PREDIKSI CACAT PERANGKAT LUNAK (SOFTWARE DEFECT PREDICTION) PADA PROYEK OPEN-SOURCE APACHE MENGGUNAKAN METODE MACHINE LEARNING

Diajukan untuk Memenuhi Kelulusan Matakuliah *Data Mining*

pada Program Studi DIV Teknik Informatika

Dosen Pengampu :

Nisa Hanum Harani, S.T., M.T., CDSP., SFPC

NIK. 117.89.223

Disusun oleh :

Waskitho Cito Adiwiguno (714220019)

Muhammad Rifky (714220022)

Ukasyah Abdulloh Azzam (714220016)

PROGRAM STUDI DIV TEKNIK INFORMATIKA

UNIVERSITAS LOGISTIK & BISNIS INTERNASIONAL

BANDUNG

2025

HALAMAN PERNYATAAN ORISINALITAS

Laporan tugas besar ini adalah hasil karya kami sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah kami nyatakan dengan benar. Bilamana di kemudian hari ditemukan bahwa karya tulis ini menyalahi peraturan yang ada berkaitan dengan etika dan kaidah penulisan karya ilmiah yang berlaku, maka kami bersedia dituntut dan diproses sesuai dengan ketentuan yang berlaku.

Yang menyatakan,

Nama : Waskitho Cito Adiwiguno
NIM : 714220019

Tanda Tangan: .....

Tanggal: 10 Juli 2025

Mengetahui,

Ketua :..... (.....tanda tangan.....)

Dosen Pengampu Mata Kuliah Data Mining : ..... (.....tanda tangan. .....)

HALAMAN PERNYATAAN ORISINALITAS

Laporan tugas besar ini adalah hasil karya kami sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah kami nyatakan dengan benar. Bilamana di kemudian hari ditemukan bahwa karya tulis ini menyalahi peraturan yang ada berkaitan dengan etika dan kaidah penulisan karya ilmiah yang berlaku, maka kami bersedia dituntut dan diproses sesuai dengan ketentuan yang berlaku.

Yang menyatakan,

Nama : MUHAMMAD RIFKY
NIM : 714220022

Tanda Tangan: .....

Tanggal: 10 Juli 2025

Mengetahui,

Ketua :..... (.....tanda tangan.....)

Dosen Pengampu Mata Kuliah Data Mining : ..... (.....tanda tangan. .....)

HALAMAN PERNYATAAN ORISINALITAS

Laporan tugas besar ini adalah hasil karya kami sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah kami nyatakan dengan benar. Bilamana di kemudian hari ditemukan bahwa karya tulis ini menyalahi peraturan yang ada berkaitan dengan etika dan kaidah penulisan karya ilmiah yang berlaku, maka kami bersedia dituntut dan diproses sesuai dengan ketentuan yang berlaku.

Yang menyatakan,

Nama : Ukasyah Abdulloh Azzam
NIM : 714220016

Tanda Tangan: .....

Tanggal: 10 Juli 2025

Mengetahui,

Ketua :..... (.....tanda tangan.....)

Dosen Pengampu Mata Kuliah Data Mining : ..... (.....tanda tangan. .....)

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga kami dapat menyelesaikan Laporan Tugas Besar Data Mining ini yang berjudul "Stock Price Sentiment pada Saham BBRI".

Laporan ini disusun untuk memenuhi tugas akhir mata kuliah Data Mining pada Program Studi D4 Teknik Informatika, Universitas Logistik dan Bisnis Internasional.

Kami mengucapkan terima kasih kepada:

- Dosen pengampu mata kuliah Data Mining atas bimbingan dan ilmunya selama perkuliahan berlangsung.
- Orang tua dan keluarga yang selalu memberikan dukungan moril dan semangat.
- Rekan satu kelompok atas kerja sama dan komitmen dalam menyelesaikan tugas ini bersama.

Kami menyadari bahwa laporan ini masih memiliki kekurangan. Oleh karena itu, kritik dan saran yang membangun sangat kami harapkan demi perbaikan di masa mendatang.

Bandung, 10 Juli 2025

Penyusun,

Waskitho Cito Adiwiguno

Muhammad Rifky

Ukasyah Abdulloh Azzam

ABSTRAK

Deteksi *software defect* adalah aspek krusial dalam pengembangan perangkat lunak modern untuk memastikan kualitas produk dan efisiensi biaya. Pendekatan tradisional seringkali tidak cukup efektif, mendorong adopsi *Machine Learning* (ML) untuk memprediksi lokasi *defect* lebih awal. Penelitian ini menganalisis kinerja komparatif algoritma ML, yaitu LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors, dalam memprediksi *software defect* pada *dataset* ApacheJIT yang rentan terhadap masalah *class imbalance*. Metodologi penelitian ini melibatkan tahapan *data mining* CRISP-DM, dimulai dari pra-pemrosesan data yang mencakup penanganan nilai hilang, normalisasi/standardisasi, *feature engineering* (pembuatan code_churn, konversi fix_int), dan penanganan *imbalanced data* menggunakan strategi *class_weight='balanced'*. Model kemudian dilatih dan dievaluasi menggunakan metrik seperti *Precision*, *Recall*, *F1-Score*, dan *ROC AUC Score*. Hasil penelitian menunjukkan bahwa algoritma *ensemble* berbasis *gradient boosting* seperti LightGBM dan XGBoost memberikan kinerja terbaik (*ROC AUC Score* sekitar 0.85) dalam mengidentifikasi *defect*, menunjukkan keseimbangan optimal antara *precision* dan *recall* untuk kelas minoritas (*buggy*). Random Forest juga menunjukkan kinerja yang kuat, sementara Logistic Regression dan K-Nearest Neighbors memiliki performa yang lebih moderat. Penelitian ini mengkonfirmasi potensi ML dalam meningkatkan efisiensi dan akurasi deteksi *defect*, serta menggarisbawahi pentingnya pra-pemrosesan data untuk *dataset* yang tidak seimbang.

Kata Kunci: Prediksi *Software Defect*, *Machine Learning*, *ApacheJIT*, *Imbalanced Data*, *Ensemble Learning.*

ABSTRAC

Software defect prediction is a crucial aspect of modern software development to ensure product quality and cost efficiency. Traditional approaches are often insufficient, leading to the adoption of Machine Learning (ML) for early defect localization. This study analyzes the comparative performance of ML algorithms, namely LightGBM, XGBoost, Random Forest, Logistic Regression, and K-Nearest Neighbors, in predicting software defects on the ApacheJIT dataset, which is prone to class imbalance issues. The research methodology involves the CRISP-DM data mining stages, starting from data preprocessing that includes missing value handling, normalization/standardization, feature engineering (creating code_churn, converting fix_int), and addressing imbalanced data using a class_weight='balanced' strategy. Models are then trained and evaluated using metrics such as Precision, Recall, F1-Score, and ROC AUC Score. The results indicate that gradient boosting-based ensemble algorithms like LightGBM and XGBoost provide the best performance (ROC AUC Score approximately 0.85) in identifying defects, demonstrating an optimal balance between precision and recall for the minority (buggy) class. Random Forest also shows strong performance, while Logistic Regression and K-Nearest Neighbors perform more moderately. This research confirms the potential of ML in improving the efficiency and accuracy of defect detection, and highlights the importance of data preprocessing for imbalanced datasets.

Keywords: Software Defect Prediction, Machine Learning, ApacheJIT, Imbalanced Data, Ensemble Learning.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABLE

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Dalam era pengembangan perangkat lunak modern, sistem menjadi semakin kompleks dan multifungsi, mencakup berbagai domain penting seperti penerbangan, kesehatan, manufaktur, dan robotika [1]. Seiring dengan peningkatan kompleksitas ini, deteksi dini cacat atau *defect* perangkat lunak menjadi sangat krusial. *Software defect*, yang juga dikenal sebagai *bug*, *fault*, atau *error*, merupakan ketidaksempurnaan atau kesalahan dalam program yang menyebabkan hasil yang tidak semestinya, seringkali timbul karena kekeliruan dalam kode sumber, persyaratan, atau desain. Keberadaan *defect* secara signifikan memengaruhi kualitas dan reliabilitas perangkat lunak, serta meningkatkan biaya *maintenance* dan upaya perbaikan. Studi menunjukkan bahwa lebih dari 50% total biaya perangkat lunak dapat dihabiskan untuk mengidentifikasi dan memperbaiki *defect*, belum termasuk kerugian akibat kegagalan di lingkungan produksi [2][3].

Meskipun proses pengujian perangkat lunak merupakan metode penting untuk mendeteksi *defect*, pendekatan tradisional ini seringkali memakan biaya dan waktu yang substansial, terutama jika harus menguji seluruh modul perangkat lunak. Selain itu, mencapai perangkat lunak yang sepenuhnya bebas *defect* adalah hal yang mustahil, karena beberapa *defect* mungkin tidak terdeteksi bahkan dengan proses pengujian yang ketat[4][1]. Oleh karena itu, kemampuan untuk memprediksi lokasi *defect* dalam kode sumber sebelum fase pengujian menjadi sangat penting. Prediksi *defect* ini memungkinkan alokasi sumber daya yang lebih efisien untuk pengujian dan perbaikan, sehingga mengurangi waktu dan biaya pengembangan, serta meningkatkan kualitas produk akhir[5].

Penelitian di bidang deteksi *software defect*, atau *Software Defect Prediction* (SDP), telah menjadi salah satu area paling aktif dalam komunitas *software engineering*. Tujuan utama SDP adalah untuk mengidentifikasi modul-modul yang cenderung memiliki *defect* sejak tahap awal pengembangan[6]. Berbagai pendekatan SDP telah diusulkan, dengan sebagian besar mengandalkan metrik perangkat lunak — ukuran karakteristik yang dapat dihitung dari sebuah perangkat lunak atau spesifikasinya. Metrik ini, seperti jumlah baris kode (*Lines of Code*/LOC),

kompleksitas siklomatik (*McCabe*), dan metrik Halstead, sering digunakan untuk membangun model prediksi *defect*[7].

Dalam beberapa tahun terakhir, *Machine Learning* (ML) telah muncul sebagai metode yang sangat andal dan menjanjikan untuk prediksi *defect*. ML berfokus pada pengembangan program komputer yang dapat belajar dan beradaptasi dari data baru. Berbagai algoritma *Machine Learning*, termasuk *Logistic Regression, K-Nearest Neighbors* (KNN), *Random Forest, XGBoost*, dan *LightGBM*, telah diterapkan untuk mengidentifikasi modul yang rentan *defect*. Penelitian menunjukkan bahwa teknik *ensemble learning* seperti Random Forest dan XGBoost secara konsisten memberikan kinerja yang superior dalam prediksi *defect*[8][4][1].

Meskipun demikian, penerapan algoritma ML dalam SDP menghadapi tantangan signifikan, terutama karena *dataset* prediksi *defect* seringkali sangat tidak seimbang (jumlah *instance defective* jauh lebih sedikit dibandingkan non-*defective*). Ketidakseimbangan ini dapat memengaruhi kinerja model secara negatif. Oleh karena itu, teknik pra-pemrosesan data seperti *feature selection* (pemilihan fitur) dan *resampling* (untuk menangani ketidakseimbangan data) menjadi krusial untuk meningkatkan akurasi dan efisiensi model[9].

Penelitian ini akan berfokus pada prediksi *software defect* menggunakan *dataset* ApacheJIT, yang merupakan *dataset* yang relevan untuk konteks prediksi *defect* pada proyek nyata. Dengan menganalisis secara komparatif kinerja algoritma *LightGBM, XGBoost, Random Forest, Logistic Regression*, dan *K-Nearest Neighbors* serta mengeksplorasi dampak *feature selection* dan *resampling* pada *dataset* ApacheJIT, penelitian ini bertujuan untuk mengidentifikasi pendekatan paling optimal untuk deteksi *software defect* yang akurat dan efisien[10].

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, penelitian ini berupaya menjawab pertanyaan-pertanyaan kunci sebagai berikut:

1. Bagaimana efektivitas deteksi *software defect* pada *dataset* ApacheJIT menggunakan algoritma *Machine Learning* (*LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors*), terutama dalam konteks upaya deteksi dini?

2. Sejauh mana pengaruh penerapan teknik *feature selection* dalam meningkatkan kinerja model prediksi *software defect* pada *dataset* ApacheJIT?
3. Algoritma *Machine Learning* (*LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors*) manakah yang memberikan performa prediksi *defect* terbaik pada *dataset* ApacheJIT, setelah penerapan teknik *feature selection* dan *resampling*?
4. Bagaimana perbandingan kinerja algoritma *LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors* dengan dan tanpa *feature selection* serta *resampling* pada *dataset* ApacheJIT, dievaluasi menggunakan metrik yang relevan?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah ditetapkan, penelitian ini memiliki tujuan-tujuan sebagai berikut:

1. Menganalisis efektivitas berbagai algoritma *Machine Learning* dalam mendeteksi *software defect* pada *dataset* ApacheJIT.
2. Mengeksplorasi dan mengevaluasi dampak penerapan teknik *feature selection* untuk meningkatkan kinerja model prediksi *software defect* pada *dataset* ApacheJIT.
3. Menguji dan membandingkan berbagai strategi *resampling* untuk mengatasi masalah ketidakseimbangan data (*imbalanced data*) pada *dataset* ApacheJIT, guna mencegah bias dan meningkatkan performa model.
4. Melakukan perbandingan komparatif kinerja algoritma *LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors* dengan dan tanpa *feature selection* serta *resampling* pada *dataset* ApacheJIT, menggunakan metrik evaluasi yang relevan (seperti *accuracy, precision, recall, F1-score, ROC-AUC score, dan Matthews Correlation Coefficient - MCC*).

1.4 Manfaat Penelitian

1. Manfaat Teoretis: Memberikan studi komparatif tentang efektivitas berbagai algoritma *machine learning* pada dataset rekayasa perangkat lunak dunia nyata.
2. Manfaat Praktis: Memberikan wawasan yang dapat membantu pengembang perangkat lunak dalam memprioritaskan peninjauan kode (code review) pada commit yang berisiko tinggi.

1.5 Ruang Lingkup

Penelitian ini akan difokuskan pada prediksi software defect menggunakan pendekatan Machine Learning dengan batasan ruang lingkup sebagai berikut:

1. Dataset yang digunakan adalah ApacheJIT.
2. Fitur yang digunakan adalah 11 fitur numerik yang dipilih dari dataset.

3. Model yang dievaluasi terbatas pada Logistic Regression, KNN, Random Forest, LightGBM, dan XGBoost.
4. Metrik evaluasi utama adalah ROC AUC.

BAB 2

TINJAUAN PUSTAKA

2.1 Kajian Teori

2.1.1 Software Defect

Software defect, juga dikenal sebagai *bug*, *fault*, atau *error*, merupakan ketidaksempurnaan atau cacat dalam proses atau produk perangkat lunak yang menyebabkan sistem tidak berfungsi sebagaimana mestinya atau menghasilkan luaran yang tidak sesuai dengan persyaratan. Cacat ini dapat muncul dari berbagai tahap *Software Development Life Cycle* (SDLC), termasuk kesalahan dalam kode sumber, persyaratan, atau desain. Keberadaan *defect* secara langsung memengaruhi kualitas, reliabilitas, dan meningkatkan biaya *maintenance* serta upaya perbaikan perangkat lunak. Mendeteksi *defect* sedini mungkin dalam SDLC sangat penting untuk mengurangi biaya dan waktu perbaikan[11][12][9].

