

Laporan Penambangan Data

Prediksi Kegagalan Bayar Kartu Kredit Nasabah dengan Menggunakan *Logistic Regression dan Random Forest*



Oleh :

Safana Maraya Nasiha 5002201051

Aqilla Yumna Aliefia 5002201086

M. Jalu Herlambang 5002201132

DEPARTEMEN MATEMATIKA
FAKULTAS SAINS DAN ANALITIKA DATA
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2023

Kata Pengantar

Puji syukur kehadirat Allah SWT atas limpahan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan laporan penelitian ini dengan baik. Laporan penelitian ini berjudul "Klasifikasi Penentuan Kelayakan Pemberian Kredit kepada Nasabah Berdasarkan Perilaku Penggunaan Kartu Kredit dengan Menggunakan Random Forest dan Logistic Regression". Kami juga tidak lupa mengucapkan terima kasih kepada semua pihak yang telah membantu kami, sehingga dapat memperlancar pembuatan laporan ini.

Penelitian ini diharapkan dapat memberikan manfaat bagi lembaga keuangan dan peneliti. Kami menyadari bahwa laporan ini tidak lepas dari kekurangan, baik dari segi bahasa maupun penulisan. Oleh karena itu, kami berharap adanya kritik dan saran guna menjadi bahan evaluasi dan perbaikan di masa yang akan datang.

Surabaya, 13 Desember 2023

Penulis

Daftar Isi

Kata Pengantar	1
Daftar Isi	2
BAB I	1
PENDAHULUAN	1
1.1 Latar Belakang	1
1.3 Tujuan	2
1.4 Manfaat	2
BAB II	3
KAJIAN TEORI	3
2.1 Kredit	3
2.2 Data Mining	3
2.3 Classification (Klasifikasi)	4
2.4 Logistic Regression	6
2.5 Decision Tree	7
2.6 Random Forest	7
BAB III	10
METODOLOGI	10
3.1 Sumber Data dan Variabel	10
3.2 Metode Analisis Data	10
3.3 Langkah Analisis	11
BAB IV	12
HASIL DAN PEMBAHASAN	12
4.1 Pengumpulan Data	12
4.1.1 Observasi	12
4.1.2 Studi Pustaka	12
4.2 Preprocessing Data	12
4.2.1 Data Cleaning	12
4.2.2 Exploratory Data Analysis	12
4.2.3 Label Encoding	16
4.2.4 One Hot Encoding	17
4.2.5 Handling Imbalance Class	17
4.2.6 Transformasi Data	19
4.2.7 Train Testing Splitting	19
4.3 Konstruksi Model	20
4.3.1 Logistic Regression	20
4.3.2 Klasifikasi Random Forest	21
4.3.3 Perbandingan Baseline Model	21
4.3.4 Kurva Receiver Operating Characteristic (ROC) Gabungan untuk Kedua Model	22
4.4 Peningkatan Performa Model	23

4.4.1 Feature Importance pada Model Random Forest	23
4.4.2 Cross Validation dan Hyperparameter Tuning	24
4.4.2.1 GridSearch pada Logistic Regression	24
4.4.2.2 GridSearch pada Random forest	24
4.5 Perbandingan Baseline model dan Optimal model	25
BAB V	27
KESIMPULAN	27
DAFTAR PUSTAKA	28
LAMPIRAN	1

BAB I

PENDAHULUAN

1.1 Latar Belakang

Menurut Undang-Undang Perbankan Nomor 10 Tahun 1998, kredit adalah penyediaan uang atau tagihan yang dapat dipersamakan dengan itu berdasarkan persetujuan atau kesepakatan pinjam-meminjam antar bank dengan pihak lain yang mewajibkan pihak peminjam melunasi hutangnya setelah jangka waktu tertentu dengan pemberian bunga. Pemberian kredit kepada nasabah merupakan salah satu kegiatan penting yang dilakukan oleh lembaga keuangan, seperti bank. Kredit merupakan salah satu bisnis utama bank yang memberikan keuntungan terbesar bagi bank, akan tetapi kredit juga memiliki resiko yang tinggi bagi kesehatan bank apabila dalam penyaluran kredit tidak dilakukan dengan baik. Pemberian kredit merupakan bisnis utama bank, sehingga bagian terbesar dari aset bank ialah kredit, hal ini juga menjadi alasan mengapa kredit sangat penting bagi sebuah bank, kualitas kredit yang baik akan sangat membantu bank dalam upaya meningkatkan keuntungan. Pemberian kredit yang tidak tepat dapat menyebabkan kerugian bagi lembaga keuangan, sedangkan pemberian kredit yang tepat dapat meningkatkan profitabilitas lembaga keuangan tersebut (Basori & Wahyuningsih, 2018).

Salah satu faktor yang penting untuk dipertimbangkan dalam pemberian kredit adalah kelayakan kredit nasabah. Kelayakan kredit nasabah adalah kemampuan nasabah untuk membayar kembali pinjamannya. Penilaian kelayakan kredit nasabah dapat dilakukan dengan berbagai cara, salah satunya adalah dengan menggunakan perilaku penggunaan kartu kredit nasabah. Perilaku penggunaan kartu kredit dapat memberikan gambaran tentang kemampuan nasabah untuk membayar kembali pinjamannya. Misalnya, nasabah yang memiliki riwayat pembayaran kartu kredit yang lancar cenderung memiliki kemampuan untuk membayar kembali pinjamannya. Bank perlu memiliki keyakinan dalam memberikan kredit kepada nasabah melalui analisis kelayakan kredit, evaluasi kemampuan pembayaran, dan pertimbangan lainnya. Oleh karena itu, keberhasilan debitur dalam mengelola usahanya dan mencapai keuntungan menjadi solusi utama untuk melunasi pinjaman yang diberikan oleh bank. Dengan demikian, sebagai penyedia kredit, bank harus menerapkan kebijakan perkreditan yang sesuai untuk mencapai keseimbangan yang adil antara memanfaatkan laba dan memberikan semua pinjaman yang terutang (Djuarni & Ratnasari, 2022).

Dalam konteks ini, penting untuk mengembangkan model penentuan pengajuan kredit yang cerdas dan akurat untuk mengoptimalkan portofolio kredit dan meminimalkan risiko kredit yang mungkin timbul. Perkembangan teknologi dan kemajuan dalam analisis data memungkinkan lembaga keuangan untuk memanfaatkan data penggunaan kartu kredit nasabah sebagai sumber informasi yang berharga. Kartu kredit menjadi salah satu instrumen keuangan yang paling umum digunakan oleh masyarakat modern, mencerminkan kebiasaan pengeluaran, perilaku transaksi, dan kestabilan keuangan nasabah. Salah satu metode yang dapat digunakan adalah melalui teknik data mining. Data mining merupakan suatu proses yang memanfaatkan teknik statistik, matematika, kecerdasan buatan, dan *machine learning* untuk mengekstraksi serta mengidentifikasi

informasi dari sejumlah besar data. Tujuan utamanya adalah untuk menemukan pengetahuan baru. Salah satu teknik data mining yang populer adalah teknik klasifikasi. (Jasmir et al., 2022). Oleh karena itu, penelitian ini akan berfokus pada prediksi kegagalan bayar nasabah berdasarkan perilaku penggunaan kartu kredit dengan membandingkan algoritma *random forest* dan *logistic regression*.

1.2 Rumusan Masalah

Berdasarkan latar belakang dan deskripsi masalah yang telah dipaparkan, maka dapat dirumuskan rumusan masalah penelitian sebagai berikut :

1. Bagaimana pengaruh perilaku penggunaan kartu kredit terhadap kegagalan bayar kartu kredit nasabah?
2. Bagaimana kinerja algoritma *Random Forest* dan *Logistic Regression* dalam melakukan prediksi kegagalan bayar kartu kredit nasabah berdasarkan perilaku penggunaan kartu kredit?
3. Metode manakah yang lebih efektif untuk klasifikasi kegagalan bayar kartu kredit nasabah berdasarkan perilaku penggunaan kartu kredit, *random forest* atau *logistic regression*?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dipaparkan, maka tujuan penelitian ini adalah untuk:

1. Menganalisis pengaruh perilaku penggunaan kartu kredit terhadap kegagalan bayar kartu kredit nasabah.
2. Mengembangkan model klasifikasi menggunakan algoritma *Random Forest* dan *Logistic Regression* untuk memprediksi kegagalan bayar kartu kredit nasabah berdasarkan perilaku penggunaan kartu kredit.
3. Membandingkan kinerja kedua metode *random forest* dan *logistic regression*.

1.4 Manfaat

Penelitian ini diharapkan dapat memberikan manfaat bagi lembaga keuangan dan peneliti, antara lain:

1.4.1 Bagi Perusahaan:

1. Meningkatkan akurasi penilaian kelayakan kredit nasabah, sehingga dapat mengurangi risiko kerugian yang mungkin terjadi.
2. Membantu lembaga keuangan untuk menentukan tingkat suku bunga yang akan dikenakan kepada nasabah.
3. Membantu lembaga keuangan untuk menentukan jangka waktu pinjaman yang akan diberikan kepada nasabah.

1.4.2 Bagi Peneliti :

1. Menambah pengetahuan tentang penggunaan metode pembelajaran mesin untuk klasifikasi kegagalan bayar kredit.
2. Menambah pengetahuan tentang faktor-faktor perilaku penggunaan kartu kredit yang berpengaruh terhadap kegagalan bayar.

BAB II

KAJIAN TEORI

2.1 Kredit

Kartu kredit adalah salah satu alat pembayaran pengganti uang tunai yang dapat digunakan untuk transaksi jual beli barang atau jasa dengan syarat bahwa pelunasan dilakukan dengan sistem cicilan atau sekaligus (Otoritas Jasa Keuangan, 2019). Dalam pasal 1 angka 4 Peraturan Bank Indonesia Nomor 7/52/PBI/2005 dan diubah dengan Peraturan Bank Indonesia Nomor 10/8/PBI/2008, disebutkan bahwa kartu kredit adalah alat pembayaran dengan menggunakan kartu yang dapat digunakan untuk melakukan pembayaran atas kewajiban yang timbul dari suatu kegiatan ekonomi, termasuk transaksi pembelanjaan dan/ atau untuk melakukan penarikan tunai dimana kewajiban pembayaran pemegang kartu dipenuhi terlebih dahulu oleh *acquirer* atau penerbit, dan pemegang kartu berkewajiban melakukan pelunasan kewajiban pembayaran tersebut pada waktu yang disepakati baik secara sekaligus (*charge card*) ataupun secara angsuran.

