *2021*. https://doi.org/10.1155/2021/2323100

Malhotra, R., Kashyap, V., & Sharma, V. (2022). Comparative study of Sampling Techniques for Software Defect Prediction. *8th International Conference on Advanced Computing and Communication Systems, ICACCS 2022*, 760–766.

https://doi.org/10.1109/ICACCS54159.2022.9785238

Naseem, R., Khan, B., Ahmad, A., Almogren, A., Jabeen, S., Hayat, B., & Shah, M. A. (2020). Investigating tree family machine learning techniques for a predictive system to unveil software defects. *Complexity*, *2020*. https://doi.org/10.1155/2020/6688075

Pahlevi, O.-, Amrin, A.-, & Handrianto, Y.-. (2023). Implementasi Algoritma Klasifikasi Random Forest Untuk Penilaian Kelayakan Kredit. *Jurnal Infortech*, *5*(1), 71–76. https://doi.org/10.31294/infortech.v5i1.15829

Peng, J., Shao, Y., & Huang, L. (2022). A General Framework for Finding the Optimal Imbalance Ratio in Sampling Methods. *2022 IEEE 5th International Conference on Electronics Technology (ICET)*, 1303–1307. https://doi.org/10.1109/ICET55676.2022.9825442

Pressman, R. S. (2010). Software Engineering A practitioner's Approach Seventh Edition. In *Proceedings of the Conference on The Future of Software Engineering*. McGraw Hill.

Qualetics, T. (2021). *Software Defect Management: Overview, Best Practices & AI Impact*. https://qualetics.com/software-defect-management-overview-best-practices-ai-impact/

Soe, Y. N., Santosa, P. I., & Hartanto, R. (2018). Software Defect Prediction Using Random Forest Algorithm. *2018 12th South East Asian Technical University Consortium (SEATUC)*, 1–5. https://doi.org/10.1109/SEATUC.2018.8788881

Suci Amaliah, Nusrang, M., & Aswi, A. (2022). Penerapan Metode Random Forest Untuk Klasifikasi Varian Minuman Kopi di Kedai Kopi Konijiwa Bantaeng. *VARIANSI: Journal of Statistics and Its Application on Teaching and Research*, *4*(3), 121–127. https://doi.org/10.35580/variansiunm31

Taskeen, A., Khan, S. U. R., & Felix, E. A. (2023). A research landscape on software defect prediction. *Journal of Software: Evolution and Process*, *35*(12). https://doi.org/10.1002/smr.2549

Thapa, S., Alsadoon, A., Prasad, P. W. C., Al-Dala'In, T., & Rashid, T. A. (2020). Software defect prediction using atomic rule mining and random forest. *CITISIA 2020 - IEEE Conference on Innovative Technologies in Intelligent Systems and Industrial Applications, Proceedings*. https://doi.org/10.1109/CITISIA50690.2020.9371797

Thomas, N. S., & Kaliraj, S. (2024a). An Improved and Optimized Random Forest Based Approach to Predict the Software Faults. *SN Computer Science*, *5*(5). https://doi.org/10.1007/s42979-024-02764-x

Thomas, N. S., & Kaliraj, S. (2024b). An Improved and Optimized Random Forest Based Approach to Predict the Software Faults. In *SN Computer Science* (Vol. 5, Issue 5). Springer. https://doi.org/10.1007/s42979-024-

02764-x

Xie, G., Xie, S., Peng, X., & Li, Z. (2021). Prediction of number of software defects based on SMOTE. *International Journal of Performability Engineering*, *17*(1), 123–134. https://doi.org/10.23940/ijpe.21.01.p12.123134

Zhao, Y., Damevski, K., & Chen, H. (2023). A Systematic Survey of Just-in-Time Software Defect Prediction. *ACM Computing Surveys*, *55*(10). https://doi.org/10.1145/3567550