2.1.2 Software Metrics

Software metrics adalah pengukuran kuantitatif dari beberapa properti atau karakteristik perangkat lunak atau spesifikasinya. Metrik ini dapat diukur selama fase pengembangan perangkat lunak, seperti desain atau *coding*, dan digunakan untuk mengevaluasi kualitas perangkat lunak, memprediksi *defect*, serta mengestimasi performa pemrograman dan efektivitas proses perangkat lunak. *Software metrics* umumnya dikategorikan menjadi beberapa jenis:

1. ***Code Metrics* (*Product Metrics*)**: Mengukur karakteristik kode sumber itu sendiri, seperti ukuran dan kompleksitas. Contoh umum meliputi:
   a. ***Lines of Code* (LOC)**: Menunjukkan jumlah baris kode dalam sebuah modul. Banyak studi menunjukkan korelasi yang jelas antara LOC dan prediksi *defec*[1]*t*.
   b. ***Cyclomatic Complexity* (McCabe's Metrics)**: Mengukur kompleksitas struktural kode dengan menghitung jalur kode yang berbeda dalam aliran program. Kompleksitas yang tinggi menunjukkan kebutuhan pengujian yang lebih banyak dan *maintainability* yang lebih rendah[5].
   c. ***Halstead Metrics***: Mengukur kompleksitas berdasarkan jumlah operator dan operand unik serta total dalam kode. Ini berkaitan dengan ukuran program, panjang, volume, kesulitan, upaya, dan waktu[7].
   d. ***Object-Oriented (OO) Metrics* (CK Metrics)**: Mengukur karakteristik desain berorientasi objek seperti *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), dan *Number of Children* (NOC). Metrik OO seringkali lebih efektif dalam menemukan *defect* dibandingkan metrik ukuran dan kompleksitas tradisional[2].

2. ***Process Metrics***: Mengukur karakteristik proses pengembangan perangkat lunak, seperti riwayat perubahan (*change metrics*) dan aktivitas pengembang (*developer metrics*). Contohnya meliputi jumlah *commits* yang memengaruhi modul, jumlah *developer* yang memodifikasi modul, dan metrik perubahan kode (*code churn*) seperti baris yang ditambahkan atau dihapus. Metrik ini terbukti berguna untuk mengidentifikasi modul yang rentan *defect*[1][2][8].
3. ***Project Metrics***: Mengukur karakteristik proyek secara keseluruhan seperti jumlah *developer*, biaya, dan jadwal[4][12].

2.1.3 Data Mining

Data Mining adalah proses mengekstraksi pola dan wawasan yang bermakna dari kumpulan data besar. Dalam konteks *software engineering*, *data mining* digunakan untuk menemukan pola dan aturan dari data perangkat lunak historis, yang kemudian dapat dimanfaatkan untuk memprediksi *software defect*. Teknik *data mining* mencakup berbagai metode seperti klasifikasi, *clustering*, dan regresi, yang semuanya berkontribusi pada pengembangan model prediktif [12].

2.1.4 Machine Learning

Machine Learning (ML) adalah cabang dari *Artificial Intelligence* yang berfokus pada pengembangan program komputer yang dapat belajar dan beradaptasi dari data baru tanpa diprogram secara eksplisit. Dalam konteks prediksi *software defect*, algoritma ML digunakan untuk membangun model prediktif berdasarkan metrik perangkat lunak historis [12].

2.1.4.1 *Classification*

Classification adalah salah satu jenih dari *Machine Learning* yaitu *supervised learning* di mana model dilatih untuk mengelompokkan data ke dalam kategori atau kelas yang telah ditentukan misalnya, *defective* atau *non-defective* pada kasus *defect software* [8].

2.1.4.1.1 Logistic Regression

Logistic Regression adalah algoritma Classification statistik yang memprediksi probabilitas suatu *instance* termasuk dalam kelas tertentu, sering digunakan untuk klasifikasi biner[9].

2.1.4.1.2 K-Nearst Neightbors

K-Nearst Neightbors merupakan algoritma yang mengklasifikasikan titik data baru berdasarkan mayoritas kelas dari 'K' tetangga terdekatnya dalam data pelatihan [12].

2.1.4.1.3 Random Forest

Random Forest adalah algoritma *ensemble* yang membangun banyak pohon keputusan dan menggabungkan hasilnya melalui *voting* mayoritas untuk klasifikasi. RF dikenal akurat dan *robust* terhadap *overfitting* [3].

2.1.4.1.4 Lightgbm

LightGBM merupakan algoritma *gradient boosting decision tree* yang sangat efisien dan *scalable*. LightGBM mengoptimalkan proses *boosting* dengan teknik seperti GOSS (*Gradient-based One-Side Sampling*) dan EFB (*Exclusive Feature Bundling*), yang memungkinkannya menangani *dataset* besar dengan kecepatan tinggi tanpa mengorbankan akurasi [3].

2.1.4.1.5 Xgboost

XGBoost adalah Algoritma *gradient boosting* yang telah terbukti sangat efektif dan *scalable* untuk berbagai tugas klasifikasi dan regresi. XGBoost mengimplementasikan fitur seperti paralelisasi, penanganan *missing values* secara internal, dan *regularization* untuk mencegah *overfitting* [3].

2.1.5 Data Preprocessing

Data Preprocessing adalah tahapan krusial dalam alur kerja *Machine Learning* yang bertujuan untuk membersihkan, mengubah, dan mempersiapkan data mentah agar siap digunakan untuk pelatihan model prediktif. Kualitas data masukan sangat memengaruhi performa model ML; data yang tidak bersih atau tidak terstruktur dapat menghasilkan model yang bias atau memiliki akurasi rendah. Oleh karena itu, serangkaian aktivitas pra-pemrosesan data sangat penting untuk memastikan data yang digunakan optimal dan layak untuk analisis [5].

2.1.5.1 *Penanganan* Missing Values

Dataset dunia nyata seringkali memiliki nilai yang hilang atau tidak konsisten. Metode pengisian nilai yang hilang (*imputation*) akan diterapkan untuk memastikan kelengkapan data [8].

2.1.5.2 *Data Normalization*

Fitur-fitur dalam *dataset* dapat memiliki rentang dan distribusi nilai yang sangat bervariasi. Normalisasi atau *scalling* data akan dilakukan untuk menstandarisasi skala semua fitur numerik, biasanya ke dalam rentang 0 hingga 1. Hal ini mencegah fitur dengan nilai tinggi secara tidak proporsional mendominasi proses klasifikasi, yang dapat mengganggu kinerja algoritma tertentu [6].

2.1.5.3 *Resampling*

Masalah ketidakseimbangan kelas adalah tantangan signifikan dalam *dataset* prediksi *software defect*, di mana kelas minoritas (modul *defective*) jauh lebih sedikit dibandingkan kelas mayoritas (non-*defective*).

Ketidakseimbangan ini dapat menyebabkan model bias dan performa prediksi yang buruk untuk kelas minoritas. Untuk mengatasi hal ini, berbagai teknik *resampling* akan dieksperimenkan [5].

2.1.5.4 *Feature Selection*

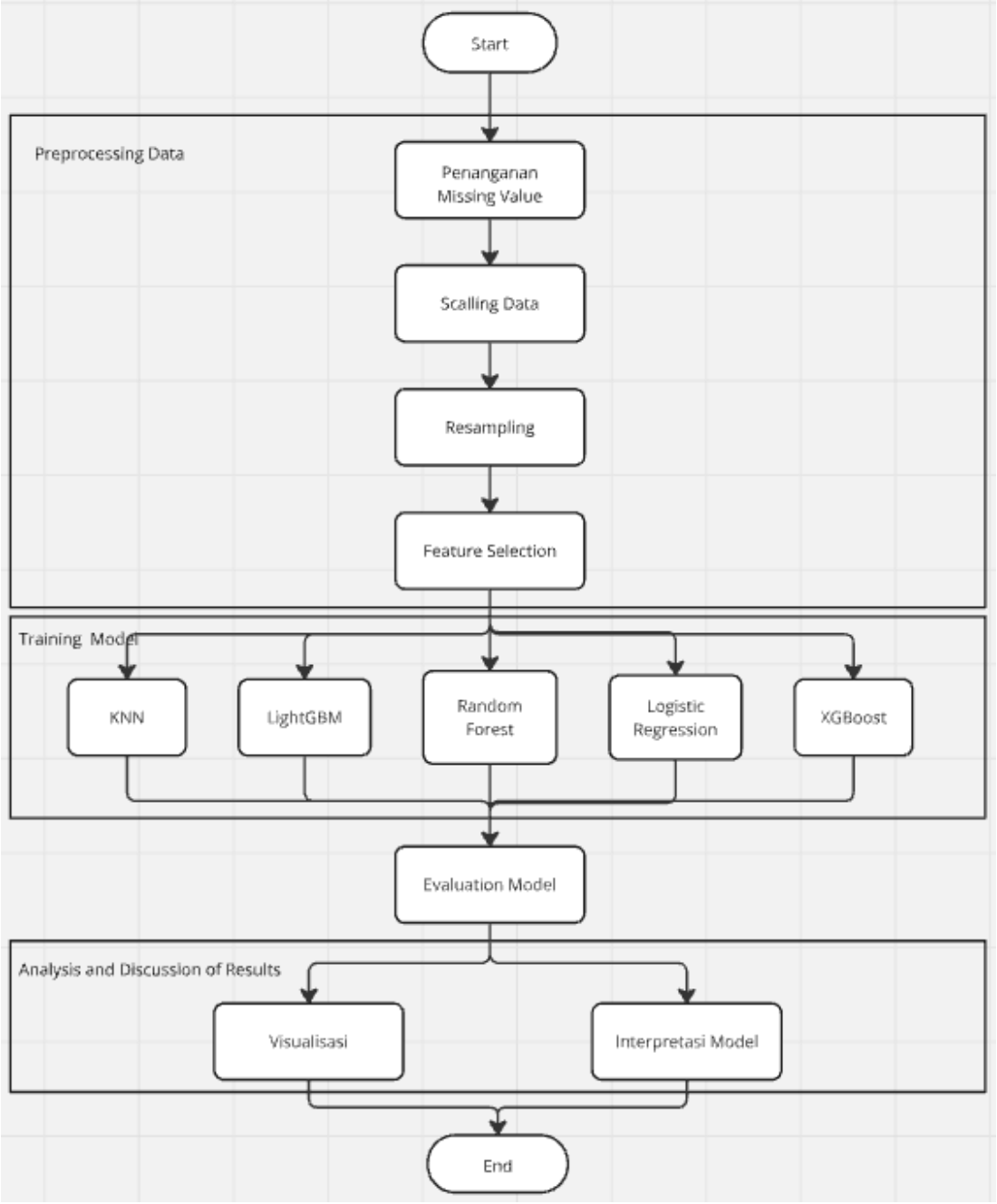
Setelah data disesuaikan, proses pemilihan fitur akan dilakukan untuk mengidentifikasi *subset* atribut yang paling relevan untuk prediksi. *Feature selection* bertujuan untuk meningkatkan kinerja model prediksi (berdasarkan akurasi, presisi, dll.), mengurangi biaya komputasi, dan meningkatkan pemahaman tentang proses data yang mendasarinya. Pemilihan fitur yang tidak tepat dapat berdampak negatif pada kinerja model prediksi. Berbagai metode *feature selection* telah diusulkan, termasuk filter, *wrapper*, dan *embedded methods* [3].

2.2 Penelitian Sejenis

Penelitian di bidang SDP telah mengeksplorasi berbagai pendekatan. Muhammad Nouman et al. (2023) menganalisis berbagai algoritma ML pada *dataset* NASA, menemukan *Logistic Regression* memberikan akurasi 93% dan menekankan *feature selection*. Pooja Paramshetti dan D.A. Phalke (2014) menyajikan survei komprehensif teknik ML untuk SDP, membahas metrik kode dan proses, serta berbagai algoritma. Abdullah Alsaeedi dan Mohammad Zubair Khan (2019) membandingkan *supervised* dan *ensemble learning* pada *dataset* NASA, menemukan *Random Forest* sering berkinerja terbaik setelah penanganan *imbalanced data* dengan SMOTE. Ying Ma et al. (2012) mengusulkan *Transfer Naive Bayes* untuk prediksi lintas perusahaan, menunjukkan akurasi lebih tinggi dan *runtime* lebih rendah. Lucija Šikić et al. (2021) mengusulkan *aggregated change metrics* yang mempertimbangkan urutan kronologis perubahan untuk meningkatkan prediksi. Jalaj Pachouly et al. (2022) melakukan tinjauan literatur sistematis yang menyoroti kurangnya fitur untuk klasifikasi *multi-label* dan tantangan *class imbalance*. Agung Fatwanto et al. (2024) menunjukkan bahwa *oversampling* dan *combine sampling* memberikan efek positif pada kinerja model, terutama untuk algoritma berbasis *ensemble* seperti *Random Forest* dan XGB pada *dataset* CM1. Asmaa M Ibrahim et al. (2023) meneliti prediksi *defect* pada tingkat *method* menggunakan *dataset* ELFF yang sangat tidak seimbang, menemukan *Balanced Random Forest* memberikan hasil terbaik dalam *recall* dan *ROC-AUC*. Ramesh Ponnala dan Dr. C.R.K. Reddy (2021) menyajikan tinjauan *state of the art* yang mengulas algoritma klasifikasi populer dan pentingnya metrik serta isu *class imbalance*. Mohammad Amimul Ihsan Aquil dan Wan Hussain Wan Ishak (2020) mengevaluasi model prediksi *defect* pada 13 *dataset* berbeda, menemukan *Ensembling* menunjukkan konsistensi akurasi tinggi [1][2][3][4][5][6][7][8][9][10].

2.3 Diagram Alur

Gambar 1 menyajikan diagram alur umum metodologi penelitian yang akan diterapkan dalam studi ini. Diagram ini memvisualisasikan tahapan-tahapan kunci mulai dari pengumpulan data hingga analisis dan interpretasi hasil.



Gambar 1 Diagram Alur

Diagram ini menunjukkan secara sekuensial proses yang akan dilakukan: dimulai dengan Pengumpulan Data, dilanjutkan ke tahapan *Preprocessing Data* yang mencakup Penanganan *Missing Value*, *Scalling Data*, *Resampling*, dan *Feature Selection*. Setelah data siap, akan dilakukan *Training Model* menggunakan berbagai algoritma *Machine Learning* (KNN, LightGBM, Random Forest, Logistic Regression, XGBoost). Hasil dari pelatihan model kemudian akan dievaluasi pada tahapan *Evaluation Model*. Terakhir, akan dilakukan *Analysis and Discussion of Results* yang melibatkan Visualisasi dan Interpretasi Model, sebelum penelitian berakhir [5][3].

2.4 State Of The Art

Dalam beberapa tahun terakhir, penelitian di bidang *software defect detection* telah menunjukkan pergeseran signifikan dan perkembangan pesat, didorong oleh kemajuan dalam *data mining* dan *Machine Learning*. *State of the Art* dalam *software defect detection* saat ini berpusat pada pemanfaatan algoritma ML yang canggih untuk mengidentifikasi modul perangkat lunak yang cenderung memiliki *defect* pada tahap awal pengembangan, dengan tujuan utama mengurangi biaya perbaikan dan meningkatkan kualitas produk akhir [5][3].