Dalam proses pembayaran tagihan kartu kredit, jumlah yang harus dibayarkan oleh debitur dihitung dari nilai saldo tagihan ditambah dengan nominal bunga bulanan. Dalam hal ini, tagihan dan bunga (*retail interest*) pada bulan sebelumnya dijadikan pokok pinjaman pada bulan selanjutnya. Pada penggunaan kartu kredit, terdapat tanggal jatuh tempo setiap bulan yang wajib dipatuhi oleh debitur. Apabila terjadi keterlambatan dalam pembayaran tagihan, debitur akan dikenai denda keterlambatan. Selain itu, kartu kredit juga dapat digunakan untuk penarikan uang tunai melalui *teller* bank atau mesin ATM yang tertera logo dari kartu kredit, baik untuk penggunaan di dalam maupun luar negeri.

Dalam pemutusan terkait penyetujuan, nominal, dan jangka waktu kredit, terdapat prinsip-prinsip yang harus dijalankan. Tujuh prinsip tersebut di antaranya adalah *four eye principle*, prinsip one obligor, prinsip konsolidasi eksposur, kepatuhan terhadap regulasi, memenuhi karakteristik analisis kredit, pemutusan kredit menggunakan data yang valid dan akurat, *up-to-date* dan *disclosure information*, *bottom-up approach*, dan pemantauan kredit. Pada prinsip pemantauan kredit, debitur akan dikategorikan sesuai dengan kualitas kredit setiap debitur. Kelompok kategori tersebut di antaranya adalah “Lancar”, “Dalam Perhatian Khusus”, “Kurang Lancar”, “Diragukan”, dan “Macet”. Dengan begitu, bank bisa melakukan langkah antisipasi penurunan kualitas kredit debitur (Otoritas Jasa Keuangan, 2019).

2.2 Data Mining

Data mining merupakan gabungan dari dua kata, yaitu data dan *mining*. Data merujuk pada kumpulan bahan, alat, dan teks yang belum memiliki makna, umumnya bersifat tunggal dan kaku, memerlukan proses pengolahan yang tepat agar dapat menghasilkan informasi (Anas & Delima, 2021). Proses *data mining* sendiri adalah langkah penambangan data untuk menghasilkan informasi bernilai. Secara umum, data mining juga dikenal sebagai suatu proses yang digunakan untuk menggali pengetahuan dan penemuan yang terkandung dalam sebuah *database* (Anas, 2020). Proses ini melibatkan teknik-teknik kompleks seperti pemanfaatan *artificial intelligence*, teknik statistik, ilmu matematika, *machine learning*, dan lain sebagainya. Teknik-teknik canggih ini akan

digunakan untuk mengidentifikasi serta mengekstraksi informasi yang berharga dari sebuah *database* yang besar (Sudarsono et al., 2021). Berdasarkan pengertian *data mining* yang telah dijelaskan sebelumnya, didapatkan garis besar bahwa data mining merupakan pengetahuan yang tersembunyi di dalam database yang di proses untuk menemukan pola dan teknik statistik matematika, kecerdasan buatan, dan machine learning untuk mengekstraksi dan mengidentifikasi informasi pengetahuan dari database tersebut (Utomo & Mesran, 2020).

Dalam data mining, terdapat istilah lain yang memiliki makna serupa dengan data mining, yaitu *Knowledge Discovery in Database* (KDD). Baik data mining maupun *Knowledge Discovery in Database* (KDD) memiliki tujuan yang sama, yaitu menggunakan data yang terdapat dalam basis data, kemudian memprosesnya untuk mendapatkan informasi baru yang bermanfaat. Beberapa orang menganggap data mining sebagai sinonim untuk istilah populer lainnya seperti penemuan pengetahuan dari data atau KDD, sementara yang lain melihat data mining sebagai langkah penting dalam proses penemuan pengetahuan (Utomo & Mesran, 2020). *Knowledge Discovery in Data* KDD, yaitu proses yang sistematis yang memiliki tahapan-tahapan sebagai berikut (Anas & Delima, 2021) :

1. *Data Cleaning* adalah proses pembersihan data dari kesalahan data dan data yang tidak konsisten.
2. *Data Integration* adalah proses untuk penggabungan data dari beberapa sumber.
3. *Data Selection* adalah proses untuk pemilihan data dari basis data yang sesuai dengan tujuan analisa.
4. *Data Transformation* adalah proses pengubahan bentuk data menjadi data yang diinginkan.
5. *Data Mining* adalah proses penting yang menggunakan sebuah metode tertentu mendapatkan pengetahuan dari sekumpulan data.
6. *Pattern Evaluation* adalah proses mengidentifikasi hasil pengolahan data.
7. *Knowledge Presentation* adalah proses merepresentasikan informasi yang dibutuhkan, proses dimana informasi yang telah didapatkan kemudian digunakan oleh pemilik data.

2.3 Classification (Klasifikasi)

Klasifikasi merupakan salah satu tugas penting dalam data mining, dimana suatu pengklasifikasi dibuat berdasarkan kumpulan data latih yang telah diberi kelas sebelumnya (Wibawa et al., 2018). Klasifikasi merupakan suatu proses penemuan model yang menggambarkan dan membedakan kelas data yang bertujuan bisa digunakan untuk memprediksi kelas. Proses klasifikasi melibatkan empat elemen utama, yaitu: (1) Kelas , (2) Predictor, (3) Training dataset, dan (4) Testing dataset (Rohman & Rochcham, 2019). Klasifikasi adalah suatu metode pembelajaran terawasi (Supervised learning) yang berupaya mengidentifikasi hubungan antara atribut input dan atribut target. Tujuan dari klasifikasi adalah meningkatkan keandalan hasil yang dihasilkan dari data (Hendrian, 2018). Seperti yang telah dijelaskan dalam literatur, berbagai teknik klasifikasi telah diajukan. Secara khusus, proses klasifikasi dapat dikelompokkan ke dalam berbagai kategori yang disebut sebagai keputusan berbasis pengklasifikasi. Terdapat beberapa metode klasifikasi, di antaranya yaitu (Wibawa et al., 2018) :

1. *Naive Bayes*

Naive Bayes Classifier adalah suatu metode klasifikasi yang berdasarkan teorema Bayes. Karakteristik utama dari *Naive Bayes Classifier* ini terletak pada asumsi yang sangat kuat (naif) akan independensi dari setiap kondisi atau kejadian.

2. *Support Vector Machine*

SVM (*Support Vector Machine*) merupakan metode mesin pembelajaran yang beroperasi berdasarkan prinsip *Structural Risk Minimization* (SRM), dengan tujuan menemukan *hyperplane* optimal yang dapat memisahkan dua kelas dalam ruang input.

3. Jaringan Saraf Tiruan

Jaringan syaraf tiruan adalah suatu paradigma pengolahan informasi yang terinspirasi oleh sistem sel syaraf dalam biologi, mirip dengan cara otak manusia memproses informasi. Paradigma ini memiliki elemen dasar berupa struktur sistem pemrosesan informasi yang baru. Seperti halnya manusia, Jaringan Syaraf Tiruan mampu belajar dari contoh-contoh yang diberikan. Tujuan pembentukan Jaringan Syaraf Tiruan adalah untuk menyelesaikan masalah tertentu, seperti pengenalan pola atau klasifikasi, melalui proses pembelajaran.

4. *Decision Tree*

Decision tree merupakan algoritma yang paling umum digunakan dalam masalah klasifikasi. Sebuah *decision tree* terdiri dari beberapa simpul yaitu tree's root, internal node dan leafs. Dalam pohon keputusan, setiap simpul internal membagi ruang menjadi dua atau lebih sesuai dengan fungsi diskrit dari input atribut nilai.

5. *Fuzzy*

Logika *fuzzy* adalah logika yang kabur atau mengandung unsur ketidakpastian. Kebenaran dalam logika *fuzzy* dapat dinyatakan dalam derajat kebenaran yang nilainya antara 0 sampai 1. Logika ini memang cenderung lebih praktis untuk digunakan karena sederhana, mudah dimengerti, fleksibel, serta lebih baik dan hemat.

Proses Klasifikasi dapat dilakukan setelah melakukan analisis relevansi yang bertujuan untuk mengidentifikasi atribut yang relevan dalam proses tersebut. Akurasi prediksi dari suatu pengklasifikasi dapat direpresentasikan dalam tabel kontingensi atau matriks kebingungan (*Confusion matrix*), seperti yang terlihat dalam tabel berikut ini (Hendrian, 2018) :

Tabel 2.3.1 Confusion Matrix

	Prediksi (+)	Prediksi (-)
Aktual (+)	TP (True Positives)	FN (False Negatives)
Aktual (-)	FP (False Positives)	TN (True Negatives)

Keterangan :

TP: Jumlah kasus positif yang diklasifikasi sebagai positif

FP : Jumlah kasus negatif yang diklasifikasi sebagai positif

TN : Jumlah kasus negatif yang diklasifikasi sebagai negatif

FN : Jumlah kasus positif yang diklasifikasi sebagai positif

TP dan TN memberikan informasi ketika classifier benar, sedangkan FP dan FN memberitahu ketika classifier salah.

2.4 Logistic Regression

Logistic Regression adalah algoritma klasifikasi *machine learning* yang digunakan untuk memprediksi probabilitas variabel dependen kategoris. Dalam *logistic regression*, variabel yang terikat adalah variabel biner yang berisi data berkode 1 (Ya) atau 0 (Tidak). Metode ini merupakan metode regresi linier umum untuk mempelajari pemetaan dari sejumlah variabel numerik ke variabel biner atau probabilistik (Hasanli & Rustamov, 2019). Model *Logistic Regression* merupakan salah satu jenis model klasifikasi. Menurut Hosmer dan Lemeshow persamaan model regresi logistik adalah :

$$\pi(x_i) = \frac{e^{\beta_0 + \beta_1 x_{1i}}}{1 + e^{\beta_0 + \beta_1 x_{1i}}} \quad [1]$$

Dengan fungsi logit $g(x_i)$, yaitu :

$$g(x_i) = \ln\left[\frac{\pi(x_i)}{1 - \pi(x_i)}\right] = \beta_0 + \beta_1 x_{1i} \quad [2]$$

Dari persamaan [1] dan [2] dapat disederhanakan menjadi :

$$\pi(x_i) = \frac{e^{g(x_i)}}{1 + e^{g(x_i)}} \quad [3]$$

Berbeda dengan regresi linier biasa, *logistic regression* tidak mengasumsikan hubungan antara variabel independen dan dependen secara linier. *Logistic regression* merupakan suatu pendekatan untuk membangun model prediksi, mirip dengan regresi linear atau yang umumnya dikenal dengan istilah *Ordinary Least Squares (OLS) regression*. Perbedaannya terletak pada fokus regresi logistik yang berprediksi variabel terikat dengan skala dikotomi, yaitu skala data nominal dengan dua kategori seperti ya dan tidak, baik dan buruk, atau tinggi dan rendah (Widodo, 2021). Koefisien regresi dalam model *logistic regression* memiliki interpretasi yang berbeda dengan model regresi linier. Koefisien regresi dalam model regresi logistik menunjukkan seberapa besar perubahan probabilitas variabel dependen yang disebabkan oleh perubahan satu unit variabel independen. Algoritma *logistic regression* memiliki beberapa kelebihan dan kekurangan. Kelebihan dari algoritma *logistic regression* antara lain yaitu sangat efektif untuk klasifikasi data biner, mudah diimplementasikan dan diinterpretasikan, serta dapat digunakan pada dataset yang memiliki banyak variabel input. Sedangkan kekurangannya yaitu tidak efektif untuk klasifikasi data yang kompleks dengan banyak variabel input, rentan terhadap overfitting jika digunakan pada dataset yang tidak seimbang, dan tidak dapat mengatasi masalah multicollinearity di antara variabel input. *Logistic regression* ini

dapat digunakan untuk berbagai aplikasi, antara lain yaitu : (1) Prediksi kelayakan kredit, (2) Prediksi *churn* pelanggan, (3) Prediksi penyakit, dan (4) Prediksi perilaku konsumen (Arslan & Al-Haq, 2022).