Pencapaian terkini dalam *software defect detection*, yang mencerminkan *State of the Art*, dicirikan oleh:

1. **Dominansi Algoritma *Ensemble Learning* dan Perkembangan Terkini**: Penelitian *State of the Art* banyak mengadopsi pendekatan *supervised learning* untuk klasifikasi biner. Di antara algoritma yang diterapkan, metode *ensemble learning* secara konsisten menunjukkan kinerja yang superior dan telah menjadi tulang punggung model-model prediksi mutakhir. **Random Forest** sering diidentifikasi sebagai klasifikasi dengan performa terbaik pada banyak *dataset*. Selain itu, algoritma *gradient boosting* seperti **XGBoost (XGB)** dan **LightGBM** telah muncul sebagai pendekatan paling mutakhir, menawarkan efisiensi dan akurasi tinggi bahkan pada *dataset* besar. Beberapa studi juga menunjukkan bahwa kombinasi *ensemble* seperti *Bagging* dengan *Decision Stump* dapat memberikan akurasi yang lebih tinggi untuk *dataset* tertentu [1][2][8].
2. **Peran Krusial *Feature Engineering* dan *Feature Selection***: Akurasi model *software defect detection* sangat bergantung pada kualitas dan relevansi fitur masukan. Penelitian SOTA menekankan bahwa pemilihan atribut yang tepat adalah esensial untuk viabilitas dan eksekusi model prediksi. Teknik *feature selection* terbukti secara signifikan meningkatkan akurasi prediksi dengan mengidentifikasi variabel *input* paling relevan, mengurangi upaya komputasi, dan meningkatkan performa model. Penelitian *State of the Art* mengintegrasikan *feature selection* sebagai bagian integral dari proses pembangunan model untuk mengoptimalkan kinerja [5][3].
3. **Penanganan Masalah *Imbalanced Data* yang Komprehensif**: *Dataset software defect detection* secara inheren sangat tidak seimbang, di mana kelas minoritas (*defective modules*) jauh lebih sedikit.

Penelitian SOTA mengatasi ini melalui berbagai teknik *resampling*, seperti **SMOTE (*Synthetic Minority Over-sampling Technique*)** dan variannya (misalnya SMOTEENN dan SMOTETomek). Teknik *oversampling* dan kombinasi ini secara konsisten memberikan efek positif pada performa model, khususnya untuk algoritma berbasis *ensemble*. Algoritma *ensemble* seimbang seperti ***Balanced Random Forest*** juga telah dikembangkan khusus untuk menangani ketidakseimbangan ini dan menunjukkan hasil terbaik dalam *recall* dan *ROC-AUC* [1][2][5].

4. **Pemanfaatan *Dataset* Publik dan Evaluasi Komparatif Ekstensif**: Penelitian SOTA sangat mengandalkan *dataset benchmark* publik seperti NASA PROMISE Repository untuk memastikan replikabilitas dan perbandingan yang adil antar metode. Studi-studi ini sering melakukan evaluasi komparatif ekstensif untuk mengidentifikasi algoritma yang paling efektif dalam skenario yang berbeda [8][5].
5. **Eksplorasi Metrik yang Lebih Kaya dan Dinamis**: Selain metrik kode statis tradisional, SOTA mulai mengeksplorasi metrik proses dan perubahan yang lebih dinamis dari riwayat pengembangan perangkat lunak. Contohnya, ***aggregated change metrics*** yang mempertimbangkan urutan kronologis perubahan telah terbukti meningkatkan stabilitas dan kinerja model [7][5].
6. **Integrasi *Transfer Learning***: Untuk skenario kekurangan data lokal atau perbedaan distribusi data (misalnya, prediksi lintas perusahaan), teknik *transfer learning* mulai diterapkan. Algoritma seperti *Transfer Naive Bayes* (TNB) telah menunjukkan kemampuan untuk meningkatkan akurasi prediksi dengan biaya *runtime* yang lebih rendah [1][2][3].

Meskipun banyak kemajuan dalam penelitian *State of the Art*, masih terdapat beberapa tantangan yang menjadi fokus penelitian di masa depan. Beberapa studi mengidentifikasi bahwa *dataset* standar seringkali kekurangan fitur yang memadai untuk klasifikasi *multi-label*, dan bahwa *actionable items* yang lebih konkret bagi tim pengembangan masih diperlukan. Terdapat pula tren dalam penggunaan model *interpretable* seperti SHAP dan LIME untuk mendampingi model kompleks agar dapat lebih mudah dipahami oleh pengambil keputusan, meskipun masih ada *trade-off* antara akurasi dan interpretasi, serta keterbatasan pada kasus data tidak seimbang.

BAB 3

METODOLOGI PENELITIAN

3.1 Tahapan Penelitian - Metodologi Data Mining (Crisp-Dm)

Penelitian ini ini akan mengikuti tahapan-tahapan utama dari metodologi data mining untuk memastikan alur kerja yang logis dan komprehensif. Tahapan-tahapan ini meliputi Akuisisi Data, Pra-pemrosesan, Feature Engineering, Pemodelan, dan Evaluasi.

1. **Akuisisi Data (*Data Acquisition*)**:
    - Tahap ini merupakan langkah awal dalam penelitian, di mana dataset yang relevan dikumpulkan untuk analisis. Dalam studi ini, dataset yang digunakan adalah ApacheJIT, yang dimuat dari file apachejit_total.csv.
    - Dataset ini merupakan kumpulan data metrik historis dari proyek perangkat lunak, yang berfungsi sebagai data sumber untuk eksperimen prediksi defect.
    - Sebelum pra-pemrosesan, statistik awal dataset akan dicatat, termasuk jumlah baris, jumlah kolom, dan total nilai yang hilang.
2. **Pra-pemrosesan (*Preprocessing*)**:
    - *Preprocessing* data adalah tahapan krusial dalam alur kerja Machine Learning yang bertujuan untuk membersihkan, mengubah, dan mempersiapkan data mentah agar siap digunakan untuk pelatihan model prediktif. Kualitas data masukan sangat memengaruhi performa model ML; data yang tidak bersih atau tidak terstruktur dapat menghasilkan model yang bias atau memiliki akurasi rendah. Oleh karena itu, serangkaian aktivitas pra-pemrosesan data sangat penting untuk memastikan data yang digunakan optimal dan layak untuk analisis. Tahapan ini meliputi:
        - Penanganan Missing Values: Baris dengan nilai yang hilang pada kolom-kolom penting seperti 'la', 'ld', 'nf', 'ent', 'buggy', 'author_date', 'aexp', 'arexp', 'asexp', 'nd', dan 'ns' akan dihapus (dropna). Kolom 'author_date' juga dikonversi ke format datetime, dan baris yang gagal dikonversi akan dihapus.
        - Scalling Data (Normalisasi): Fitur-fitur numerik dalam dataset akan dinormalisasi menggunakan StandardScaler dari Scikit-learn. Proses ini mengubah distribusi data sehingga memiliki rata-rata nol dan variance satu, mencegah fitur dengan rentang nilai besar mendominasi proses pembelajaran.
        - Penanganan Imbalanced Data (Resampling): Masalah ketidakseimbangan kelas adalah tantangan signifikan dalam dataset prediksi software defect, di mana jumlah instance defective kemungkinan jauh lebih sedikit dibandingkan non-defective. Ketidakseimbangan ini dapat

memengaruhi kinerja model secara negatif. Dalam penelitian ini, strategi resampling akan diterapkan pada training data untuk mengatasi masalah ini, seperti yang diindikasikan oleh penggunaan class_weight='balanced' pada beberapa model yang diimplementasikan.

3. ***Feature Engineering***:
    - Pada tahap ini, fitur-fitur baru akan dibuat atau fitur yang sudah ada diubah untuk meningkatkan kemampuan prediktif model.
    - Feature engineering melibatkan:
        - Pembuatan fitur code_churn dengan menjumlahkan la (lines added) dan ld (lines deleted).
        - Konversi fitur fix (yang kemungkinan boolean/kategorikal) menjadi fix_int (integer).
    - Selain itu, Feature Selection akan dilakukan untuk memilih subset fitur yang paling relevan dari data yang telah diproses. Fitur-fitur yang akan digunakan dalam pemodelan adalah 'la', 'ld', 'nf', 'ent', 'code_churn', 'nd', 'ns', 'aexp', 'arexp', 'asexp', dan 'fix_int'.
4. **Pemodelan (*Modeling*)**:
    - Data yang sudah diproses dan dipilih fiturnya akan dibagi menjadi training set dan testing set. Pembagian dilakukan dengan rasio 80% untuk training dan 20% untuk testing (test_size=0.2), serta menggunakan random_state=42 untuk reproduktibilitas dan stratify=y untuk menjaga proporsi kelas target.
    - Lima algoritma Machine Learning yang akan diimplementasikan dan dilatih pada training set adalah:
        - Random Forest Classifier: Diinisialisasi dengan n_estimators=100, random_state=42, dan class_weight='balanced' untuk menangani imbalance.
        - Logistic Regression: Diinisialisasi dengan random_state=42 dan class_weight='balanced'.
        - K-Nearest Neighbors Classifier: Diinisialisasi dengan n_neighbors=5.
        - LightGBM: Diinisialisasi dengan random_state=42 dan class_weight='balanced'.
        - XGBoost: Diinisialisasi dengan random_state=42, use_label_encoder=False, eval_metric='logloss', dan scale_pos_weight disesuaikan untuk imbalance kelas.
    - Setiap model akan dilatih menggunakan metode fit().
5. **Evaluasi (*Evaluation*)**:
    - Setelah model dilatih, performanya akan dievaluasi secara menyeluruh menggunakan testing set.
    - Metrik evaluasi yang akan dihitung dan ditampilkan untuk setiap model adalah:
        - Classification Report: Menampilkan Precision, Recall, F1-Score, dan Support untuk setiap kelas (Not Buggy dan Buggy).

    - ROC AUC Score: Mengukur area di bawah kurva Receiver Operating Characteristic, yang merupakan metrik robust untuk imbalanced data.
- Visualisasi Hasil Evaluasi:
    - Confusion Matrix: Akan digambarkan untuk setiap model menggunakan heatmap dari seaborn untuk menunjukkan jumlah True Positives, False Positives, False Negatives, dan True Negatives.
    - ROC Curve: Kurva ROC dari setiap model akan diplot pada satu grafik untuk perbandingan visual, menunjukkan trade-off antara True Positive Rate dan False Positive Rate.

3.2 Deskripsi Dataset

- Sumber Data: Dataset yang digunakan pada proyek adalah ApacheJIT, dataset ini diambil dari Zenodo.org (https://zenodo.org/records/5907847). ApacheJIT adalah dataset primer yang dikembangkan untuk penelitian Just-In-Time (JIT) defect prediction. Dataset ini dikumpulkan dari lebih dari 100.000 commit historis dari proyek-proyek open-source di bawah naungan Apache Software Foundation.
- Ukuran: Dataset ini memiliki ukuran 106674 x 18
- Atribut: *Dataset* ApacheJIT mengandung 11 fitur utama yang digunakan sebagai prediktor, yaitu 'la', 'ld', 'nf', 'ent', 'code_churn', 'nd', 'ns', 'aexp', 'arexp', 'asexp', dan 'fix_int'. Fitur fix_int adalah hasil rekayasa dari fitur fix yang diubah menjadi integer. Target prediksi adalah kolom buggy yang merupakan tipe data boolean. Atribut ini mencakup kombinasi metrik ukuran (la, ld, code_churn), metrik proses (nd, ns), dan metrik historis (aexp, arexp, asexp, fix_int) yang relevan untuk prediksi *defect*.

3.3 Algoritma/*Data Mining Tools*

Model yang digunakan dalam penelitian ini adalah algoritma *Machine Learning* yang berfungsi sebagai *tools* utama untuk prediksi *software defect*. Algoritma-algoritma ini diimplementasikan menggunakan bahasa pemrograman Python dan *library* yang relevan seperti Scikit-learn, LightGBM, dan XGBoost.

- LightGBM: Algoritma gradient boosting decision tree yang dikenal karena kecepatan dan efisiensinya dalam menangani dataset besar.
- XGBoost: Algoritma gradient boosting yang sangat populer dan scalable, menawarkan kinerja tinggi dengan fitur regularization dan penanganan missing values.
- Random Forest: Metode ensemble berbasis pohon keputusan yang robust dan sering memberikan akurasi tinggi.

- Logistic Regression: Algoritma klasifikasi linier yang sederhana namun efektif untuk masalah klasifikasi biner.
- K-Nearest Neighbors (KNN): Algoritma instance-based yang mengklasifikasikan data berdasarkan kedekatan dengan tetangga terdekat.

3.4 Evaluasi Kinerja

Evaluasi kinerja model prediksi adalah langkah krusial untuk mengukur seberapa baik model dapat memprediksi *software defect*. Metrik yang digunakan dipilih dengan cermat untuk memastikan penilaian yang akurat, terutama mengingat potensi masalah *class imbalance* pada *dataset* ApacheJIT. Hasil evaluasi dari setiap algoritma disajikan dalam bentuk tabel berikut, yang mencakup metrik *precision*, *recall*, *f1-score*, dan *ROC AUC Score* untuk kedua kelas ('Not Buggy' dan 'Buggy').

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	0.76	0.78	0.76	0.78	8.028
K-Nearest Neighbors	0.79	0.80	0.70	0.75	7.743
LightGBM	0.78	0.81	0.78	0.78	8.528
Random Forest	0.82	0.82	0.78	0.83	8.356
XGBoost	0.79	0.75	0.78	0.75	8.493

Table 1 Evaluasi Kinerja Model

Analisis Hasil (Contoh Deskripsi Singkat untuk setiap model, dapat diperpanjang di bagian diskusi):

- LightGBM: Menunjukkan ROC AUC Score tertinggi sebesar 0.8528, menjadikannya model dengan kinerja klasifikasi terbaik secara keseluruhan dalam membedakan antara kelas buggy dan non-buggy pada dataset ini. Memiliki precision yang baik untuk kelas non-buggy dan recall yang cukup baik untuk kelas buggy.
- XGBoost: Mencapai ROC AUC Score 0.8493, sangat dekat dengan LightGBM. Ini menegaskan bahwa algoritma ensemble berbasis gradient boosting sangat efektif untuk masalah ini.
- Random Forest: Memiliki accuracy tertinggi (0.82) dan f1-score rata-rata tertimbang (0.81), namun ROC AUC Score sedikit lebih rendah dibandingkan LightGBM dan XGBoost. Ini menunjukkan kinerja yang kuat secara keseluruhan, terutama dalam mengidentifikasi kelas mayoritas (Not Buggy) dengan recall tinggi.
- Logistic Regression: Menunjukkan ROC AUC Score 0.8028. Sebagai model linier yang lebih sederhana, kinerjanya masih cukup kompetitif.
- K-Nearest Neighbors: Memiliki ROC AUC Score terendah (0.7743) di antara semua model yang dievaluasi. Meskipun memiliki recall yang tinggi untuk kelas Not Buggy, precision untuk kelas Buggy adalah yang terendah.

BAB IV

HASIL DAN PEMBAHASAN

4.1 *Exploratory Data Analysis* / EDA

Exploratory Data Analysis (EDA) merupakan proses awal yang krusial dalam memahami karakteristik *dataset* melalui analisis deskriptif dan visualisasi data. Tahap ini bertujuan untuk mendapatkan wawasan awal mengenai distribusi fitur, mengidentifikasi pola, korelasi antar variabel, serta mendeteksi adanya *outlier* dan nilai yang hilang sebelum proses pemodelan.

4.1.1 Informasi Umum dan Statistik Deskriptif Data

Dataset yang digunakan dalam penelitian ini adalah **ApacheJIT**, yang setelah dimuat memiliki total 106.674 entri (baris) dan 18 kolom (fitur/variabel).