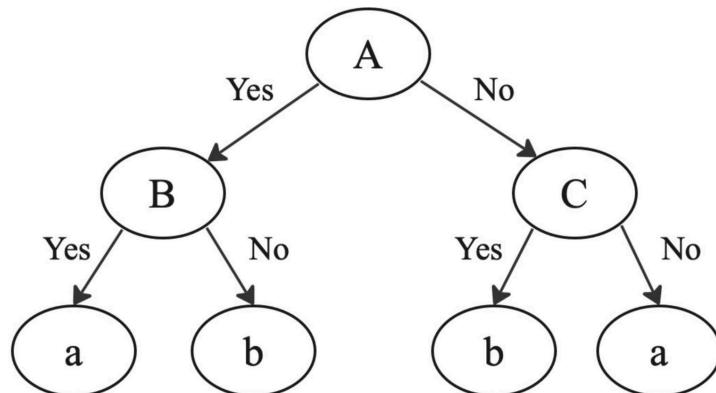
2.5 Decision Tree

Decision tree merupakan suatu algoritma klasifikasi yang mengadopsi bentuk struktur pohon, mirip dengan diagram alir yang digunakan dalam *flowchart*. Pada struktur ini, setiap *node* internal merepresentasikan pengujian pada suatu atribut, setiap cabang atau ranting mencerminkan hasil dari pengujian tersebut, dan setiap *node* daun mencerminkan kelas atau distribusi kelas. *Pada decision tree*, konsep entropi digunakan untuk menentukan bagaimana pohon akan dibagi (*split*) (Marutho, 2019). Setelah pembangunan *Decision Tree*, setiap *instance* diberikan penugasan ke setiap leaf node, di mana j berkisar dari 1 hingga N dengan bobot w_{ij} yang dapat berupa 0 atau 1. Jika setiap uji atribut diketahui untuk i , seluruh populasi awalnya membentuk akar pohon yang berbeda. Untuk menciptakan berbagai cabang pohon, fitur yang dipilih dibedakan berdasarkan karakteristik populasi. Fitur ini disebut sebagai tes yang menghasilkan node anak baru (Wibawa et al., 2018).

Entropi digunakan untuk mengukur sejauh mana suatu sampel tidak murni, dan semakin tinggi nilai entropi, semakin tidak murni sampel tersebut. Rumus yang digunakan untuk menghitung entropy sampel S adalah (Marutho, 2019) :

$$Entropy(S) = \sum_i^c - p_i \log_2 p_i \quad [4]$$

Dimana c adalah jumlah nilai yang terdapat pada atribut target (jumlah kelas). Sementara itu p_i menyatakan rasio antara jumlah sampel di kelas I dengan jumlah semua sampel pada himpunan data.

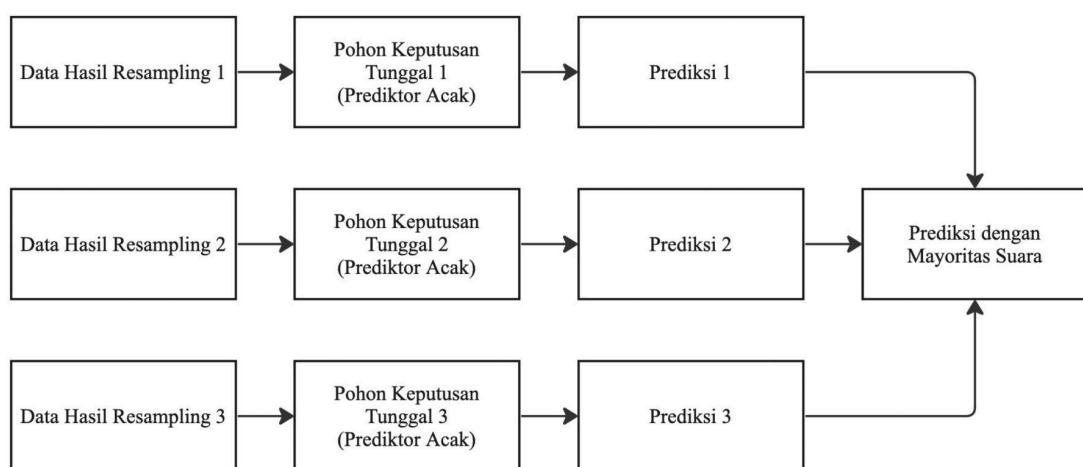


Gambar 2.5.1 Model Susuan *Decision Tree*

2.6 Random Forest

Random forest adalah sebuah metode *ensemble learning* yang pertama kali diusulkan oleh Breiman pada tahun 2001 (Syukron & Subekti, 2018). *Random forest* merupakan metode klasifikasi yang dilakukan dengan mengembangkan metode *decision*

tree berdasarkan pemilihan atribut acak pada setiap *node* untuk menentukan klasifikasi. Dalam proses klasifikasinya, keputusan didasarkan pada suara terbanyak dari pohon keputusan yang dihasilkan (Ratnawati & Sulistyaningrum, 2019). *Random Forest* adalah suatu metode klasifikasi yang terdiri dari kumpulan terstruktur pohon keputusan dimana vektor acak independen didistribusikan secara identik dan setiap pohon keputusan memberikan suara unit untuk kelas paling populer pada masukan x , seperti pada Gambar X (Goel & Abhilasha, 2017).



Gambar 2.6.1 Algoritma Sederhana *Random Forest*

Random Forest adalah sebuah metode yang dikembangkan dari metode CART (*Classification and Regression Trees*), yang pada dasarnya juga merupakan metode atau algoritma dari teknik pohon keputusan. Yang membedakan metode *random forest* dari metode CART adalah *Random forest* menerapkan metode *bootstrap aggregating (bagging)* dan juga seleksi fitur *random* atau bisa disebut *random feature selection* (Binarwati et al., 2017). Ada tiga poin utama dalam metode *Random forest*, tiga poin utama tersebut yaitu (1) melakukan *bootstrap sampling* untuk membangun pohon prediksi; (2) masing-masing pohon keputusan memprediksi dengan prediktor acak; (3) kemudian *random forest* melakukan prediksi dengan mengombinasikan hasil dari tiap tiap pohon keputusan dengan cara *majority vote* untuk klasifikasi atau rata-rata untuk regresi (Primajaya & Sari, 2018).

Random forest telah banyak digunakan baik untuk klasifikasi dan regresi karena kinerjanya yang unggul dan strukturnya yang sederhana (Lin et al., 2017). *Random forest* memiliki kelebihan sebagai berikut (Goel & Abhilasha, 2017) :

1. Memiliki akurasi yang bagus.
2. Relatif kuat terhadap *outliers* dan *noise*.
3. Lebih cepat dari pada *Bagging* dan *Boosting*.
4. Sederhana dan mudah diparalelkan.

Proses untuk membangun *Random forest* adalah sebagai berikut (Goel & Abhilasha, 2017) :

1. Ketika sampel *bootstrap* dibangun dengan mengambil sampel data dengan penggantian setiap pohon keputusan, maka sepertiga dari contoh ditinggalkan.
2. Contoh yang ditinggalkan dikenal sebagai data OOB (*Out of Bag*).
3. Setiap pohon keputusan pada forest memiliki data OOB sendiri yang digunakan untuk estimasi kesalahan masing-masing pohon keputusan yang dikenal sebagai estimasi kesalahan OOB.
4. *Random forest* juga dapat menghitung tingkat kepentingan dan perkiraan variabel. Perkiraan digunakan untuk menghapus dan mengganti nilai-nilai dan *outlier* yang hilang.

BAB III

METODOLOGI

3.1 Sumber Data dan Variabel

Penelitian ini mengambil data yang diambil dari website *UC Irvine Machine Learning repository*, sebuah *website* yang menyediakan ratusan dataset yang ditujukan kepada penelitian dengan menggunakan *machine learning*. Dataset dengan judul “*Default of Credit Card Clients*” merupakan sebuah dataset yang berisi informasi tentang pembayaran default kredit, faktor demografis, data kredit, riwayat pembayaran, dan laporan tagihan dari klien kartu kredit di Taiwan dari April 2005 hingga September 2005. Dataset ini memiliki 25 variabel, di antaranya : ID, limit saldo, gender, pendidikan, status marital, umur, status pembayaran pada bulan April-September, jumlah pembayaran pada bulan April-September, jumlah pembayaran sebelumnya pada bulan April-September, dan status *default payment* yang akan diberikan kepada nasabah. Variabel-variabel yang tersedia ini akan dianalisis untuk melakukan identifikasi pihak bank dalam menentukan apa variabel yang paling berpengaruh terhadap kelayakan pemberian kredit kepada nasabah. Dengan begitu, penelitian ini diharapkan dapat menciptakan penemuan dalam menentukan faktor-faktor penentu kelayakan pemberian kredit.

3.2 Metode Analisis Data

Metode analisis data yang digunakan dalam penelitian ini adalah penentuan kelayakan pemberian kredit kepada nasabah berdasarkan perilaku penggunaan kartu kredit dengan menggunakan dua metode, *random forest* dan *logistic regression*. Dengan dilakukan analisis data, didapatkan klasifikasi penentuan kelayakan pemberian kartu kredit dengan menggunakan dua metode. Metode pertama, *logistic regression*, adalah salah satu metode penambangan data yang dapat digunakan untuk memprediksi nilai variabel target berdasarkan nilai variabel-variabel lain. Dalam kasus dataset yang digunakan, variabel target adalah apakah pelanggan akan default atau tidak. *Logistic regression* cocok digunakan untuk kasus ini karena variabel targetnya adalah kategorikal, yaitu default atau tidak default.

Metode kedua, *random forest*, adalah model yang terdiri dari kombinasi dari pohon dari decision tree (pohon keputusan) yang kemudian digabungkan menjadi satu model. Metode random forest memiliki beberapa keunggulan, yaitu kemampuan meningkatkan akurasi prediksi jika terdapat data yang hilang atau tidak lengkap, kemampuan untuk menangani outliers (data yang nilainya berbeda dengan yang lain) dengan baik, dan kemampuan menyimpan informasi tentang decision tree yang paling penting. Kedua metode adalah metode yang tepat, karena data target merupakan data kategorikal, yaitu hanya memiliki dua nilai, *default* atau tidak *default*. Selain itu, dataset memiliki 23 variabel dan kedua metode dapat menangani variabel berjumlah banyak dengan baik. Juga, kedua metode adalah metode yang toleran terhadap data tidak normal, seperti yang tertera dalam dataset pada penelitian ini.

Analisis data dalam penelitian ini menggunakan Google Colaboratory dan menerapkan bahasa pemrograman Data Python. Python merupakan bahasa pemrograman yang cepat dan efisien, yang membuatnya cocok untuk pengolahan dataset yang besar.

Selain itu, Python juga memiliki beberapa *library* interaktif yang menyediakan beberapa fungsi. Beberapa di antaranya adalah *library* NumPy yang dapat digunakan untuk memproses *array* numerik, *library* Pandas untuk bekerja dengan *data frame*, *library* Scikit-learn untuk pemrosesan data dan evaluasi model *machine learning*. Selain itu, Python dapat melakukan visualisasi data secara menarik dan informatif, sehingga dapat membantu *user* untuk memahami data dengan mudah. Dengan menggunakan Python dalam melakukan penambangan data, diharapkan hasil analisis dapat dilakukan dengan mudah, cepat, dan efisien.

3.3 Langkah Analisis

Dalam proses penambangan dan analisis data pada penelitian ini, langkah-langkah yang diterapkan adalah :

1. Pengumpulan Data
2. Import libraries dan data
3. Preprocessing data
4. Konstruksi model
5. Evaluasi model

BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

4.1.1 Observasi

Observasi dilakukan dengan kegiatan penelitian yang melibatkan pengumpulan informasi yang berkaitan dengan topik penelitian. Informasi ini diperoleh dari *UC Irvine Machine Learning Repository*, sebuah sumber data yang dikenal luas dalam komunitas machine learning. Repository ini menyediakan akses ke berbagai dataset yang bervariasi, mulai dari data medis hingga keuangan, memungkinkan para peneliti untuk mengakses dan menganalisis informasi yang relevan dengan beragam bidang studi. Dengan memanfaatkan dataset yang tersedia di *UC Irvine Machine Learning Repository*, peneliti dapat melakukan analisis mendalam, mengidentifikasi pola, dan membangun model-machine learning untuk memahami fenomena yang mereka teliti dengan lebih baik.

4.1.2 Studi Pustaka

Studi pustaka dilakukan dengan melibatkan pencarian referensi dari sumber-sumber kredibel yang terkait dengan metode yang akan diterapkan dalam penelitian. Proses pengumpulan referensi ini memerlukan penelusuran melalui berbagai literatur ilmiah, jurnal-jurnal terkemuka, publikasi riset terbaru, dan sumber-sumber terpercaya lainnya. Referensi yang dikumpulkan dengan seksama akan dijadikan sebagai dasar penelitian yang kuat dalam laporan penelitian, diperkuat dengan kutipan yang tepat untuk mendukung setiap argumen atau langkah metodologi yang diambil dalam penelitian tersebut. Dengan memastikan keakuratan referensi yang disertakan, laporan penelitian akan memberikan landasan yang kuat dalam menguraikan proses yang dijalani selama penelitian.

4.2 Preprocessing Data

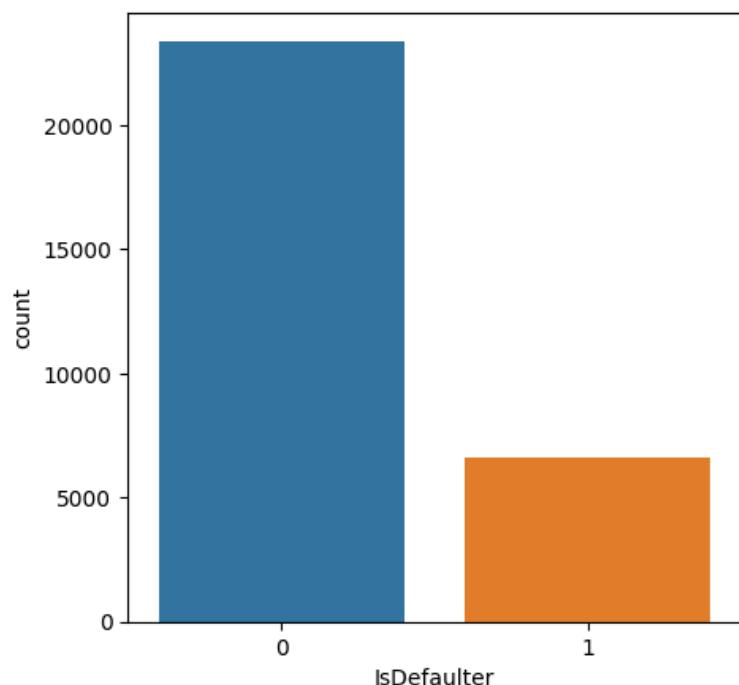
4.2.1 Data Cleaning

Data cleaning adalah proses pembersihan dan mempersiapkan data yang akan diolah agar dapat dianalisis dengan baik. Mulanya, akan diperiksa apakah terdapat data yang terduplikasi. Proses data cleaning dilakukan dengan melihat informasi umum dataset menggunakan metode `df.info()` dan `df.describe()`. Duplikat data diperiksa dengan menghitung panjang `df[df.duplicated()]`. Nama kolom diubah agar lebih deskriptif, seperti mengganti 'default.payment.next.month' menjadi 'IsDefaulter'. Selain itu, penyesuaian nama kolom dilakukan untuk merepresentasikan catatan pembayaran bulanan dari April hingga September 2005. Skala pengukuran status pembayaran dijelaskan, di mana nilai -1 menandakan pembayaran tepat waktu, 1 menunjukkan keterlambatan pembayaran satu bulan, dan seterusnya hingga 9 untuk keterlambatan sembilan bulan ke atas. Upaya ini bertujuan untuk memberikan struktur yang lebih jelas pada dataset terkait catatan pembayaran bulanan dalam periode yang diteliti.

4.2.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) bertujuan untuk memahami kumpulan data dengan merangkum karakteristik utamanya, seringkali memplotnya secara visual. Proses

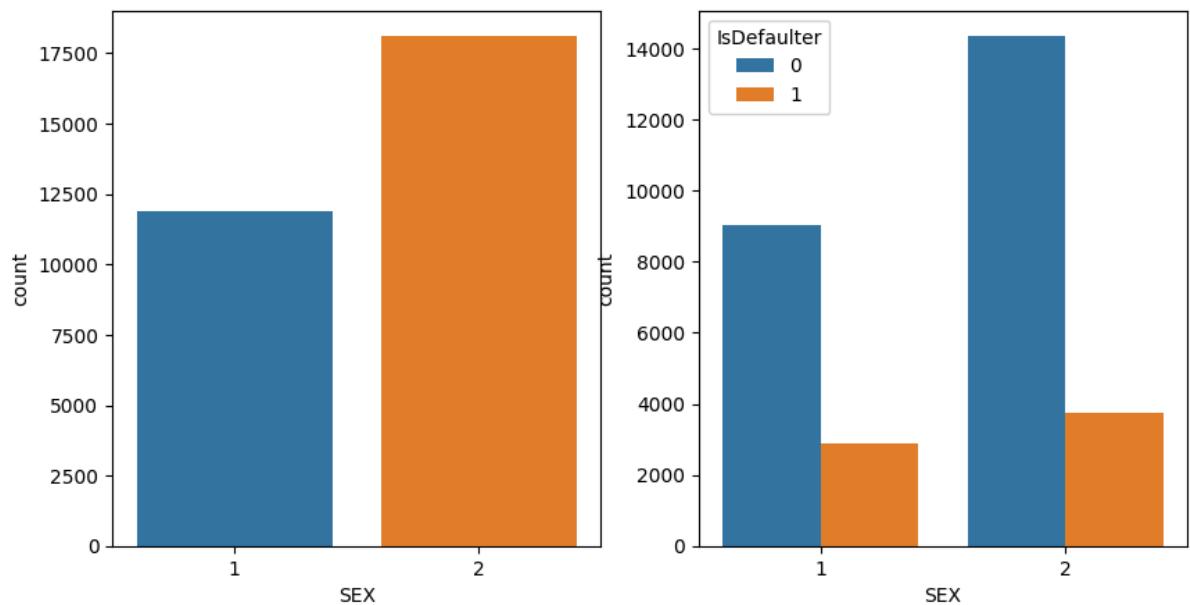
ini sangat penting, terutama saat memasuki tahap pemodelan data untuk menerapkan teknik pembelajaran mesin. Dalam proses ini memanfaatkan data nasabah sebelumnya yang sudah pernah mengajukan kredit dan dijadikan sebagai data *training* yang digunakan untuk menguji data baru (*data testing*). Dalam proses EDA dilakukan pemeriksaan terhadap status pembayaran yang direpresentasikan dalam dataset. Plot yang digunakan pada EDA ini adalah histogram. Pada Gambar 4.2.1 menunjukkan visualisasi distribusi kategori peminjam yang dapat dibedakan berdasarkan nilai pada kolom 'IsDefaulter'. Nilai 1 menandakan bahwa peminjam tersebut memiliki status keterlambatan pembayaran, sedangkan nilai 0 menunjukkan bahwa peminjam tersebut tidak mengalami masalah dalam pembayaran kreditnya. Histogram pada proses EDA ini memberikan gambaran jelas tentang proporsi antara jumlah peminjam yang kurang lancar, dengan mereka yang berhasil mempertahankan keteraturan pembayarannya setiap bulan.