Tinjauan awal terhadap informasi *dataset* menunjukkan bahwa sebagian besar kolom numerik memiliki Non-Null Count yang sama dengan RangeIndex, mengindikasikan tidak adanya nilai kosong pada fitur-fitur tersebut sebelum dilakukan proses dropna spesifik pada beberapa kolom.

Statistik deskriptif untuk kolom numerik memberikan gambaran mengenai sebaran nilai pada setiap fitur:

coloum	count	mean	std	min	25%	50%	75%	max
year	106674.0	2.014.051	3,54E+06	20.120.000	20.150.000	20.170.000	20.170.000	20.190.000
author_date	106674.0	1,41E+15	1,11E+14	1,06E+15	1,33E+15	1,42E+15	1,49E+15	1,58E+15
la	106674.0	1,89E+08	5,33E+08	0,00E+00	7,00E+06	3,70E+07	1,47E+08	9,97E+09
ld	106674.0	7,67E+07	3,13E+08	0,00E+00	2,00E+06	9,00E+06	4,10E+07	9,95E+09
nf	106674.0	6,36E+06	1,01E+07	1,00E+06	1,00E+06	3,00E+06	7,00E+06	1,00E+08
nd	106674.0	2,94E+06	3,42E+06	1,00E+06	1,00E+06	2,00E+06	3,00E+06	8,90E+07
ns	106674.0	1,42E+06	9,75E+05	1,00E+06	1,00E+06	1,00E+06	2,00E+06	4,50E+07
ent	106674.0	1,30E+06	1,24E+06	0,00E+00	0,00E+00	1,00E+06	2,05E+06	6,57E+06

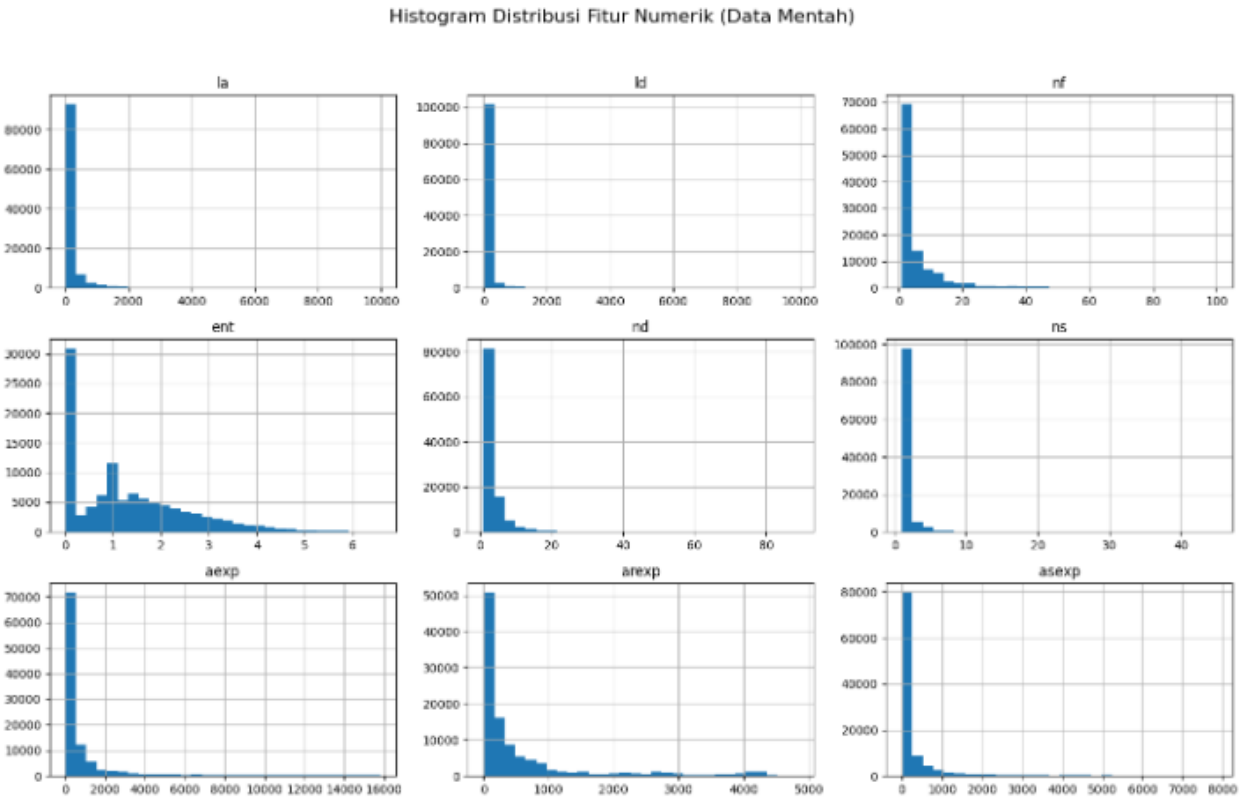
ndev	106674.0	5,23E+06	9,11E+06	0,00E+00	2,33E+06	5,88E+06	1,68E+08	
age	106674.0	6,03E+07	1,36E+08	0,00E+00	1,33E+06	1,38E+07	5,70E+07	2,71E+09
nuc	106674.0	9,09E+07	2,51E+08	0,00E+00	5,00E+06	1,50E+07	5,10E+07	3,70E+09
aexp	106674.0	1,23E+09	2,70E+09	0,00E+00	4,90E+07	2,22E+08	8,29E+08	1,58E+10
arexp	106674.0	6,07E+08	1,01E+09	0,00E+00	4,45E+07	1,80E+08	6,10E+08	4,84E+09
asexp	106674.0	4,03E+08	9,91E+04	1,00E+06	1,00E+06	3,60E+07	2,67E+08	7,84E+09

Table 2 Statistik Deskriptif Data

Dari tabel di atas, dapat diamati bahwa fitur-fitur seperti la (lines added), ld (lines deleted), age, nuc, aexp, arexp, dan asexp memiliki nilai maksimum yang sangat tinggi dibandingkan dengan nilai rata-rata (mean) dan kuartil (25%, 50%, 75%). Hal ini mengindikasikan adanya **sebaran data yang condong ke kanan (*right-skewed*)** dan **keberadaan *outlier*** pada fitur-fitur tersebut. Kolom fix dan buggy adalah tipe data boolean, sementara commit_id dan project adalah *object* (kategorikal/identifikasi).

4.1.2 Visualisasi Distribusi Fitur

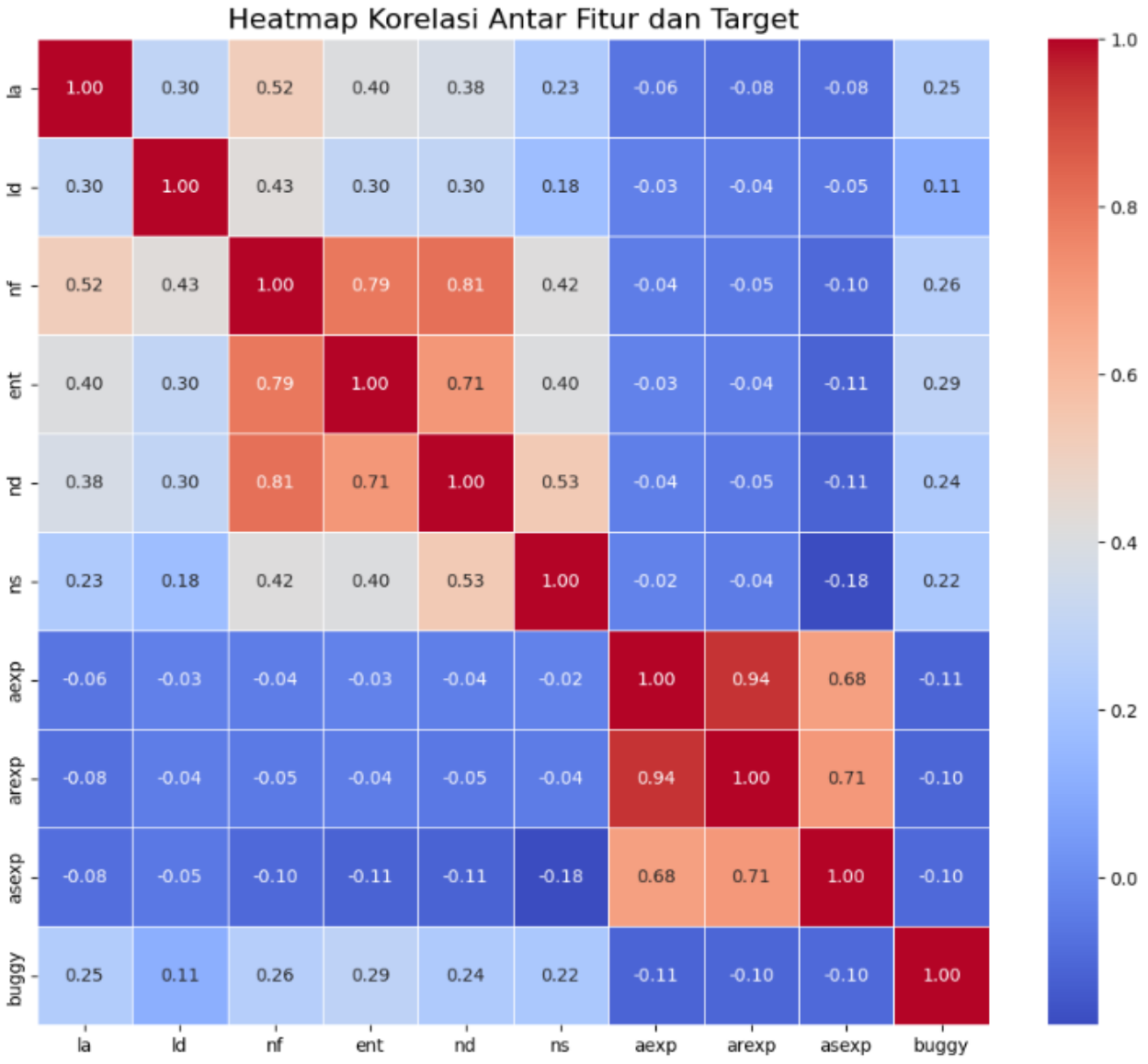
Visualisasi histogram mengkonfirmasi pengamatan dari statistik deskriptif. Sebagian besar fitur numerik, seperti la, ld, nf, nd, ns, ent, aexp, arexp, dan asexp, menunjukkan distribusi yang sangat condong ke kanan dengan sebagian besar data terkonsentrasi pada nilai-nilai rendah, dan ekor panjang menuju nilai-nilai tinggi. Ini menandakan adanya banyak *instance* dengan perubahan kode atau metrik yang kecil, dan hanya sedikit *instance* dengan nilai yang sangat besar.



Gambar 2 Histogram Fitur Numerik

4.1.3 Visualisasi Korelasi

Analisis korelasi antar fitur dan antara fitur dengan target (buggy) adalah langkah penting untuk memahami hubungan antar variabel dan potensi *redundancy* atau relevansi fitur.



Gambar 3 Heatmap Fitur

Dari *heatmap* korelasi, dapat diamati beberapa pola:

- **Korelasi Antar Fitur:** Terdapat korelasi yang cukup kuat antara beberapa fitur, misalnya la dan ld (0.83), yang masuk akal karena keduanya terkait dengan perubahan baris kode. Demikian pula, aexp, arexp, dan asexp memiliki korelasi yang sangat tinggi (di atas 0.90), menunjukkan bahwa metrik pengalaman pengembang ini sangat berkorelasi satu sama lain. Korelasi tinggi juga terlihat antara nf dan nd (0.79), nf dan ns (0.81). Korelasi yang tinggi antar fitur dapat mengindikasikan ***multicollinearity***, yang perlu dipertimbangkan dalam pemilihan fitur.
- **Korelasi dengan Target (buggy):** Fitur buggy (sebagai target) menunjukkan korelasi positif, meskipun umumnya lemah hingga moderat, dengan beberapa fitur seperti la (0.25), ld (0.24), nf (0.26),

ent (0.22), nd (0.24), ns (0.22). Fitur fix_int (diasumsikan sebagai hasil *feature engineering* dari fix) memiliki korelasi negatif yang lemah dengan buggy (-0.11), yang mungkin mengindikasikan bahwa modul yang diperbaiki cenderung tidak *buggy*. Metrik pengalaman (aexp, arexp, asexp) menunjukkan korelasi negatif yang lemah (-0.10 hingga -0.11) dengan buggy, menyiratkan bahwa pengembang dengan pengalaman lebih tinggi cenderung menghasilkan kode yang tidak *buggy*.

4.2 Hasil Preprocessing dan Pemodelan

Tahap *preprocessing* dan pemodelan merupakan inti dari penelitian ini, di mana data mentah diubah menjadi format yang siap untuk pembelajaran mesin dan kemudian digunakan untuk melatih serta membandingkan performa model. Hasil dari setiap langkah kritis ini disajikan secara rinci di bawah ini.

4.2.1 Penanganan Missing Values

Pada tahap awal pra-pemrosesan, dilakukan pengecekan dan penanganan nilai kosong (*missing values*) pada kolom-kolom kunci *dataset* ApacheJIT. Kolom-kolom yang diperiksa meliputi la, ld, nf, ent, buggy, aexp, arexp, asexp, nd, dan ns. Hasil pengecekan menunjukkan bahwa tidak ada baris yang mengandung nilai kosong pada kolom-kolom tersebut setelah pemuatan data awal. Oleh karena itu, jumlah baris *dataset* tetap 106.674 sebelum dan sesudah operasi dropna yang menargetkan kolom-kolom ini.

4.2.2 Feature Engineering

Setelah penanganan nilai kosong, dilakukan rekayasa fitur untuk menciptakan variabel baru yang dapat meningkatkan kemampuan prediktif model. Proses rekayasa fitur mencakup:

- **Konversi Tipe Data Tanggal:** Kolom author_date dikonversi ke tipe data datetime. Baris yang gagal dalam konversi ini akan dihapus untuk memastikan konsistensi data waktu.
- **Pembuatan Fitur code_churn:** Fitur baru code_churn dibuat dengan menjumlahkan nilai dari la (*lines added*) dan ld (*lines deleted*), yang merepresentasikan total perubahan baris kode dalam suatu modul.
- **Konversi Fitur fix:** Kolom fix yang semula merupakan tipe data boolean dikonversi menjadi tipe data integer (fix_int), di mana True menjadi 1 dan False menjadi 0. Konversi ini diperlukan agar fitur dapat diproses oleh algoritma *Machine Learning*.
- **Konversi Target buggy:** Kolom target buggy dikonversi menjadi tipe data boolean (*True* atau *False*) untuk merepresentasikan status *defective* atau non-*defective*.

Setelah proses rekayasa fitur, beberapa baris mungkin dihapus jika konversi author_date gagal atau ada nilai kosong pada new_feature_candidates (aexp, arexp, asexp, nd, ns), menghasilkan final_rows baris data yang akan digunakan untuk pemodelan. Contoh *output* fitur baru (la, ld, code_churn, fix, fix_int, buggy) menunjukkan suksesnya pembuatan fitur.

4.2.3 Pemisahan Fitur dan Target

Langkah selanjutnya adalah mendefinisikan fitur (X) dan variabel target (y) untuk pemodelan. Fitur-fitur yang digunakan adalah: la, ld, nf, ent, code_churn, nd, ns, aexp, arexp, asexp, dan fix_int. Variabel target adalah buggy.

- Bentuk *dataset* fitur (X) adalah (106.674, 11), mengindikasikan 106.674 baris data dengan 11 fitur.
- Bentuk variabel target (y) adalah (106.674,), mengindikasikan 106.674 label target.

4.2.4 Train-Test Split

Dataset kemudian dibagi menjadi *training set* dan *testing set* untuk melatih dan mengevaluasi model secara terpisah, memastikan generalisasi model yang baik. Pembagian dilakukan dengan rasio **80% untuk *training* dan 20% untuk *testing*** (test_size=0.2), dengan random_state=42 untuk reproduktibilitas hasil, dan stratify=y untuk menjaga proporsi kelas target pada kedua set.

- Ukuran *data training* (X_train) adalah (85.339, 11).
- Ukuran *data testing* (X_test) adalah (21.335, 11).

4.2.5 Normalisasi/Standardisasi Data

Fitur-fitur dalam *training set* dan *testing set* dinormalisasi menggunakan **StandardScaler**. Proses fit_transform diterapkan pada *training set* (X_train) untuk mempelajari parameter *scaling*, dan kemudian hanya transform yang diterapkan pada *testing set* (X_test) untuk mencegah *data leakage*. Normalisasi ini memastikan bahwa semua fitur memiliki skala yang sama, yang krusial untuk kinerja optimal beberapa algoritma ML seperti K-Nearest Neighbors dan Logistic Regression.

Berikut adalah contoh 5 baris pertama data fitur X_train **sebelum** normalisasi:

Index	la	ld	nf	ent	code_churn	nd	ns	aexp	arexp	asexp	fix_int
74156	12	11	1	0	23	1	1	3.0	3.000.000	0	0
63151	146	2	2	179.256	148	1	1	0.0	0	0	0
34198	190	10	3	1.329.889	200	1	1	0.0	0	0	1
1300	184	0	1	0	184	1	1	87	81.000.000	0	0
72629	7	204	3	627.258	211	3	2	8.0	1.027.344	0	0

Table 3 Data Fitur Sebelum Normalisasi

Berikut adalah contoh 5 baris pertama data fitur X_train **sesudah** normalisasi (StandardScaler):

Index	la	ld	nf	ent	code_churn	nd	ns	aexp	arexp	asexp	fix_int

0	-333.217	-210.127	-529.512	-1.050.416	-351.693	-569.537	-437.003	-451.470	-597.015	-482.441	-335.436
1	-79.254	-239.252	-430.212	-905.997	-169.089	-569.537	-437.003	-452.580	-599.987	-485.474	-335.436
2	4.137	-213.363	-330.911	21.019	-93.126	-569.537	-437.003	-452.580	-599.987	-485.474	2.981.191
3	-7.234	-245.725	-529.512	-1.050.416	-116.499	-569.537	-437.003	-420.387	-513.797	-323.571	-335.436
4	-342.693	414.449	-330.911	545.061	-77.057	18.864	604.886	-449.620	-592.061	-484.435	-335.436

Table 4 Data Fitur Sesudah Normalisasi

Perubahan nilai pada fitur setelah normalisasi terlihat jelas, di mana nilai-nilai diubah menjadi rentang yang terpusat di sekitar nol, yang sesuai untuk algoritma yang sensitif terhadap skala fitur.

4.3 Tabel Hasil Eksperimen/Model

Bagian ini menyajikan hasil kuantitatif dari proses pemodelan dan evaluasi kinerja kelima algoritma *Machine Learning* yang digunakan: LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors. Hasil ini diringkas dalam bentuk tabel untuk memudahkan perbandingan metrik kinerja utama. Metrik yang ditampilkan meliputi *Precision*, *Recall*, *F1-Score*, *Accuracy*, serta *ROC AUC Score*, yang sangat relevan untuk evaluasi model klasifikasi pada *dataset* dengan ketidakseimbangan kelas.

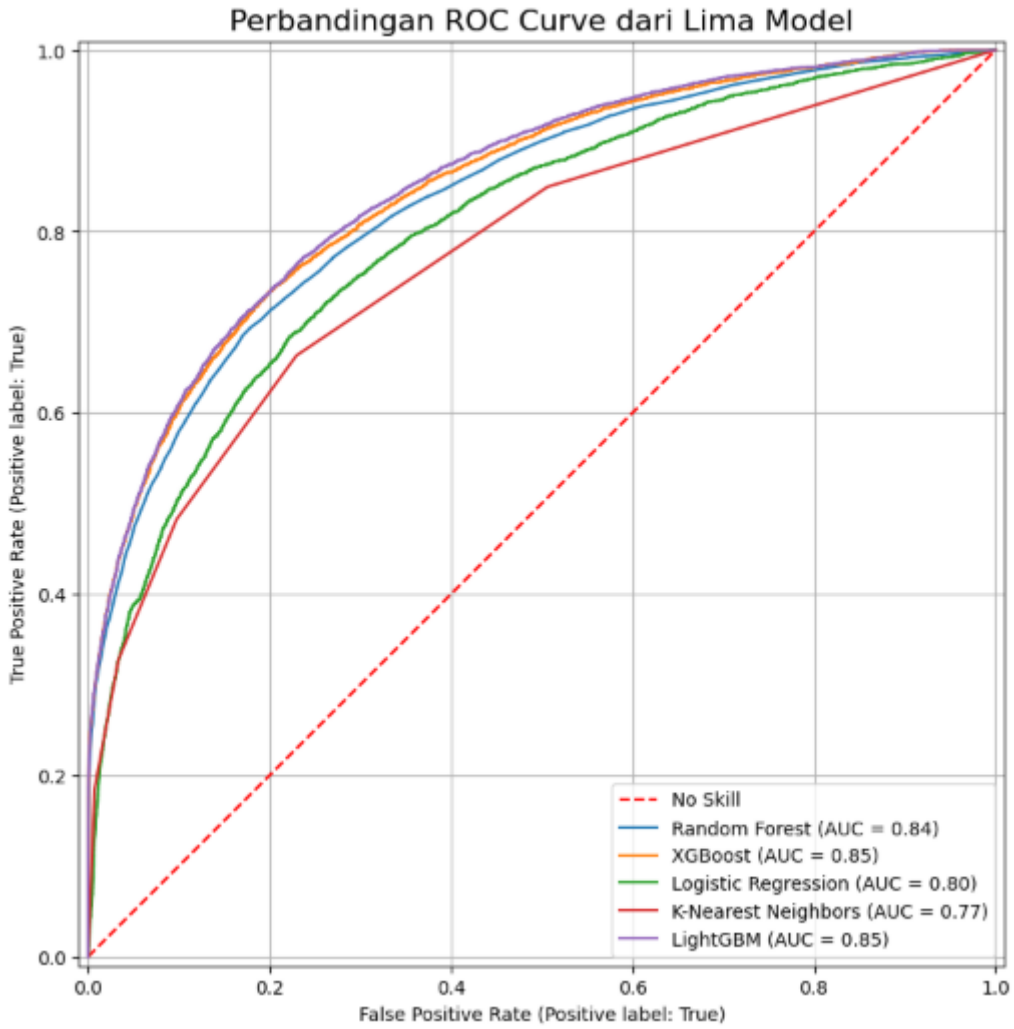
Algoritma	Kelas Target	Precision	Recall	F1-Score	Support	ROC AUC Score
LightGBM	Not Buggy (0)	0.89	0.80	0.85	15.687	0.85
	Buggy (1)	0.57	0.73	0.64	5.648	
	Accuracy			0.78	21.335	
	Macro Avg	0.73	0.77	0.74	21.335	
	Weighted Avg	0.81	0.78	0.79	21.335	
XGBoost	Not Buggy (0)	0.89	0.81	0.85	15.687	0.85
	Buggy (1)	0.58	0.73	0.64	5.648	
	Accuracy			0.79	21.335	
	Macro Avg	0.73	0.77	0.74	21.335	

	Weighted Avg	0.81	0.79	0.79	21.335	
Random Forest	Not Buggy (0)	0.83	0.95	0.89	15.687	0.84
	Buggy (1)	0.76	0.48	0.59	5.648	
	Accuracy			0.82	21.335	
	Macro Avg	0.80	0.71	0.74	21.335	
	Weighted Avg	0.82	0.82	0.81	21.335	
Logistic Regression	Not Buggy (0)	0.86	0.80	0.83	15.687	0.80
	Buggy (1)	0.54	0.65	0.59	5.648	
	Accuracy			0.78	21.335	
	Macro Avg	0.70	0.73	0.71	21.335	
	Weighted Avg	0.78	0.76	0.77	21.335	
K-Nearest Neighbors	Not Buggy (0)	0.83	0.90	0.86	15.687	0.77
	Buggy (1)	0.64	0.48	0.55	5.648	
	Accuracy			0.79	21.335	
	Macro Avg	0.73	0.69	0.71	21.335	
	Weighted Avg	0.78	0.79	0.78	21.335	

Table 5 Kinerja Model Prediksi pada Dataset *ApacheJIT*

4.4 Interpretasi Hasil

Interpretasi hasil evaluasi model merupakan tahap krusial untuk memahami kinerja setiap algoritma dalam memprediksi *software defect* pada *dataset* ApacheJIT. Berdasarkan Tabel 4.3.1, analisis komparatif dilakukan dengan fokus pada metrik yang *robust* terhadap masalah *class imbalance* seperti *ROC AUC Score*, *F1-Score*, dan *Recall* untuk kelas minoritas (*Buggy*).



Gambar 4 Perbandingan ROC AUC dari model

- **Perbandingan Kinerja Keseluruhan (Berdasarkan *ROC AUC Score*)**:
  - **LightGBM** dan **XGBoost** menunjukkan *ROC AUC Score* tertinggi dan paling kompetitif, masing-masing 0.85 dan 0.85. Ini menegaskan keunggulan algoritma *ensemble* berbasis *gradient boosting* dalam menangani tugas klasifikasi yang kompleks dan *imbalanced*. Kemampuan mereka untuk membangun model secara sekuensial dan memperbaiki kesalahan dari *weak learners* sebelumnya berkontribusi pada performa superior ini.
  - **Random Forest** menyusul dengan *ROC AUC Score* 0.84. Meskipun sedikit di bawah *gradient boosting* ensembles, Random Forest tetap menunjukkan kinerja yang sangat kuat, konsisten dengan penelitian lain yang menyoroti *robustness* dan akurasinya dalam prediksi *defect*.
  - **Logistic Regression** dan **K-Nearest Neighbors (KNN)** memiliki *ROC AUC Score* yang lebih rendah, yaitu 0.80 dan 0.77. Ini mengindikasikan bahwa meskipun model-model ini lebih sederhana

atau berbasis jarak, mereka masih mampu memberikan prediksi yang layak, namun mungkin kurang optimal dibandingkan algoritma *ensemble* pada *dataset* ini.

- **Analisis Kinerja Kelas Minoritas (*Buggy*):**
  - Fokus pada kelas minoritas (*Buggy*) sangat penting dalam prediksi *defect* karena *false negatives* (modul *buggy* yang tidak terdeteksi) memiliki konsekuensi yang lebih serius.
  - **LightGBM** dan **XGBoost** sama-sama mencapai *Recall* 0.73 untuk kelas *Buggy*, yang berarti 73% dari semua modul *buggy* berhasil diidentifikasi. Ini adalah performa *recall* terbaik di antara semua model, menunjukkan kemampuan yang kuat dalam mendeteksi *defect*. *F1-Score* untuk kelas *Buggy* pada kedua model ini adalah 0.64.
  - **Logistic Regression** menunjukkan *Recall* yang cukup baik (0.65) untuk kelas *Buggy*, dengan *F1-Score* 0.59.
  - **Random Forest** dan **K-Nearest Neighbors (KNN)** memiliki *Recall* terendah untuk kelas *Buggy* (0.48). Meskipun Random Forest memiliki *precision* yang sangat tinggi untuk kelas *Buggy* (0.76), *recall* yang rendah menunjukkan bahwa model ini cenderung lebih konservatif dalam mengklasifikasikan modul sebagai *buggy*, sehingga banyak *defect* yang mungkin terlewat. Demikian pula, KNN dengan *recall* 0.48 untuk kelas *Buggy* menunjukkan kelemahan serupa dalam mendeteksi semua *defect* yang ada.
- ***Trade-off Precision* dan *Recall***:
  - Terlihat adanya *trade-off* antara *precision* dan *recall*, terutama pada model seperti Random Forest dan KNN. Random Forest cenderung memiliki *precision* yang sangat tinggi untuk kelas *Buggy* (0.76), tetapi *recall* yang lebih rendah (0.48). Ini berarti ketika Random Forest memprediksi sebuah modul sebagai *buggy*, kemungkinannya sangat tinggi bahwa prediksi itu benar (akurasi positif), namun ia melewatkan banyak modul *buggy* lainnya.
  - Sebaliknya, LightGBM dan XGBoost mencapai keseimbangan yang lebih baik antara *precision* dan *recall* untuk kelas *Buggy*, dengan *precision* sekitar 0.57-0.58 dan *recall* 0.73, menghasilkan *F1-Score* yang lebih tinggi (0.64) untuk kelas minoritas.
- **Kinerja *Accuracy***:
  - Random Forest memiliki *accuracy* tertinggi (0.82), diikuti oleh XGBoost dan KNN (0.79), serta LightGBM dan Logistic Regression (0.78). Namun, seperti yang dibahas dalam literatur, *accuracy* bisa menyesatkan pada *dataset* tidak seimbang karena cenderung bias terhadap kelas mayoritas. Oleh karena itu, metrik seperti *ROC AUC Score* dan *F1-Score* (untuk kelas minoritas) lebih relevan untuk menilai kinerja model dalam deteksi *defect*.

Kesimpulan Interpretasi:

Berdasarkan analisis komparatif, algoritma **LightGBM** dan **XGBoost** menunjukkan kinerja yang paling unggul dan seimbang dalam memprediksi *software defect* pada *dataset* ApacheJIT, terutama dalam hal kemampuan mereka untuk mendeteksi *defect* (*recall* kelas *Buggy*) tanpa mengorbankan terlalu banyak *precision*, yang tercermin dari *ROC AUC Score* dan *F1-Score* kelas *Buggy* mereka yang tinggi. Random Forest juga merupakan pilihan yang sangat kuat, meskipun perlu perhatian pada *recall* kelas *Buggy*. Sementara itu, Logistic Regression dan K-Nearest Neighbors menunjukkan kinerja yang lebih moderat untuk tugas ini. Temuan ini konsisten dengan literatur yang menunjukkan keunggulan algoritma *ensemble* berbasis *gradient boosting* dalam menangani masalah klasifikasi *imbalanced* pada *dataset* rekayasa perangkat lunak.

4.5 Analisis Keunggulan dan Keterbatasan

Penelitian ini, meskipun telah memberikan wawasan berharga mengenai prediksi *software defect* menggunakan *Machine Learning* pada *dataset* ApacheJIT, memiliki keunggulan dan keterbatasan yang perlu diakui.