Gambar 4.2.1 Grafik Batang Distribusi Status Peminjam

Kemudian dilakukan pembuatan dua grafik batang (Gambar 4.2.2) untuk menganalisis hubungan antara variabel jenis kelamin ('SEX') dan status pembayaran ('IsDefaulter') dalam dataset. Pada grafik pertama, diperlihatkan distribusi frekuensi antara kategori jenis kelamin. Sedangkan pada grafik kedua, digunakan hue 'IsDefaulter' untuk membagi data berdasarkan status pembayaran, sehingga dapat dilihat sebaran frekuensi pembayaran baik yang lancar (0) maupun yang mengalami keterlambatan (1) berdasarkan jenis kelamin (Laki-laki : 1 dan Wanita : 2). Penggunaan subplot dengan dua kolom digunakan untuk membandingkan langsung antara grafik distribusi jenis kelamin dan grafik yang memperhitungkan status pembayaran. Melalui visualisasi ini, dapat diketahui bahwa jenis kelamin wanita lebih banyak dari pada laki-laki. Wanita lebih banyak memiliki

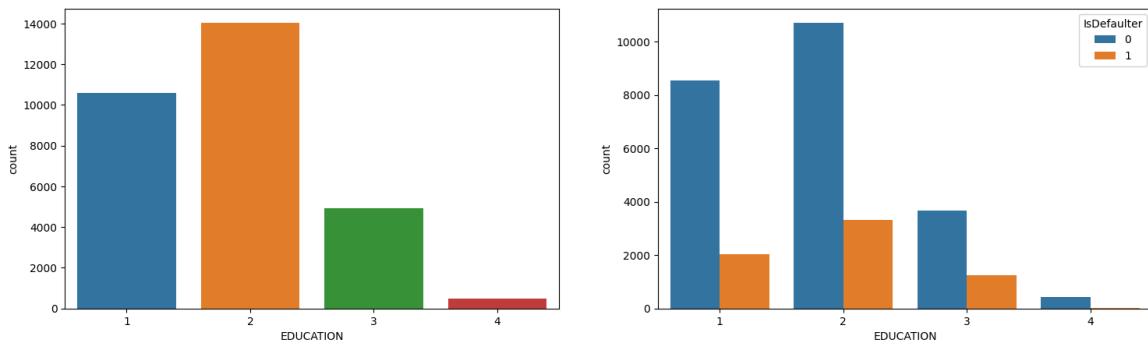
catatan pembayaran yang lancar dibandingkan dengan pria, namun untuk yang mengalami keterlambatan dalam pembayaran selisih pria dengan wanita tidak terlalu jauh.



Gambar 4.2.2 Grafik Batang Hubungan antara Variabel Jenis Kelamin dan Status Pembayaran

Selanjutnya dilakukan pendistribusian kategori pada kolom 'EDUCATION'. Dalam analisis ini, variabel 'EDUCATION' merepresentasikan tingkat pendidikan, dengan kategori sebagai berikut : 1 = *graduate school*; 2 = *university*; 3 = *high school*; 4 = *others*; 5 = *unknown*, 6 = *unknown*. Pada code `df['EDUCATION'].value_counts()`, dilakukan perhitungan frekuensi kemunculan setiap nilai dalam kolom 'EDUCATION'. Hasil dari perhitungan tersebut didapatkan data berikut : *University* (2) = 14.030; *Graduate school* (1) = 10.585; *High school* (3) = 4.917; *Unknown* (5) = 280; *Others* (4) = 123; *Unknown* (6) = 51; *Unknown* (0) = 14. Dengan menggunakan `df.EDUCATION.replace({5: 4, 6: 4, 0: 4})`, dilakukan penggantian nilai-nilai 5, 6, dan 0 dengan nilai 4, yang kemudian diinterpretasikan sebagai kategori pendidikan "*Others*". Hal ini membantu dalam merapikan dan mengelompokkan tingkat pendidikan ke dalam kategori yang lebih sedikit sehingga memudahkan analisis dan interpretasi data.

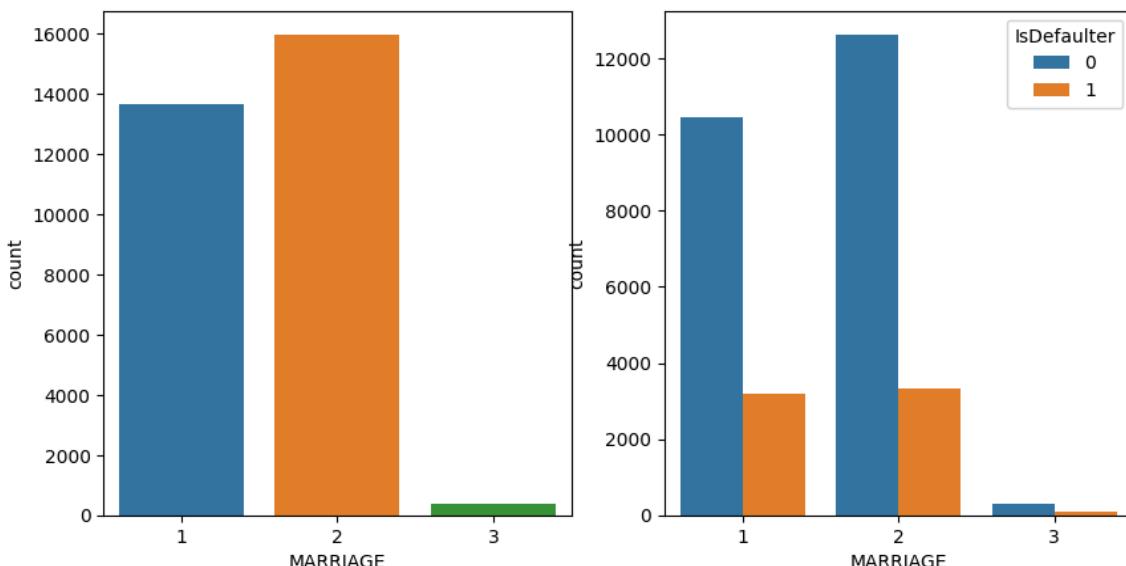
Selanjutnya dilakukan pembuatan dua grafik batang (Gambar 4.2.3) untuk menganalisis hubungan antara tingkat pendidikan ('EDUCATION') dengan status pembayaran ('IsDefaulter') dalam dataset yang direpresentasikan oleh DataFrame (df). Grafik pertama memvisualisasikan distribusi frekuensi tingkat pendidikan tanpa mempertimbangkan status pembayaran. Pada grafik kedua menggunakan parameter hue 'IsDefaulter' sehingga membagi data berdasarkan status pembayaran. Dengan demikian, dapat dilihat sebaran frekuensi tingkat pendidikan yang terkait dengan pembayaran yang lancar (0) dan yang mengalami keterlambatan (1). Melalui visualisasi ini, dapat diketahui bahwa frekuensi yang paling banyak dengan catatan pembayaran lancar maupun dengan catatan yang mengalami keterlambatan pembayaran adalah pada tingkat *university* (2).



Gambar 4.2.3 Grafik Batang Hubungan antara Variabel *Education* dan Status Pembayaran

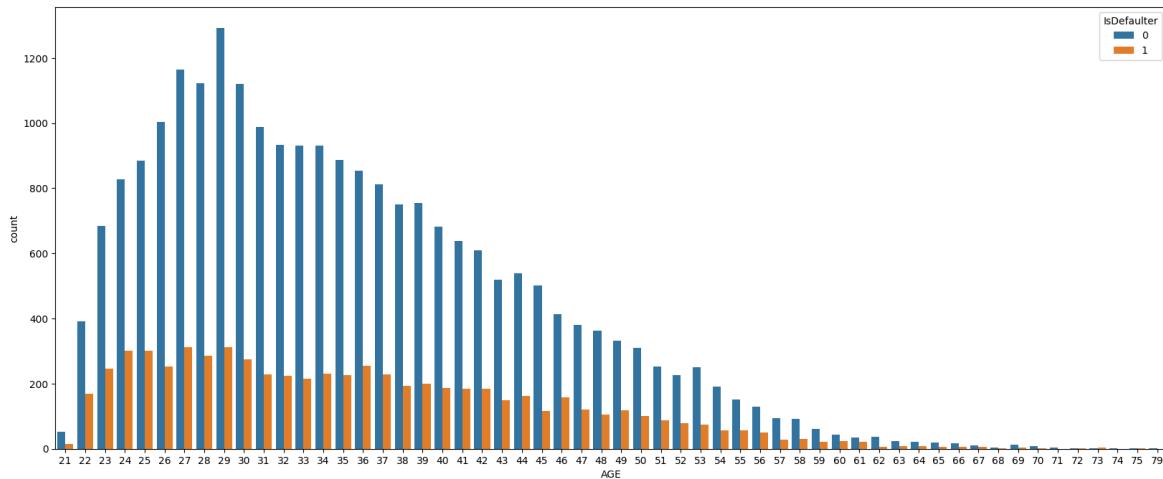
Kemudian dilakukan eksplorasi terhadap kategori dalam kolom 'MARRIAGE'. Kolom ini merepresentasikan status perkawinan dengan beberapa kategori sebagai berikut : 1 = *married*, 2 = *single*, dan 3 = *others*. Dengan menggunakan `df['MARRIAGE'].value_counts()`, dilakukan perhitungan frekuensi kemunculan setiap nilai dalam kolom 'MARRIAGE'. Hasil dari perhitungan sebagai berikut : 2 = 15964, 1 = 13659, 3 = 323, dan 0 = 54. Dengan menggunakan `df.MARRIAGE.replace({0: 3})`, nilai 0 digantikan sehingga status perkawinan yang awalnya tidak terdefinisi menjadi termasuk dalam kategori 3 (*Others*).

Selanjutnya dilakukan pembuatan dua grafik batang (Gambar 4.2.4) untuk menganalisis hubungan antara status perkawinan ('MARRIAGE') dengan status pembayaran ('IsDefaulter'). Grafik pertama memvisualisasikan distribusi frekuensi status perkawinan tanpa mempertimbangkan status pembayaran. Pada grafik kedua menggunakan parameter hue 'IsDefaulter' sehingga membagi data berdasarkan status pembayaran. Dengan demikian, dapat dilihat sebaran frekuensi status perkawinan yang terkait dengan pembayaran yang lancar (0) dan yang mengalami keterlambatan (1). Dapat dilihat bahwa frekuensi yang paling banyak adalah status perkawinan *marriage* dengan catatan pembayaran lancar dan keterlambatan pembayaran yang paling banyak dari pada yang lain.



Gambar 4.2.4 Grafik Batang Hubungan antara Status Perkawinan dan Status Pembayaran

Pada proses EDA yang terakhir, dilakukan pembuatan grafik batang (Gambar 4.2.5) untuk menganalisis distribusi umur ('AGE') dalam hubungannya dengan status pembayaran ('IsDefaulter'). Grafik batang ini memvisualisasikan sebaran frekuensi umur peminjam dengan memperhatikan status pembayaran mereka.