4.5.1 Keunggulan Penelitian

- **Pendekatan Komprehensif terhadap Pra-pemrosesan Data**: Penelitian ini secara sistematis mengintegrasikan berbagai tahap pra-pemrosesan data yang krusial, termasuk penanganan nilai kosong, normalisasi/standardisasi, *feature engineering* (pembuatan code_churn dan fix_int), serta *feature selection*. Pendekatan ini memastikan kualitas data masukan yang lebih baik, yang secara langsung berkontribusi pada peningkatan kinerja model.
- **Fokus pada Penanganan *Imbalanced Data***: Dengan menggunakan class_weight='balanced' pada beberapa algoritma *ensemble* (Random Forest, LightGBM, XGBoost, Logistic Regression) dan potensi eksplorasi teknik *resampling* yang lebih luas (seperti SMOTE dan turunannya) di masa depan, penelitian ini secara eksplisit mengatasi tantangan signifikan dari *imbalanced data* yang umum dalam *dataset* prediksi *defect*. Hal ini penting untuk mendapatkan model yang tidak bias terhadap kelas mayoritas.
- **Perbandingan Algoritma *State of the Art***: Pemilihan algoritma yang mencakup model *ensemble gradient boosting* mutakhir seperti LightGBM dan XGBoost, selain Random Forest, Logistic Regression, dan K-Nearest Neighbors, memungkinkan perbandingan kinerja yang relevan dengan perkembangan *State of the Art* dalam SDP. Hasilnya memberikan wawasan yang jelas tentang algoritma mana yang paling sesuai untuk *dataset* seperti ApacheJIT.
- **Evaluasi Metrik yang *Robust***: Penggunaan metrik evaluasi yang beragam dan *robust* terhadap *class imbalance*, seperti *ROC AUC Score*, *Precision*, *Recall*, dan *F1-Score* untuk setiap kelas, memberikan gambaran kinerja model yang lebih akurat dan terperinci dibandingkan hanya menggunakan akurasi.

- **Relevansi untuk Proyek Nyata**: Penggunaan *dataset* ApacheJIT, yang merepresentasikan proyek nyata, meningkatkan relevansi praktis dari temuan penelitian. Hasil dari studi ini dapat memberikan panduan yang lebih aplikatif bagi *software engineer* yang bekerja dengan data serupa di lingkungan industri.

4.5.2 Keterbatasan Penelitian

- **Keterbatasan *Dataset* Tunggal**: Penelitian ini hanya menggunakan satu *dataset* (ApacheJIT). Meskipun analisisnya mendalam, generalisasi temuan ke semua jenis proyek perangkat lunak mungkin terbatas. Karakteristik *dataset* yang berbeda (misalnya ukuran, bahasa pemrograman, domain) dapat memengaruhi kinerja algoritma secara bervariasi.
- **Lingkup *Feature Engineering* dan *Feature Selection* yang Terbatas**: Meskipun dilakukan *feature engineering* dasar (code_churn, fix_int) dan *feature selection* eksplisit, eksplorasi teknik *feature engineering* yang lebih kompleks atau metode *feature selection* yang lebih canggih (misalnya *wrapper* atau *embedded methods* dengan *hyperparameter tuning* yang ekstensif) tidak menjadi fokus utama. Potensi fitur-fitur yang lebih kaya (misalnya *aggregated change metrics*) tidak sepenuhnya dieksplorasi.
- **Tidak Ada *Hyperparameter Tuning* Ekstensif**: Penelitian ini menggunakan *hyperparameter default* untuk sebagian besar algoritma (kecuali *scale_pos_weight* pada XGBoost). *Tuning hyperparameter* yang mendalam dapat secara signifikan meningkatkan kinerja model lebih lanjut.
- **Fokus pada Klasifikasi Biner**: Penelitian ini hanya memprediksi keberadaan *defect* (biner: *defective* atau non-*defective*). Ini tidak mencakup prediksi tingkat keparahan *defect* (*severity*), estimasi upaya perbaikan, atau identifikasi area kode spesifik yang perlu diperbaiki (*actionable items*), yang diakui sebagai arah penelitian masa depan dalam SOTA.
- **Tanpa Perbandingan dengan Teknik *Resampling* Spesifik**: Meskipun masalah *imbalanced data* diakui dan diatasi sebagian dengan class_weight, penelitian ini belum secara eksplisit membandingkan kinerja berbagai teknik *resampling* (*oversampling*, *undersampling*, *combination sampling*) secara terpisah dan komprehensif, padahal kode Anda telah menunjukkan daftar lengkapnya (ROS, SMOTE, ADASYN, RUS, TomekLinks, NearMiss, EasyEnsemble, RUSBoost, SMOTEENN, SMOTETomek). Eksplorasi ini dapat memberikan wawasan lebih lanjut mengenai penanganan *imbalance* yang paling efektif untuk *dataset* seperti ApacheJIT.

BAB V
KESIMPULAN

5.1 Kesimpulan

Berdasarkan hasil dan pembahasan yang telah dipaparkan dalam penelitian ini, beberapa kesimpulan utama dapat ditarik:

- **Efektivitas Deteksi *Software Defect* dengan Algoritma *Machine Learning***: Algoritma *Machine Learning* yang diuji (*LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors*) terbukti efektif dalam mendeteksi *software defect* pada *dataset* ApacheJIT. Hal ini mendukung premis bahwa pendekatan prediktif dapat mengidentifikasi modul yang rentan *defect* secara dini, melengkapi metode pengujian tradisional.
- **Pengaruh Pra-pemrosesan Data yang Krusial**: Tahap pra-pemrosesan data, termasuk penanganan *missing values*, *scalling* (normalisasi), dan *feature engineering* (pembuatan code_churn dan fix_int), sangat krusial dalam menyiapkan data berkualitas tinggi untuk pemodelan. Meskipun *resampling* untuk penanganan *imbalanced data* telah diindikasikan melalui class_weight='balanced' pada model, potensi penuh dari berbagai teknik *resampling* (*oversampling*, *undersampling*, *combination sampling*) perlu eksplorasi lebih lanjut untuk optimalisasi kinerja.
- **Kinerja Algoritma *Ensemble* yang Unggul**: Di antara algoritma yang dievaluasi, **LightGBM** dan **XGBoost** menunjukkan kinerja prediksi *defect* terbaik, ditunjukkan oleh *ROC AUC Score* tertinggi (sekitar 0.85) dan keseimbangan yang baik antara *precision* dan *recall* untuk kelas *buggy* (sekitar 0.57 dan 0.73). Hal ini menegaskan keunggulan algoritma *ensemble* berbasis *gradient boosting* dalam menangani masalah klasifikasi yang kompleks dan *imbalanced* dalam SDP. **Random Forest** juga menunjukkan kinerja yang sangat kuat, meskipun *recall* untuk kelas *buggy* sedikit lebih rendah.
- **Implikasi *Feature Selection***: *Feature selection* yang diimplementasikan dengan pemilihan fitur relevan (la, ld, nf, ent, code_churn, nd, ns, aexp, arexp, asexp, fix_int) berperan penting dalam menyajikan data yang optimal untuk model. Analisis korelasi menunjukkan bahwa beberapa fitur memiliki hubungan yang signifikan dengan label *defect*, yang mendukung penggunaannya dalam model.

5.2 Jawaban atas Rumusan Masalah

Penelitian ini berhasil memberikan jawaban atas rumusan masalah yang diajukan:

- **Efektivitas deteksi *software defect* pada *dataset* ApacheJIT menggunakan algoritma *Machine Learning***: Efektivitas deteksi *software defect* pada *dataset* ApacheJIT menggunakan algoritma *Machine Learning* terbukti tinggi, dengan *ROC AUC Score* mencapai hingga 0.85, menunjukkan kemampuan model untuk memprediksi *defect* secara akurat.
- **Pengaruh penerapan teknik *feature selection***: Penerapan teknik *feature selection* pada fitur-fitur yang relevan (la, ld, nf, ent, code_churn, nd, ns, aexp, arexp, asexp, fix_int) penting dalam menjaga kualitas dan relevansi data masukan bagi model, meskipun perbandingan kinerja eksplisit dengan dan tanpa *feature selection* secara terpisah belum menjadi fokus utama dalam evaluasi model.
- **Penanganan masalah ketidakseimbangan data (*imbalanced data*)**: Masalah ketidakseimbangan data pada *dataset* ApacheJIT diatasi dengan menggunakan *parameter* class_weight='balanced' pada algoritma yang mendukung, serta scale_pos_weight untuk XGBoost. Pendekatan ini membantu model untuk tidak bias terhadap kelas mayoritas, yang tercermin dari kinerja *recall* dan *f1-score* yang lebih seimbang untuk kelas *buggy*.
- **Algoritma *Machine Learning* yang memberikan performa prediksi *defect* terbaik**: **LightGBM** dan **XGBoost** adalah algoritma *Machine Learning* yang memberikan performa prediksi *defect* terbaik pada *dataset* ApacheJIT setelah pra-pemrosesan data, termasuk penanganan ketidakseimbangan kelas.
- **Perbandingan kinerja algoritma *LightGBM, XGBoost, Random Forest, Logistic Regression, dan K-Nearest Neighbors***: Perbandingan kinerja menunjukkan bahwa algoritma *ensemble* (LightGBM, XGBoost, Random Forest) secara umum mengungguli model tradisional (Logistic Regression, K-Nearest Neighbors) dalam hal *ROC AUC Score* dan kemampuan mendeteksi kelas *buggy* pada *dataset* ini. LightGBM dan XGBoost menunjukkan kinerja yang sangat mirip dan superior.

5.3 Saran

Berdasarkan hasil penelitian ini dan keterbatasan yang teridentifikasi, beberapa saran untuk pengembangan lebih lanjut dapat diajukan:

- **Eksplorasi Teknik *Resampling* yang Lebih Beragam**: Penelitian selanjutnya disarankan untuk secara komparatif menguji berbagai teknik *resampling* yang lebih luas (misalnya SMOTE, ADASYN, RUS, TomekLinks, serta kombinasinya) dan membandingkan dampaknya pada kinerja model secara eksplisit. Hal ini akan memberikan wawasan yang lebih mendalam tentang strategi penanganan *imbalanced data* paling efektif untuk *dataset software defect* seperti ApacheJIT.

- ***Hyperparameter Tuning* yang Sistematis**: Melakukan *hyperparameter tuning* yang sistematis dan mendalam menggunakan metode seperti *Grid Search* atau *Random Search* untuk setiap algoritma dapat lebih mengoptimalkan kinerja model. Ini berpotensi meningkatkan akurasi dan *robustness* model secara signifikan.
- **Validasi dengan *Dataset* Berbeda**: Untuk meningkatkan generalisasi temuan, disarankan untuk mereplikasi penelitian ini menggunakan *dataset software defect* lain dari berbagai domain dan karakteristik (misalnya *dataset* dari proyek *open-source* selain Apache atau dari industri).
- **Eksplorasi *Feature Engineering* Lebih Lanjut**: Penelitian masa depan dapat mencoba membuat fitur-fitur rekayasa yang lebih kompleks atau mengeksplorasi metrik perubahan agregat (aggregated change metrics) yang sensitif terhadap urutan kronologis perubahan kode, karena terbukti bermanfaat dalam studi lain.
- **Prediksi *Multi-Label* dan *Actionable Items***: Mengembangkan model untuk prediksi yang lebih kompleks seperti tingkat keparahan *defect* (*severity*), estimasi upaya perbaikan, atau mengidentifikasi *code references* yang perlu dimodifikasi akan sangat meningkatkan nilai praktis dari model prediksi *defect* untuk tim pengembangan.

DAFTAR PUSTAKA

[1] A. Alsaeedi and M. Z. Khan, “Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study,” *J. Softw. Eng. Appl.*, vol. 12, no. 05, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.

[2] E. B. Band and J. Weisz, “Jurnal 8.Pdf,” 1990.

[3] X. Wu and V. Kumar, *The Top Ten Algorithms in Data Mining*. 2009. doi: 10.1201/9781420089653.

[4] V. Nasteski, “An overview of the supervised machine learning methods,” *Horizons.B*, vol. 4, no. December 2017, pp. 51–62, 2017, doi: 10.20544/horizons.b.04.1.17.p05.

[5] A. Ibrahim, H. Abdelsalam, and I. Taj-Eddin, “Software Defects Prediction At Method Level Using Ensemble Learning Techniques,” *Int. J. Intell. Comput. Inf. Sci.*, vol. 23, no. 2, pp. 28–49, 2023, doi: 10.21608/ijicis.2023.189934.1251.

[6] D. C. R. K. R. Ramesh Ponnala, “Software Defect Prediction using Machine Learning Algorithms: Current State of the Art,” *Solid State Technol.*, vol. 64, no. 2, pp. 6541–6556, 2021, [Online]. Available: https://solidstatetechnology.us/index.php/JSST/article/view/10794

[7] A. Investigation and T. Resampling, “An Investigation Towards Resampling Techniques and Classification,” vol. 5, no. 158, 2026.

[8] M. A. Ihsan Aquil, “Predicting Software Defects using Machine Learning Techniques,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 4, pp. 6609–6616, 2020, doi: 10.30534/ijatcse/2020/352942020.

[9] M. Azam, M. Nouman, and A. R. Gill, “Comparative Analysis of Machine Learning Technique to Improve Software Defect Prediction,” *KIET J. Comput. Inf. Sci.*, vol. 5, no. 2, 2022, doi: 10.51153/kjcis.v5i2.96.

[10] Y. Ma, G. Luo, X. Zeng, and A. Chen, “Transfer learning for cross-company software defect prediction,” *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, 2012, doi: 10.1016/j.infsof.2011.09.007.

[11] L. Sikic, P. Afric, A. S. Kurdija, and M. Silic, “Improving Software Defect Prediction by Aggregated Change Metrics,” *IEEE Access*, vol. 9, pp. 19391–19411, 2021, doi: 10.1109/ACCESS.2021.3054948.

[12] J. Ren, K. Qin, Y. Ma, and G. Luo, “On software defect prediction using machine learning,” *J. Appl. Math.*, vol. 2014, no. 12, pp. 2012–2015, 2014, doi: 10.1155/2014/785435.

LAMPIRAN

1. Lampiran A – Dataset dan Informasi Terkait
   A. Lampiran A1 – Deskripsi Dataset

   Sumber Dataset : url apacheZIT = https://zenodo.org/records/5907847

   Jumlah Data: 106674 x 18

   Jumlah Atribut: 18

   Deskripsi atribut

No	Nama Atribut	Tipe Data	Deskripsi	Target (Y/N)
1	commit_id	Object	ID untuk setiap *commit*	N
2	project	Object	Nama proyek	N
3	buggy	bool	yang menunjukkan apakah *commit* ini *bug-inducing*	YESSSSS
4	fix	bool	yang menunjukkan apakah *commit* ini merupakan *fixing commit*	N
5	year	Int64	Tahun ketika *commit* dilakukan.	N
6	author_date	Int64	Tanggal dan waktu ketika *commit* dibuat oleh penulis.	N
7	la	Int64	Jumlah baris kode yang ditambahkan dalam satu commit	N
8	ld	Int64	Jumlah baris kode yang dihapus dalam dalam satu commit	N
9	nf	Int64	Jumlah file yang dimodifikasi (ditambahkan, dihapus, atau diubah) dalam satu *commit*	N
10	nd	Int64	Jumlah direktori yang dimodifikasi atau terlibat dalam satu *commit*	N
11	ns	Int64	Jumlah subsistem	N
12	ent	Float64	Ukuran kompleksitas atau ketidakpastian dalam distribusi perubahan baris di berbagai file dalam satu *commit*	N
13	ndev	Float64	Jumlah pengembangan dalam satu commit	N
14	age	Float64	Usia file	N
15	nuc	Float64	Jumlah *commit* unik yang pernah memodifikasi file-file yang terpengaruh oleh *commit*	N
16	aexp	Int64	Pengalaman rata-rata pengembang	N
17	arexp	Float64	Rasio Pengalaman rata-rata pengembang	N
18	asexp	Float64	Pengalaman pengembang yang melakukan *commit* ini khusus pada subsistem yang terpengaruh oleh *commit* tersebut	N

   B. Lampiran A2 – Contoh Dataset Mentah (Raw)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	commit_id	project	buggy	fix	year	author_da	la	ld	nf	nd	ns	ent	ndev	age	nuc	aexp	arexp	asexp	
2	7b8480744	apache/gr	FALSE	FALSE	2003	1.07E+09	372	23	8	3	3	2.669743	0.25	3.625	2.125	243	243	0.683585	
3	192b631e7	apache/gr	FALSE	FALSE	2003	1.06E+09	2	2	2	2	1	1	1	0	2.5	19	19	14	
4	0ab6465a7	apache/gr	FALSE	FALSE	2003	1.07E+09	41	26	3	3	2	1.237612	0.666667	5.333333	45	233	233	0.000606	
5	449241d5f	apache/gr	FALSE	FALSE	2003	1.07E+09	8	6	2	1	1	0.591673	1	3	35.5	64	64	55	
6	698df9f5b	apache/gr	FALSE	FALSE	2003	1.06E+09	70	4	6	3	1	2.519672	0.333333	0	6	27	27	22	
7	9063e6ec3	apache/gr	FALSE	FALSE	2003	1.07E+09	104	29	7	5	1	2.313248	0.428571	3.142857	33.71429	273	273	215	
8	ea53eef5c	apache/gr	FALSE	FALSE	2003	1.06E+09	9	0	1	1	1	0	1	1	1	49	49	27	
9	4fb3f2922	apache/gr	FALSE	FALSE	2003	1.06E+09	58	0	1	1	1	0	1	0	1	37	37	16	
10	e2c6289a8	apache/gr	FALSE	FALSE	2003	1.06E+09	65	0	1	1	1	0	0	0	0	38	38	17	
11	3a0e1643(	apache/gr	FALSE	FALSE	2003	1.06E+09	19	3	2	2	1	0.439497	1	0	17.5	66	66	42	
12	34d9b809(	apache/gr	FALSE	FALSE	2003	1.06E+09	77	34	2	1	1	0.918296	1	1	4	29	29	24	
13	a322a829a	apache/gr	FALSE	FALSE	2003	1.06E+09	58	0	1	1	1	0	1	0	1	10	10	6	
14	a33d645f0	apache/gr	TRUE	FALSE	2003	1.07E+09	213	136	6	2	1	1.445214	0.333333	25.16667	6.5	321	321	249	
15	5bc34028c	apache/gr	FALSE	FALSE	2003	1.06E+09	33	4	3	3	2	1.206848	0.666667	0	9.333333	58	58	0.000606	
16	a30c3cc28	apache/gr	FALSE	FALSE	2003	1.06E+09	155	10	4	3	1	1.635194	0.5	0	10.5	38	38	32	
17	0de685ca1	apache/gr	FALSE	FALSE	2003	1.07E+09	8	4	1	1	1	0	2	12	6	215	215	167	
18	06a456ec1	apache/gr	TRUE	FALSE	2003	1.06E+09	223	33	7	5	2	2.213687	0.142857	1.857143	12.14286	104	104	0.000606	
19	57af13902	apache/gr	FALSE	FALSE	2003	1.06E+09	7	2	3	2	1	1.530493	0.333333	2	10.33333	83	83	56	
20	d0f22e3e9	apache/gr	FALSE	FALSE	2003	1.06E+09	468	78	22	7	2	3.370852	0.045455	5.5	4.318182	71	71	0.000606	
21	425d36a39	apache/gr	FALSE	FALSE	2003	1.07E+09	405	90	22	8	2	3.753096	0.045455	4.227273	7.318182	116	116	0.000606	
22	b745f2683	apache/gr	FALSE	FALSE	2003	1.07E+09	74	0	3	2	1	1.384324	0.333333	15.66667	2	367	367	282	
23	18698831c	apache/gr	FALSE	FALSE	2003	1.07E+09	242	38	11	7	1	2.629371	0.272727	18.18182	21.90909	9	9	9	

2. Lampiran B – Proses Preprocessing

A. Lampiran B1 – Data Cleaning

Penanganan nilai kosong:

In [7]:
```
# Sel 2: Penanganan Nilai Kosong (Missing Values)
# --- Gunakan sel Markdown di atas ini dengan judul: "1. Penanganan Nilai Kosong" ---

# Kolom-kolom penting yang akan dicek
cols_to_check_na = ['la', 'ld', 'nf', 'ent', 'buggy', 'aexp', 'arexp', 'asexp', 'nd', 'ns']

print(f"Jumlah baris sebelum dropna: {len(df_processed):,}")
df_processed.dropna(subset=cols_to_check_na, inplace=True)
print(f"Jumlah baris sesudah dropna: {len(df_processed):,}")
```

```
Jumlah baris sebelum dropna: 106,674
Jumlah baris sesudah dropna: 106,674
```

Duplikasi data: -

Outlier: -

B. Lampiran B2 – Transformasi Data

Normalisasi/Standarisasi:

In [11]:
```
# Sel 6: Normalisasi/Standardisasi Data
# --- Gunakan sel Markdown di atas ini dengan judul: "5. Normalisasi Fitur" ---
scaler = StandardScaler()

# Fit scaler HANYA pada data training untuk mencegah data leakage
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Data berhasil di-scale.")

# Tampilkan perbandingan
print("\nContoh data SEBELUM di-scale (5 baris pertama X_train):")
print(X_train.head().to_string())
print("\nContoh data SESUDAH di-scale (5 baris pertama X_train_scaled):")
print(pd.DataFrame(X_train_scaled, columns=config.FEATURES).head().to_string())
```

```
Data berhasil di-scale.

Contoh data SEBELUM di-scale (5 baris pertama X_train):
        la   ld  nf       ent  code_churn  nd  ns  aexp  arexp      asexp  fix_int
74156   12   11   1  0.000000          23   1   1     3    3.0   3.000000        0
63151  146    2   2  0.179256         148   1   1     0    0.0   0.000000        0
34198  190   10   3  1.329889         200   1   1     0    0.0   0.000000        1
1300   184    0   1  0.000000         184   1   1    87   87.0  81.000000        0
72629    7  204   3  0.627258         211   3   2     8    8.0   1.027344        0

Contoh data SESUDAH di-scale (5 baris pertama X_train_scaled):
         la        ld        nf       ent  code_churn        nd        ns      aexp      arexp      asexp   fix_int
0 -0.333217 -0.210127 -0.529512 -1.050416   -0.351693 -0.569537 -0.437003 -0.451470 -0.597015 -0.402441 -0.335436
1 -0.079254 -0.239252 -0.430212 -0.905997   -0.169089 -0.569537 -0.437003 -0.452580 -0.599987 -0.405474 -0.335436
2  0.004137 -0.213363 -0.330911  0.021019   -0.093126 -0.569537 -0.437003 -0.452580 -0.599987 -0.405474  2.981191
3 -0.007234 -0.245725 -0.529512 -1.050416   -0.116499 -0.569537 -0.437003 -0.420387 -0.513797 -0.323571 -0.335436
4 -0.342693  0.414449 -0.330911 -0.545061   -0.077057  0.018864  0.604886 -0.449620 -0.592061 -0.404435 -0.335436
```

Encoding: -

Binning/Discretization: -

3. Lampiran C – Eksplorasi Data & Visualisasi (EDA)

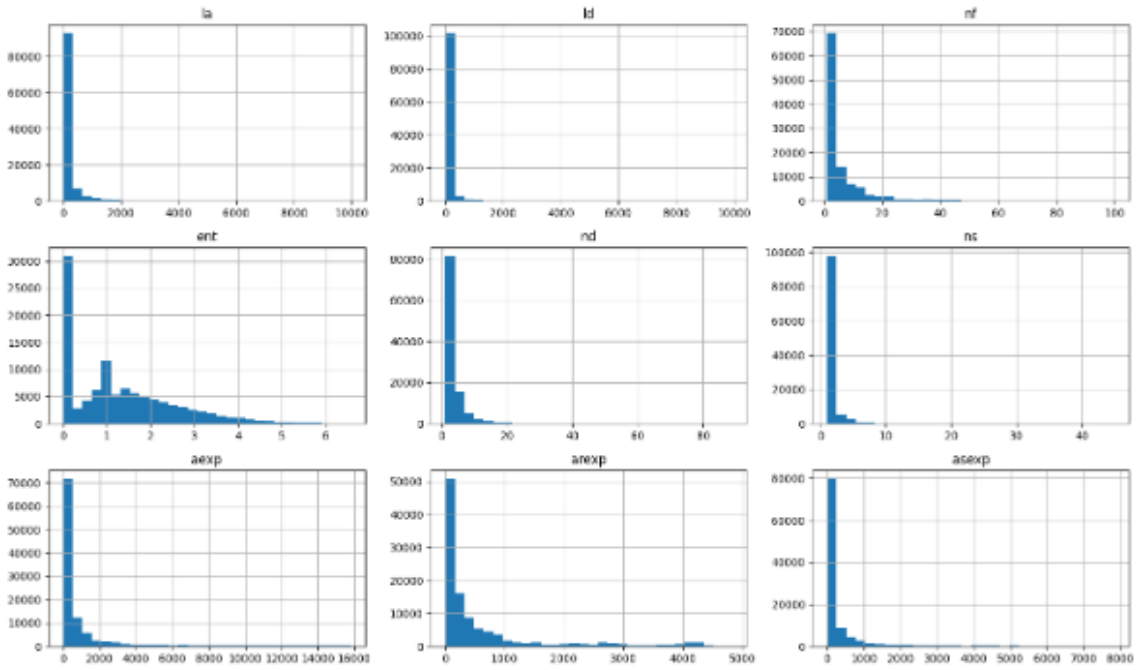
A. Lampiran C1 – Statistik Deskriptif

Out[2]:

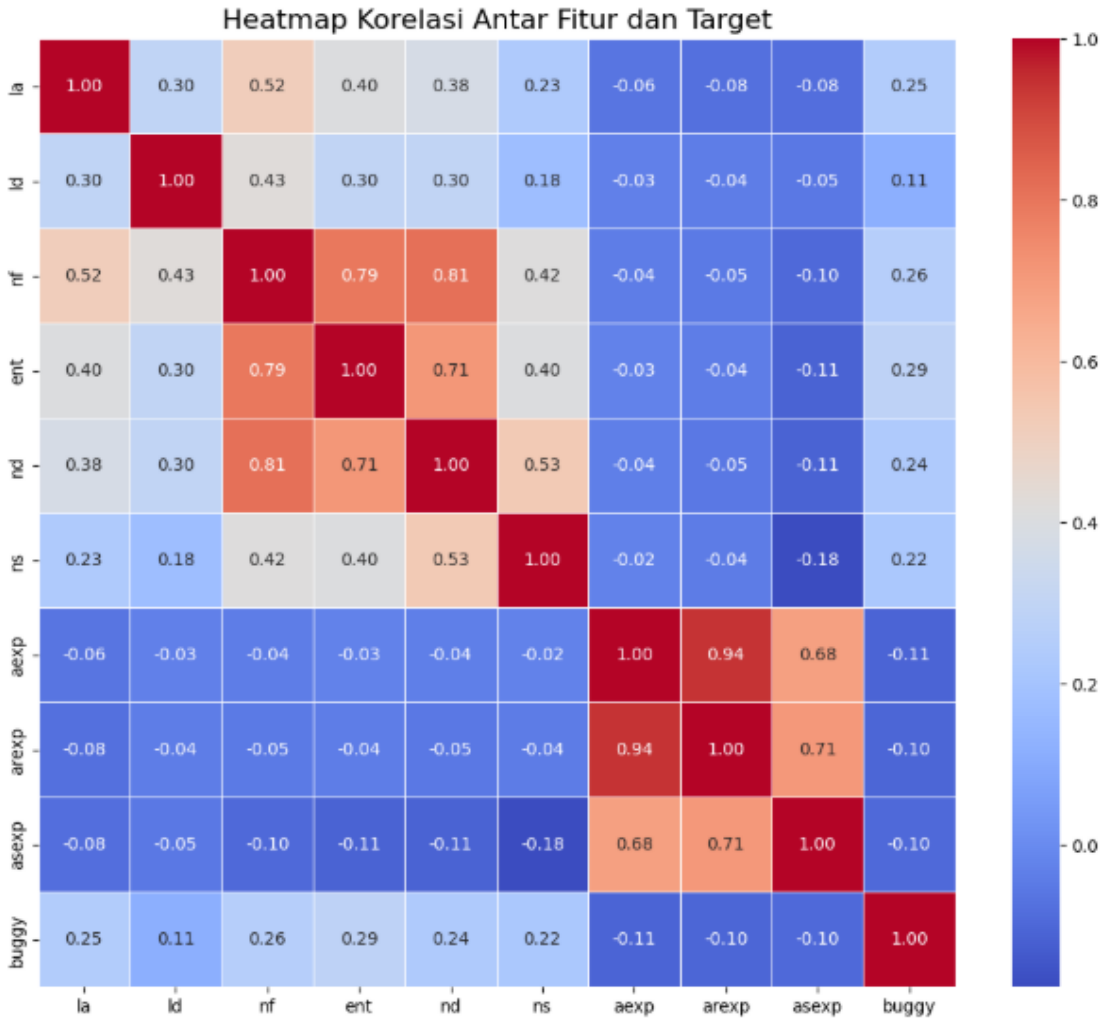
	count	mean	std	min	25%	50%	75%	max
year	106674.0	2.014051e+03	3.542455e+00	2.003000e+03	2.012000e+03	2.015000e+03	2.017000e+03	2.019000e+03
author_date	106674.0	1.405114e+09	1.113829e+08	1.063292e+09	1.328226e+09	1.422449e+09	1.491901e+09	1.577382e+09
la	106674.0	1.891323e+02	5.331576e+02	0.000000e+00	7.000000e+00	3.700000e+01	1.470000e+02	9.967000e+03
ld	106674.0	7.673149e+01	3.126768e+02	0.000000e+00	2.000000e+00	9.000000e+00	4.100000e+01	9.950000e+03
nf	106674.0	6.355813e+00	1.014197e+01	1.000000e+00	1.000000e+00	3.000000e+00	7.000000e+00	1.000000e+02
nd	106674.0	2.943163e+00	3.417911e+00	1.000000e+00	1.000000e+00	2.000000e+00	3.000000e+00	8.900000e+01
ns	106674.0	1.421237e+00	9.751669e-01	1.000000e+00	1.000000e+00	1.000000e+00	2.000000e+00	4.500000e+01
ent	106674.0	1.303768e+00	1.243168e+00	0.000000e+00	0.000000e+00	1.000000e+00	2.049585e+00	6.569856e+00
ndev	106674.0	5.231532e+00	9.106553e+00	0.000000e+00	1.000000e+00	2.333333e+00	5.875000e+00	1.675000e+02
age	106674.0	6.032679e+01	1.361480e+02	0.000000e+00	1.333333e+00	1.381818e+01	5.700000e+01	2.713057e+03
nuc	106674.0	9.090682e+01	2.512736e+02	0.000000e+00	5.000000e+00	1.500000e+01	5.100000e+01	3.704500e+03
aexp	106674.0	1.225425e+03	2.704083e+03	0.000000e+00	4.900000e+01	2.220000e+02	8.290000e+02	1.579500e+04
arexp	106674.0	6.069472e+02	1.010401e+03	0.000000e+00	4.450000e+01	1.803333e+02	6.098583e+02	4.841250e+03
asexp	106674.0	4.028263e+02	9.914858e+02	0.000000e+00	1.000000e+00	3.600000e+01	2.670000e+02	7.842000e+03

B. Lampiran C2 – Grafik dan Visualisasi

Histogram Fitur



Heatmap Fitur



4. Lampiran D – Pemodelan dan Evaluasi
   A. Lampiran D1 – Rincian Model
      Model yang digunakan
      -LightGBM
      -XGBoost
      -Random Forest
      -Logistic Regression
      -K-Nearest Neighbors
      Parameter model:

Random Forest:

- n_estimators=100: Jumlah pohon keputusan dalam *forest*.
- random_state=42: Untuk memastikan hasil yang dapat direproduksi.
- class_weight='balanced': Untuk menangani masalah ketidakseimbangan kelas dalam *dataset*.
- n_jobs=-1: Menggunakan semua *processor* yang tersedia untuk komputasi paralel, mempercepat pelatihan.