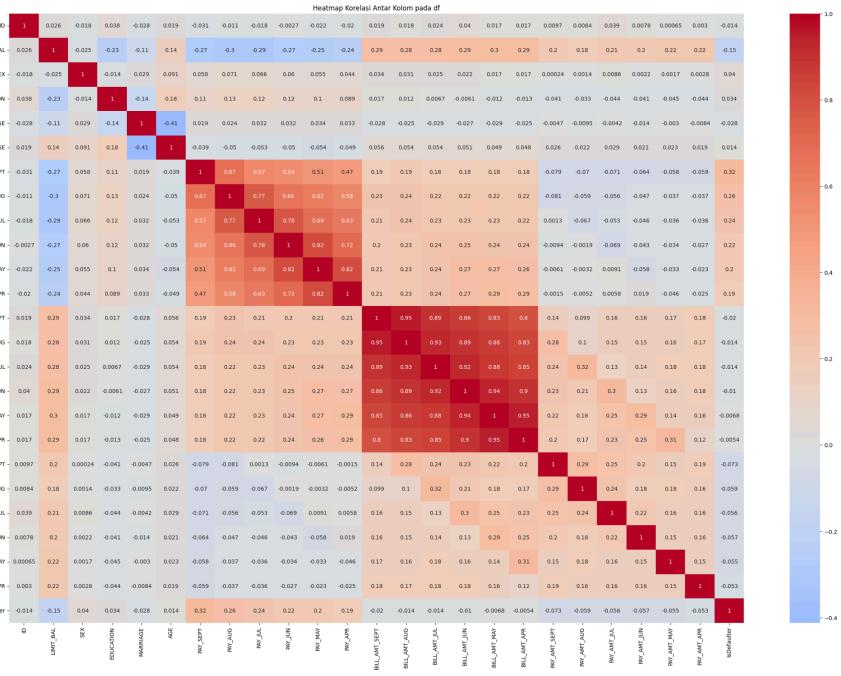


Gambar 4.2.5 Grafik Batang Hubungan antara Variabel Umur dan Status Pembayaran

4.2.3 Label Encoding

Pada proses ini, dilakukan transformasi data dengan menggunakan label encoding. Dalam tahap awal, terdapat pengubahan nilai-nilai dalam kolom 'SEX' dan 'IsDefaulter' menjadi representasi numerik. Misalnya, angka 2 yang sebelumnya mewakili jenis kelamin perempuan dalam kolom 'SEX', diubah menjadi 0, sementara 'Male' direpresentasikan sebagai nilai 1 untuk melambangkan jenis kelamin laki-laki. Pada kolom 'IsDefaulter', 'Yes' dikonversi menjadi 1, mencerminkan status *default*, sedangkan 'No' menjadi 0, menunjukkan bukan *default*.

Setelah proses ini, dataset yang sudah dimodifikasi ditampilkan lima baris pertamanya melalui `df.head()`, memperlihatkan perubahan yang diterapkan setelah label encoding. Langkah selanjutnya melibatkan pembuatan matriks korelasi dengan `df.corr()` untuk mengeksplorasi hubungan antar kolom numerik dalam dataset. Korelasi ini divisualisasikan sebagai heatmap menggunakan Seaborn dengan ukuran besar (30x20) dan anotasi angka pada setiap sel heatmap (parameter `annot=True`). Heatmap ini bertujuan memberikan gambaran visual yang jelas tentang hubungan antar variabel dalam dataset, memudahkan pemahaman terhadap pola korelasi yang mungkin ada di dalamnya.



Gambar 4.2.6 Heatmap dengan menggunakan seaborn

4.2.4 One Hot Encoding

Pada proses *one hot encoding*, terdapat persiapan data untuk analisis lebih lanjut. Mulanya, terdapat pembuatan variabel *dummy* menggunakan metode `pd.get_dummies()`. Proses ini dilakukan pada kolom 'EDUCATION' dan 'MARRIAGE' dalam dataset. Setelah proses ini, jumlah kolom dalam dataset bertambah untuk menggambarkan variasi kategori pada kolom-kolom tersebut, yang dapat membantu dalam analisis lebih lanjut.

Selanjutnya, terjadi penghapusan kolom *dummy* 'EDUCATION_4' dan 'MARRIAGE_3' menggunakan fungsi `df.drop()`. Tindakan ini dilakukan untuk menghindari *dummy variable trap* dalam proses analisis data, sehingga menghasilkan perubahan pada jumlah kolom dataset. Langkah berikutnya adalah menciptakan variabel *dummy* dengan metode yang serupa, namun dengan pengaturan 'drop_first=True' pada kolom 'PAY_SEPT', 'PAY_AUG', 'PAY_JUL', 'PAY_JUN', 'PAY_MAY', dan 'PAY_APR'. Proses ini juga menghasilkan penambahan kolom-kolom baru dalam dataset untuk merepresentasikan variasi kategori dalam kolom tersebut.

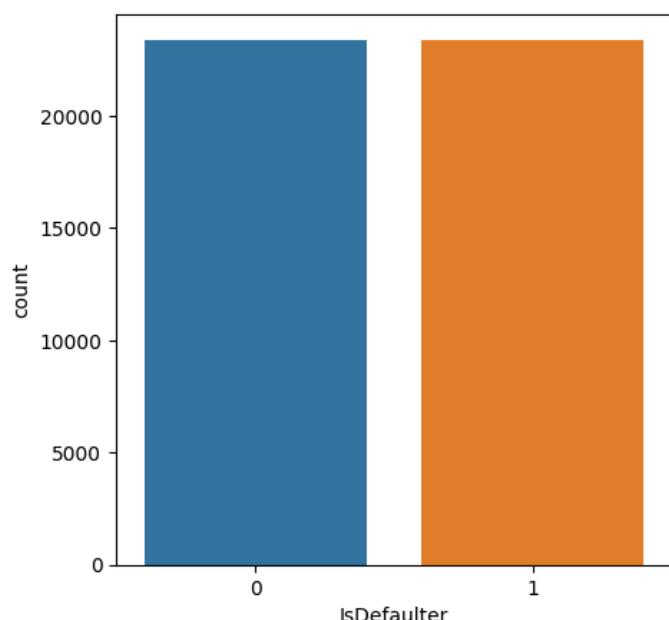
Pada akhirnya, terdapat pengecekan terhadap bentuk dan informasi dataset menggunakan `df.info()` dan `df.shape()`, yang membantu memastikan perubahan-perubahan yang telah diterapkan pada dataset serta memberikan gambaran awal mengenai dimensi data yang akan digunakan dalam analisis selanjutnya.

4.2.5 Handling Imbalance Class

Handling imbalance class merupakan strategi untuk menangani kondisi jumlah sampel dalam kelas-kelas yang berbeda dalam dataset sangat tidak seimbang. Dalam konteks klasifikasi, kelas-kelas yang tidak seimbang merujuk pada skenario di mana jumlah contoh dalam satu kelas jauh lebih banyak atau lebih sedikit dibandingkan kelas lain. Pada proses ini, diimplementasikan teknik SMOTE (Synthetic Minority

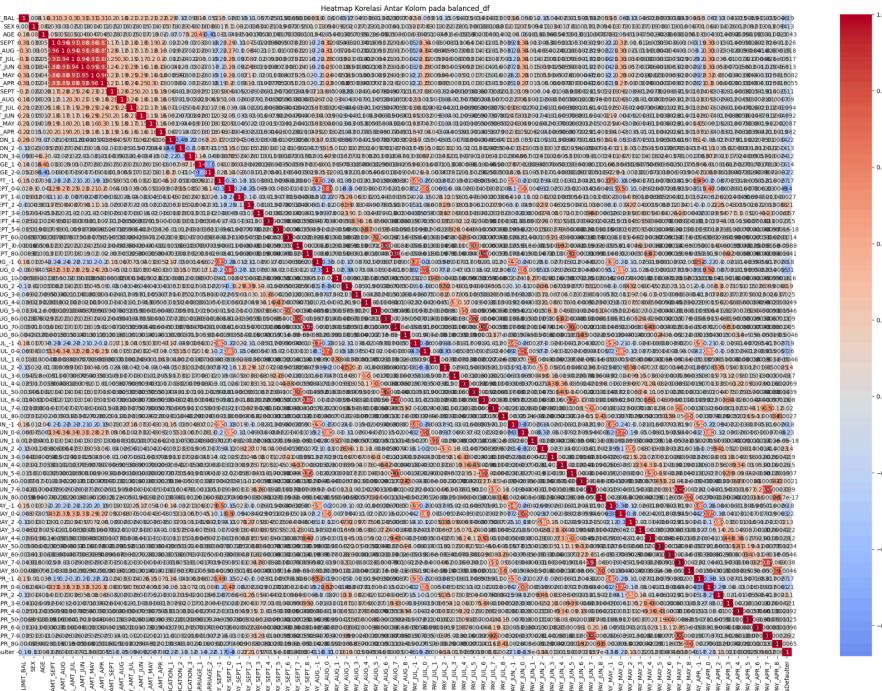
Over-sampling Technique) dari *library* ‘imbalanced-learn’ untuk menangani ketidakseimbangan kelas dalam dataset.

Mulanya, dilakukan impor SMOTE dari *library* ‘imblearn.over_sampling’ dan pembentukan objek SMOTE. Proses SMOTE dilakukan dengan memanggil metode ‘fit_resample()’ pada dataset. Dalam hal ini, variabel prediktor ‘x_smote’ dan variabel target ‘y_smote’ dihasilkan sebagai output dari proses SMOTE yang telah dilakukan terhadap dataset yang semula tidak seimbang. Selanjutnya, dibuat dataframe baru yang telah *balance* ‘balanced_df’ dari dataset yang telah diolah menggunakan teknik SMOTE. Dataframe baru ini memuat kolom-kolom yang telah dihasilkan dari proses SMOTE serta variabel target ‘IsDefaulter’. Untuk memastikan bahwa dataset telah seimbang, dilakukan visualisasi dengan *countplot* menggunakan seaborn, yang menunjukkan distribusi kelas dalam dataset yang telah diimbangi.



Gambar 4.2.7 Grafik Batang untuk dataset yang telah seimbang

Selanjutnya, dilakukan perhitungan matriks korelasi untuk dataframe yang seimbang (balanced_df) dan divisualisasikan sebagai heatmap menggunakan Seaborn. Hal ini dapat membantu dalam memahami hubungan antar variabel dalam dataset yang telah diimbangi setelah penerapan SMOTE. Terakhir, dilakukan pemisahan variabel dependen (y) dan variabel independen (X) dari dataframe yang telah seimbang. Informasi tentang bentuk variabel-variabel ini juga ditampilkan menggunakan print() untuk memberikan gambaran tentang dimensi dataset yang akan digunakan dalam analisis selanjutnya.