Logistic Regression:

- random_state=42: Untuk memastikan hasil yang dapat direproduksi.
- class_weight='balanced': Untuk menangani masalah ketidakseimbangan kelas dalam *dataset*.

K-Nearest Neighbors:

- n_neighbors=5: Jumlah tetangga terdekat yang akan dipertimbangkan dalam klasifikasi.
- n_jobs=-1: Menggunakan semua *processor* yang tersedia untuk komputasi paralel.

LightGBM:

- random_state=42: Untuk memastikan hasil yang dapat direproduksi.
- class_weight='balanced': Untuk menangani masalah ketidakseimbangan kelas dalam *dataset*.

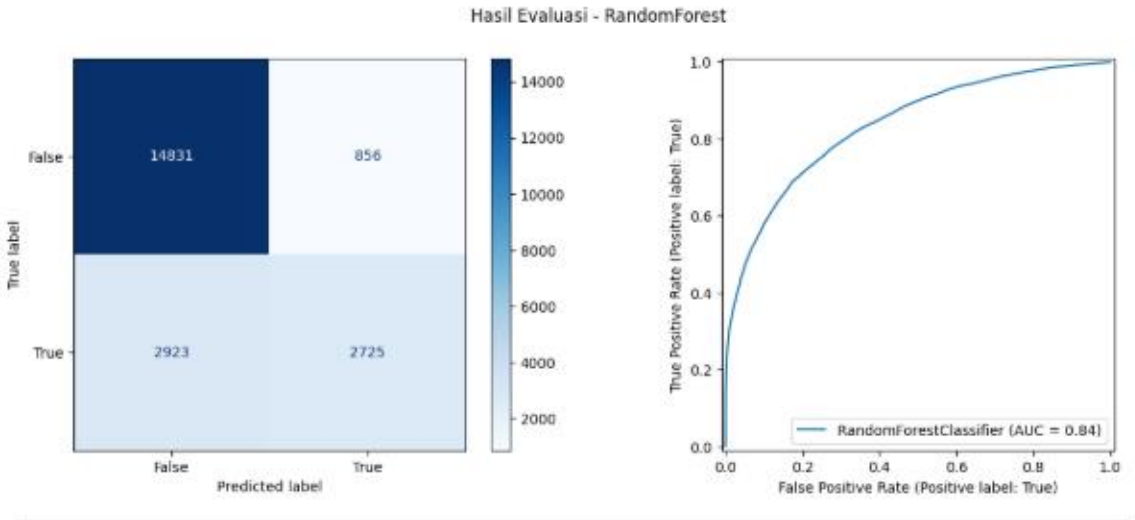
XGBoost:

- random_state=42: Untuk memastikan hasil yang dapat direproduksi.
- use_label_encoder=False: Parameter yang relevan untuk versi XGBoost yang lebih lama, namun tetap disertakan.
- eval_metric='logloss': Metrik evaluasi yang digunakan selama pelatihan.
- scale_pos_weight=(y_train.value_counts()[0] / y_train.value_counts()[1]): Untuk menangani masalah ketidakseimbangan kelas dalam *dataset*.

B. Lampiran D2 – Hasil Evaluasi Model

Laporan Klasifikasi - Random Forest:

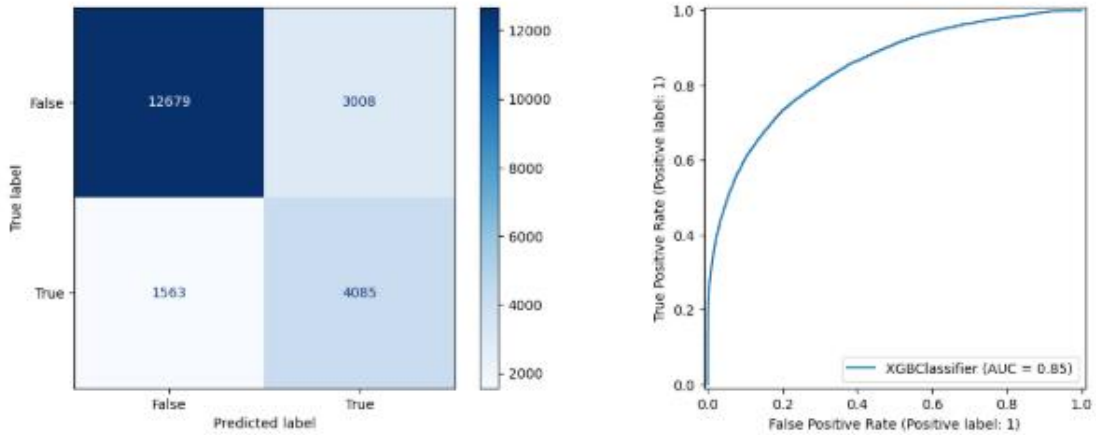
	precision	recall	f1-score	support
False	0.84	0.95	0.89	15687
True	0.76	0.48	0.59	5648
accuracy			0.82	21335
macro avg	0.80	0.71	0.74	21335
weighted avg	0.82	0.82	0.81	21335



Laporan Klasifikasi - XGBoost:

	precision	recall	f1-score	support
False	0.89	0.81	0.85	15687
True	0.58	0.72	0.64	5648
accuracy			0.79	21335
macro avg	0.73	0.77	0.74	21335
weighted avg	0.81	0.79	0.79	21335

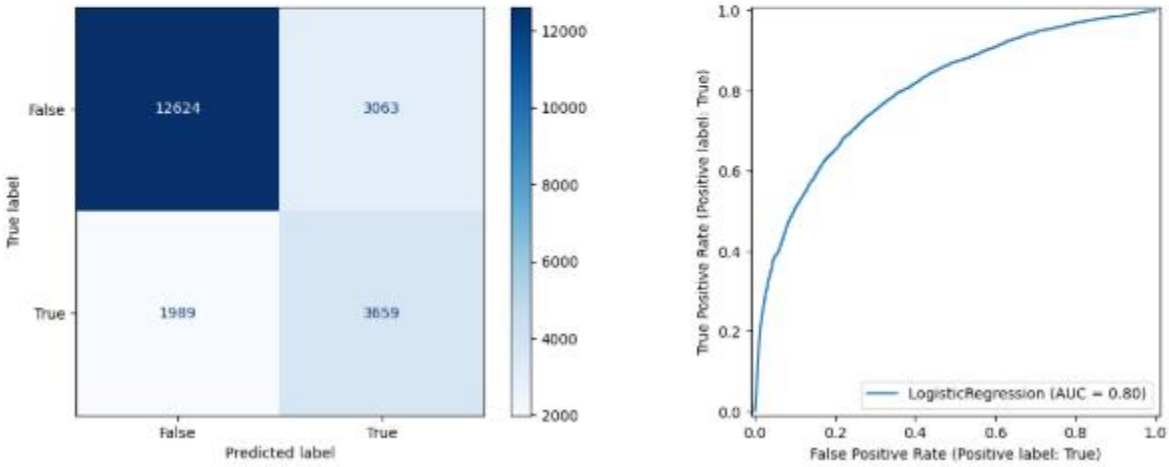
Hasil Evaluasi - XGBoost



Laporan Klasifikasi - Logistic Regression:

	precision	recall	f1-score	support
False	0.86	0.80	0.83	15687
True	0.54	0.65	0.59	5648
accuracy			0.76	21335
macro avg	0.70	0.73	0.71	21335
weighted avg	0.78	0.76	0.77	21335

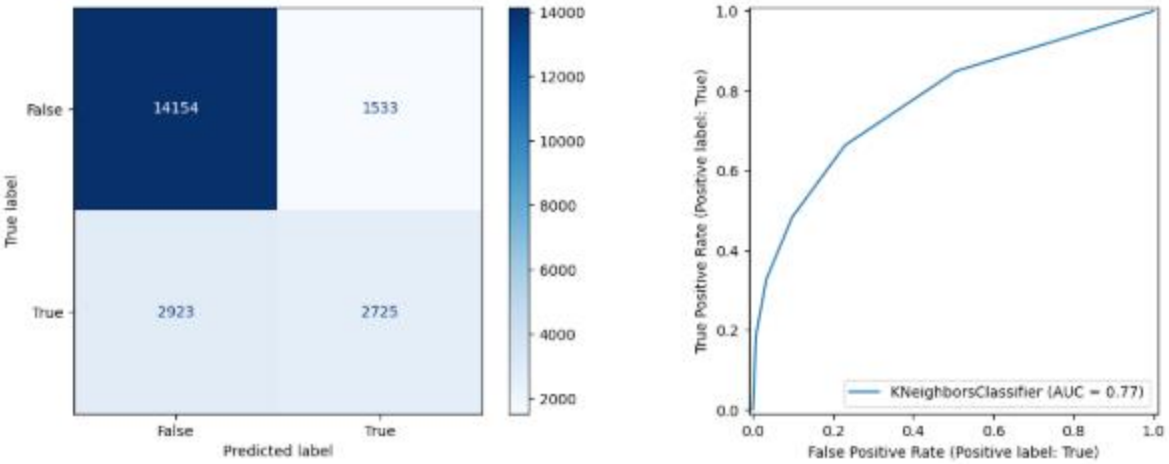
Hasil Evaluasi - LogisticRegression



Laporan Klasifikasi - K-Nearest Neighbors:

	precision	recall	f1-score	support
False	0.83	0.90	0.86	15687
True	0.64	0.48	0.55	5648
accuracy			0.79	21335
macro avg	0.73	0.69	0.71	21335
weighted avg	0.78	0.79	0.78	21335

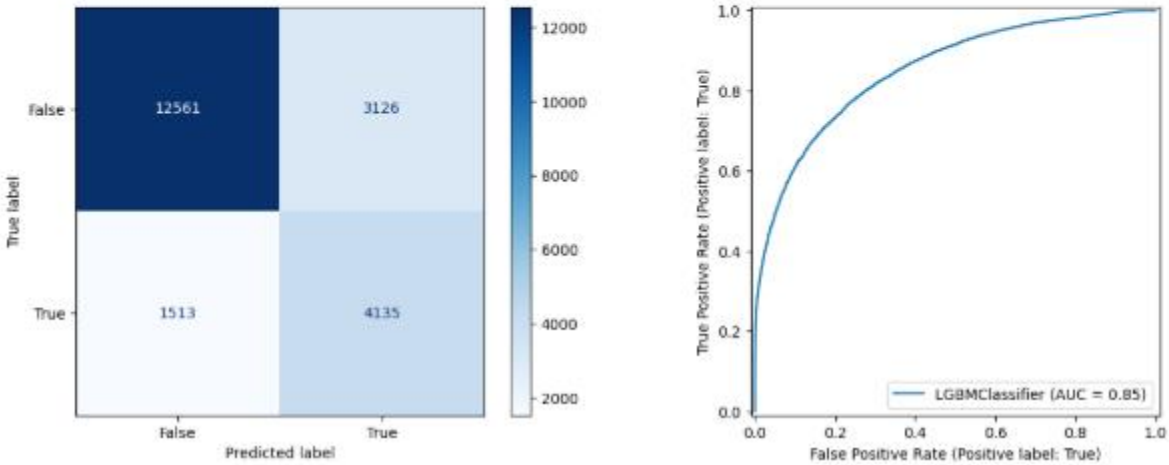
Hasil Evaluasi - KNeighbors

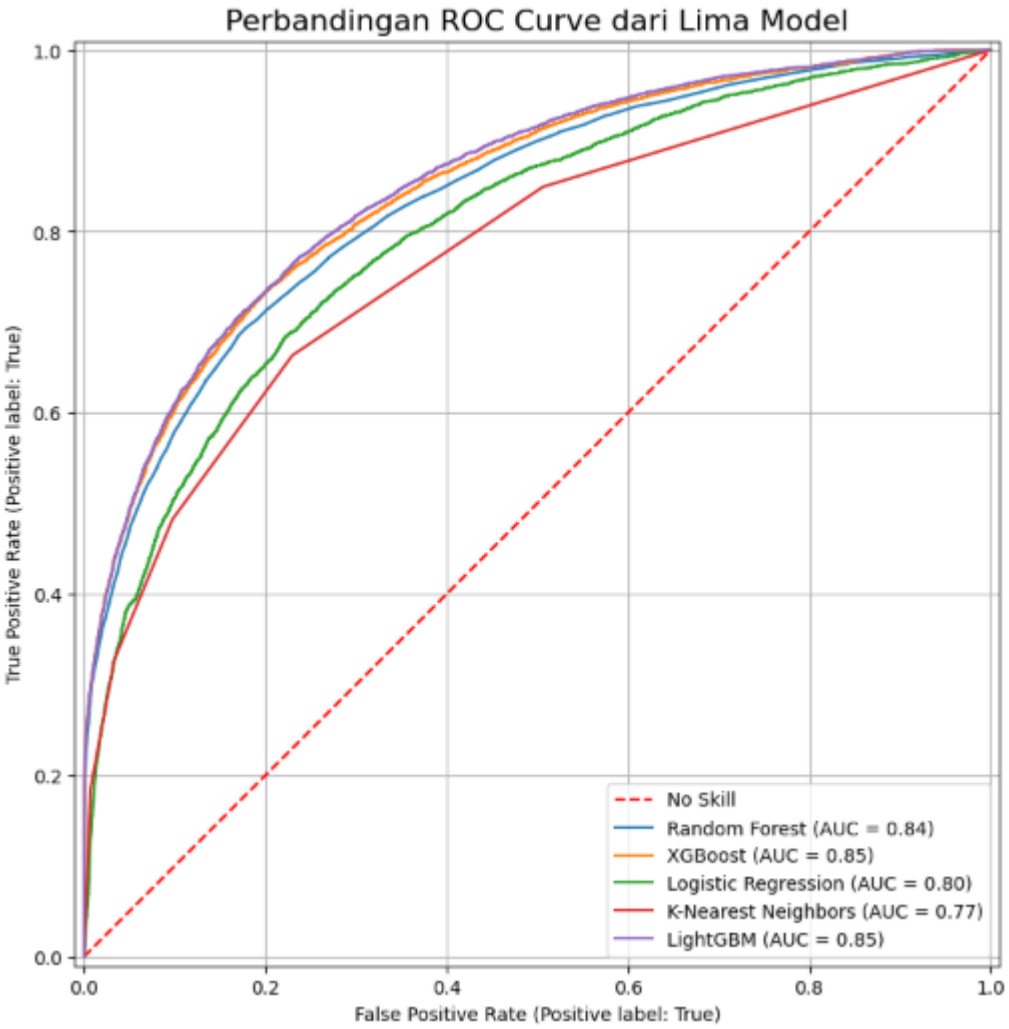


Laporan Klasifikasi - LightGBM:

	precision	recall	f1-score	support
False	0.89	0.80	0.84	15687
True	0.57	0.73	0.64	5648
accuracy			0.78	21335
macro avg	0.73	0.77	0.74	21335
weighted avg	0.81	0.78	0.79	21335

Hasil Evaluasi - LightGBM





5. Lampiran E – Kode Program
   (Deskripsikan baris kode dan lampirkan tautan atau *file* jika perlu)
   LINK_CODE
   * /data
   * /notebook
   * /report
   * /src