Gambar 4.2.8 Heatmap Korelasi antar Kolom pada ‘Balanced_df’

4.2.6 Transformasi Data

Dalam proses transformasi data dilakukan *import library* menggunakan metode *StandardScaler*. Library yang digunakan adalah ‘StandardScaler’ dari modul ‘preprocessing’ yang disediakan oleh *scikit-learn*. Selanjutnya, objek ‘scaler’ dibuat untuk menerapkan standarisasi pada data. Standarisasi adalah suatu proses yang bertujuan untuk mengubah distribusi nilai-nilai pada dataset sehingga memiliki rata-rata 0 dan standar deviasi 1. Kemudian digunakan fungsi ‘fit_transform’ dari objek ‘scaler’ untuk menerapkan transformasi standarisasi pada dataset ‘X’. Data yang sudah ditransformasi akan memiliki skala yang seragam sehingga memudahkan dalam proses analisis dan pemodelan, terutama pada algoritma yang membutuhkan skala yang seragam untuk kinerja yang optimal. Dengan demikian, pada proses ini mempersiapkan data dengan melakukan transformasi standarisasi sebelum diolah lebih lanjut dalam suatu model atau analisis data.

4.2.7 Train Testing Splitting

Train testing splitting merupakan pembagian data menjadi dua bagian, yaitu *data training* dan *data testing*. *Train testing splitting* merupakan langkah penting dalam pengembangan model *machine learning*. Tujuan utamanya adalah untuk melatih model menggunakan *data training* dan kemudian menguji kinerja model pada *data testing*. Hal ini membantu mengevaluasi sejauh mana model dapat menggeneralisasi pola dari *data testing*.

Pada *train testing splitting* dilakukan *import library* yang diperlukan untuk membagi dataset. Library yang digunakan adalah ‘train_test_split’ dari modul ‘model_selection’. Setelah *import library*, fungsi ‘train_test_split’ digunakan untuk membagi dataset menjadi empat bagian: ‘X_train’, ‘X_test’, ‘y_train’, dan ‘y_test’.

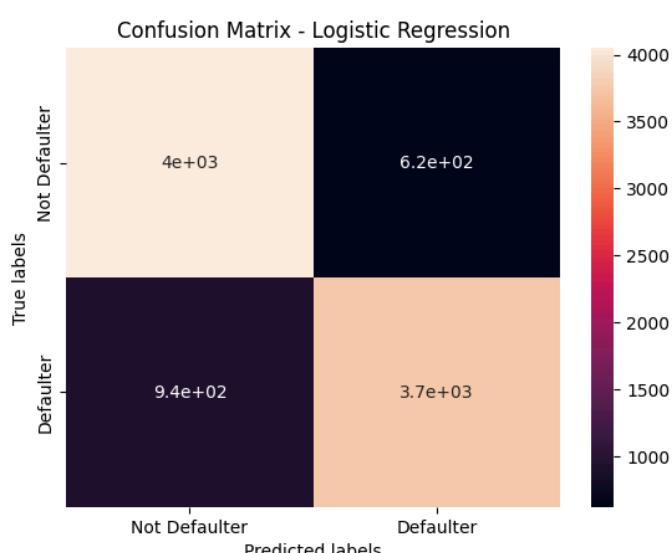
Parameter `test_size = 0.2` menunjukkan bahwa 20% dari dataset akan diambil sebagai *data testing*, sementara 80% menjadi *data training*. Parameter `random_state = 42` digunakan untuk memberikan reproduktibilitas, sehingga hasil pembagian dataset akan tetap sama setiap kali kode dijalankan. Parameter `stratify = y` memastikan bahwa pembagian dataset akan menjaga proporsi kelas pada label, sehingga distribusi kelas pada *data training* dan *data testing* tetap seimbang. Hasil dari pembagian dataset kemudian dicetak, menampilkan bentuk dari `X_train` dan `X_test`, memberikan informasi tentang jumlah baris dan kolom pada masing-masing subset data.

4.3 Konstruksi Model

4.3.1 Logistic Regression

Untuk mengontruksi model menggunakan metode *logistic regression*, digunakan *library* Scikit-learn dalam melakukan analisis prediksi. Proses dimulai dengan impor regresi logistik dari *library* Scikit-learn dan berbagai metrik evaluasi kinerja model seperti akurasi, presisi, recall, F1-score, dan area di bawah kurva ROC (AUC-ROC). Setelah inisialisasi model dengan `LogisticRegression()`, model dilatih menggunakan data *train* (`X_train` dan `y_train`) dengan menggunakan metode `fit()`. Selanjutnya, data *train* digunakan untuk melakukan prediksi terhadap data *test* (`X_test`) menggunakan `predict()`.

Hasil prediksi ini digunakan untuk menghitung sejumlah metrik evaluasi kinerja model seperti akurasi, presisi, recall, F1-score, dan area di bawah kurva ROC menggunakan fungsi-fungsi dari Scikit-learn. Selanjutnya, dilakukan perhitungan dan visualisasi *confusion matrix* untuk melihat performa model dalam mengklasifikasikan sampel ke dalam kelas positif (Defaulter) dan negatif (Not Defaulter). Visualisasi *confusion matrix* ditampilkan dalam bentuk heatmap menggunakan seaborn untuk memudahkan pemahaman terhadap distribusi prediksi yang benar dan salah dari model Regresi Logistik ini.



Gambar 4.3.1 Heatmap visualisasi *confusion matrix* menggunakan metode *Logistic Regression*

4.3.2 Klasifikasi *Random Forest*

Dengan langkah yang sama dalam konstruksi model menggunakan metode regresi logistik, dilakukan untuk mengkonstruksi model dengan metode Random Forest dengan menggunakan *library* Scikit-learn untuk melakukan prediksi dalam konteks analisis dataset. Langkah pertama adalah melakukan impor dari 'RandomForestClassifier' dari *library* Scikit-learn. Selanjutnya, dilakukan *model training* dengan menggunakan data latih ('X_train' dan 'y_train') menggunakan metode 'fit()'. Model ini menggunakan 50 *decision tree* (melalui parameter 'n_estimators') dalam pembentukannya. Setelah proses *train*, model yang telah dibuat digunakan untuk memprediksi kelas dari data *test* ('X_test') menggunakan metode 'predict()'. Prediksi yang dihasilkan digunakan untuk menghitung sejumlah metrik evaluasi kinerja model, seperti akurasi, presisi, recall, F1-score, dan area di bawah kurva ROC (AUC-ROC) menggunakan fungsi-fungsi dari Scikit-learn.

Hasil evaluasi model ditampilkan dalam bentuk perhitungan dan visualisasi *confusion matrix* untuk melihat performa model dalam mengklasifikasikan sampel ke dalam kelas positif (Defaulter) dan negatif (Not Defaulter). Visualisasi confusion matrix ditampilkan dalam bentuk *heatmap* menggunakan Seaborn untuk memudahkan pemahaman terhadap distribusi prediksi yang benar dan salah dari model klasifikasi Random Forest ini.



Gambar 4.3.2 Heatmap visualisasi *confusion matrix* menggunakan metode *Random Forest*

4.3.3 Perbandingan Baseline Model

Proses perbandingan model ini bertujuan untuk membandingkan kinerja dua model klasifikasi, yaitu "*Logistic Regression*" dan "*Random Forest*". Dalam proses ini, metrik evaluasi kinerja model (Tabel 4.3.1) digunakan untuk menilai seberapa baik keduanya dapat melakukan prediksi pada *data testing* yang belum pernah dilihat. Pertama-tama, terdapat beberapa daftar yang menyimpan nilai metrik evaluasi seperti akurasi, presisi, *recall*, *F1-score*, dan AUC-ROC *score* untuk kedua model. Setiap nilai ini dihitung berdasarkan performa model pada *data training* dan *data testing*. Selanjutnya, hasil-hasil tersebut dikompilasi ke dalam sebuah DataFrame menggunakan *library* pandas, yang diberi nama 'compare_df'. DataFrame ini memiliki kolom-kolom yang mencakup nama klasifikasi, nilai akurasi pada *data training* dan *data testing*, presisi, *recall*, *F1-score*, dan

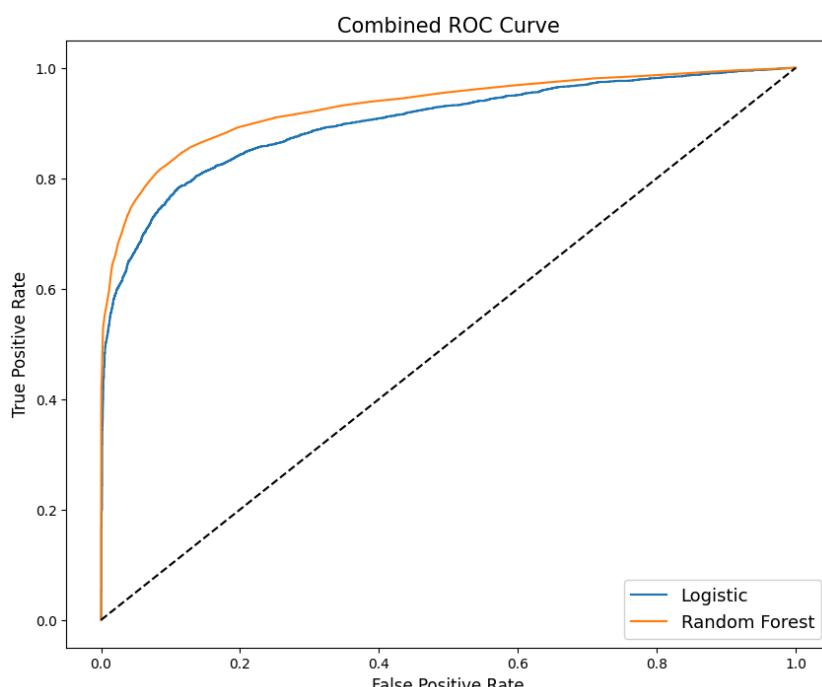
AUC-ROC *score*. Kemudian, DataFrame ‘compare_df’ diurutkan berdasarkan kolom ‘Test Accuracy’ secara menurun. Pengurutan ini bertujuan untuk memberikan gambaran yang jelas tentang kinerja relatif kedua model pada *data testing*. Model dengan tingkat akurasi pada *data testing* yang lebih tinggi akan muncul pertama dalam hasil pengurutan.

Tabel 4.3.1 Metrik Evaluasi Kinerja Kedua Model

	<i>Classifier</i>	<i>Train Accuracy</i>	<i>Test Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1 Score</i>	AUC
1	<i>Random Forest</i>	0.999	0.865	0.826	0.896	0.860	0.867
0	<i>Logistic Regression</i>	0.828	0.832	0.798	0.857	0.827	0.834

4.3.4 Kurva *Receiver Operating Characteristic (ROC)* Gabungan untuk Kedua Model

Pada proses ini, dilakukan pengimporan kurva *Receiver Operating Characteristic (ROC)* dari *library* Scikit-learn untuk melakukan evaluasi performa dua model klasifikasi, yaitu Regresi Logistik (Logistic Regression) dan klasifikasi Random Forest. Proses ini dimulai dengan menghitung prediksi probabilitas dari kelas positif (Defaulter) menggunakan model-model yang telah dilatih sebelumnya pada *data test* (‘X_test’). Selanjutnya, dilakukan visualisasi kurva ROC gabungan dari kedua model dalam satu grafik. Kurva-kurva ROC ini merepresentasikan perbandingan antara tingkat kebenaran positif (*true positive rate*) dan tingkat kesalahan positif (*false positive rate*) pada berbagai ambang batas pemutusan yang digunakan oleh kedua model.



Gambar 4.3.3 Grafik Kurva ROC Gabungan untuk Kedua Model

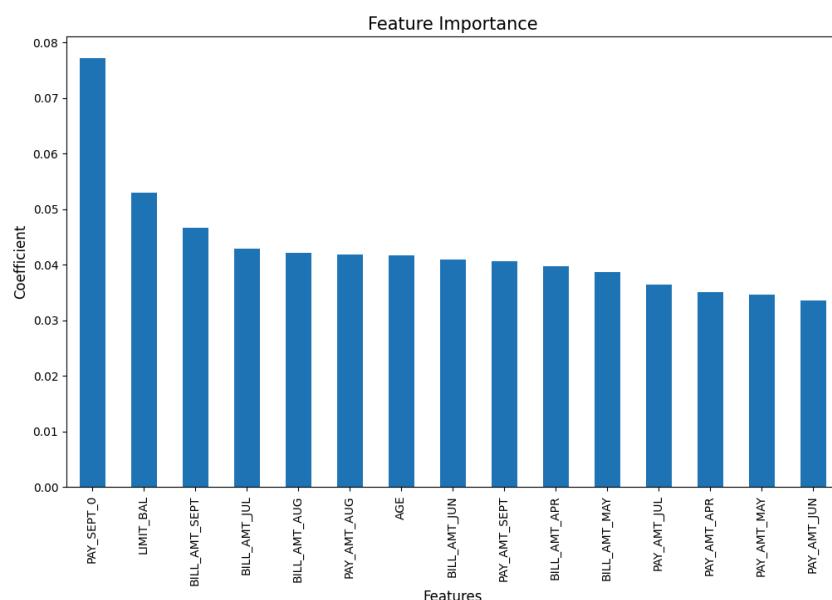
Pada grafik tersebut, ditampilkan kurva ROC untuk kedua model, yaitu Regresi Logistik dan Klasifikasi Random Forest, serta garis referensi diagonal. Pada visualisasi kurva, garis putus-putus diagonal menunjukkan *random classifier* dari model (di mana tidak ada perbedaan antara model dengan *random classifier*). Area di bawah kurva ROC (AUC-ROC) pada grafik ini mencerminkan kualitas prediksi dari masing-masing model, di mana semakin besar nilai AUC-ROC menunjukkan performa yang lebih baik dari model dalam membedakan kelas-kelas yang berbeda. Dengan pembuatan grafik, dapat memberikan pemahaman tentang seberapa baik kedua model dapat memisahkan antara kelas positif dan negatif dalam dataset yang digunakan.

4.4 Peningkatan Performa Model

4.4.1 Feature Importance pada Model Random Forest

Proses *feature importance* bertujuan untuk mengidentifikasi dan memvisualisasikan kepentingan fitur pada model klasifikasi Random Forest. Langkah awal, yaitu melibatkan pembuatan daftar fitur-fitur dari dataset yang telah diimbangi ('balanced_df'), kecuali variabel target 'IsDefaulter'. Selanjutnya, dilakukan perhitungan nilai *feature importance* menggunakan atribut 'feature_importances_' dari model Random Forest ('rfc'). Nilai-nilai ini menunjukkan kontribusi relatif masing-masing fitur terhadap kemampuan model dalam membuat prediksi. Hasil perhitungan yang telah didapat akan ditransformasikan ke dalam sebuah objek Pandas Series dengan indeks yang sesuai dengan nama fitur-fitur.

Untuk visualisasi, digunakan diagram batang (*bar chart*) yang menampilkan 15 fitur teratas berdasarkan kepentingannya secara urut. Grafik ini memberikan gambaran yang jelas tentang peran masing-masing fitur dalam membuat keputusan oleh model, dengan sumbu x menunjukkan fitur-fitur dan sumbu y menampilkan koefisien kepentingan. Hal ini membantu dalam pemahaman lebih lanjut terhadap faktor-faktor yang paling memengaruhi hasil prediksi dari model Random Forest. Pada diagram, fitur dengan kepentingan paling tertinggi adalah status pembayaran pada bulan September dengan nilai 0.



Gambar 4.4.1 Diagram Batang 15 Fitur Teratas Berdasarkan Kepentingannya

4.4.2 Cross Validation dan Hyperparameter Tuning

4.4.2.1 GridSearch pada Logistic Regression

Pada tahap evaluasi dan pengoptimalan model Klasifikasi Regresi Logistik, digunakan teknik *grid search* untuk mencari hyperparameter terbaik. Proses dimulai dengan menentukan kumpulan nilai untuk beberapa hyperparameter yang berbeda yang akan dieksplorasi dalam pencarian grid. Hyperparameter yang dieksplorasi meliputi jenis penalty (misalnya: 'l1', 'l2', 'elasticnet', 'none'), nilai C, dan jumlah iterasi maksimum. Parameter-parameter ini mempengaruhi bagaimana model regresi logistik menyesuaikan bobotnya selama proses pembelajaran.

Selanjutnya, dilakukan pencarian *grid* ('GridSearchCV') menggunakan model regresi logistik dengan konfigurasi parameter yang telah ditentukan sebelumnya. *Grid Search* digunakan untuk mencari kombinasi hyperparameter terbaik yang mengoptimalkan metrik evaluasi, dalam hal ini digunakan metrik *Area Under Curve* (AUC) dari kurva *Receiver Operating Characteristic* (ROC). Setelah proses Grid Search, model optimal yang dihasilkan ('logi_optimal_model') digunakan untuk membuat prediksi terhadap data *train* dan data *test*. Selain itu, dilakukan perhitungan sejumlah metrik evaluasi seperti akurasi, presisi, recall, dan F1-score, serta AUC untuk mengevaluasi kinerja model yang dioptimalkan. Didapatkan hasil :

Tabel 4.4.1 Metrik Evaluasi Performa Model Logistic Regression

<i>Train Accuracy</i>	<i>Test Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1 Score</i>	AUC
0.826	0.833	0.799	0.857	0.827	0.835

Hasil evaluasi kinerja model dicetak dalam bentuk teks menggunakan 'print()'. Selain itu, dilakukan juga perhitungan dari *confusion matrix* untuk data *train* dan data *test*, yang membantu dalam mengevaluasi performa model dalam mengklasifikasikan kelas positif dan negatif. Didapatkan hasil *confusion matrix* data *test* adalah array([[16199,2492],[4001,14690]]) dan *confusion matrix* data *train* adalah array([[4052,621],[940,3733]]).

4.4.2.2 GridSearch pada Random forest

Proses ini bertujuan mencari kombinasi *hyperparameter* yang memberikan performa terbaik pada model *Random Forest* untuk data *train* yang digunakan. Pada proses ini dilakukan penentuan *hyperparameter grid* yang ingin diuji, termasuk jumlah pohon ('n_estimators'), kedalaman maksimum ('max_depth'), jumlah minimum sampel yang diperlukan untuk membagi *node* internal ('min_samples_split'), dan jumlah minimum sampel yang diperlukan di setiap *leaf node* ('min_samples_leaf'). Kemudian dilakukan inisialisasi model *random forest* sebagai model dasar yang akan dioptimalkan. Selanjutnya dilakukan *grid search* dengan 'GridSearchCV' untuk melakukan pencarian *hyperparameter* dengan melibatkan *cross validation* ('cv = 5'). *Grid Search* ini digunakan untuk mencari kombinasi hyperparameter terbaik yang mengoptimalkan metrik evaluasi

dengan menggunakan metrik *Area Under Curve* (AUC) dari kurva *Receiver Operating Characteristic* (ROC).

Setelah proses *Grid Search*, dihasilkan model optimal yang kemudian digunakan untuk melakukan prediksi kelas pada data *test* ('X_test') dan data *train* ('X_train'). Proses prediksi kelas dilakukan dengan memanggil fungsi 'predict' pada 'rfc_optimal_model'. Selanjutnya, probabilitas prediksi untuk kelas positif diambil menggunakan fungsi 'predict_proba'. Kemudian dilakukan perhitungan metrik evaluasi kinerja model seperti, akurasi, presisi, *recall*, *F1-score*, dan AUC *score* yang dapat dilihat pada tabel berikut :

Tabel 4.4.2 Metrik Evaluasi Performa Model *Random Forest*

<i>Train Accuracy</i>	<i>Test Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1 Score</i>	AUC
0.843	0.834	0.799	0.86	0.828	0.836

Pada proses ini juga dilakukan perhitungan untuk mendapatkan *confusion matrix* pada data *train* dan data *test* menggunakan model *Random Forest* yang telah dioptimalkan. *Confusion matrix* ini memberikan gambaran lebih mendalam tentang sejauh mana model dapat mengklasifikasikan instance-data dengan benar. Hasil dari *confusion matrix* data *train* yaitu array([[16560, 2131], [3749, 14942]]) dan *confusion matrix* data *test* yaitu array([[4081, 592], [953, 3720]]).

4.5 Perbandingan Baseline model dan Optimal model

Hasil dari evaluasi pada baseline model dan model yang sudah dioptimalkan dibandingkan menjadi satu tabel untuk mendapatkan informasi model terbaik dalam melakukan prediksi kegagalan bayar kartu kredit suatu nasabah.

Tabel 4.5.1 Metrik Evaluasi Kinerja Model

	<i>Classifier</i>	<i>Train Accuracy</i>	<i>Test Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1 Score</i>	AUC
1	Random Forest	0.999	0.865	0.826	0.895	0.859	0.867
3	Optimal Random Forest	0.843	0.834	0.799	0.860	0.828	0.836
0	Logistic Regression	0.827	0.833	0.798	0.858	0.827	0.835
2	Optimal Logistic Regression	0.826	0.833	0.799	0.857	0.827	0.835

Dari tabel diatas bisa diketahui bahwa model random forest memiliki akurasi yang cukup akurat dalam melakukan prediksi, baik pada baseline model maupun model yang sudah di

optimalkan. meskipun hasil akurasi data latih pada baseline model random forest cukup tinggi hampir mendekati 100% namun model tersebut memiliki selisih akurasi yang cukup jauh dengan akurasi data tes nya. itu menandakan bahwa model tersebut mengalami *overfitting*. Maka dari tabel diatas dapat disimpulkan bahwa model random forest yang sudah dioptimalkan merupakan model yang paling cocok untuk kasus prediksi kegagalan bayar kartu kredit nasabah.

BAB V

KESIMPULAN

Hasil penelitian menunjukkan bahwa perilaku penggunaan kartu kredit memiliki pengaruh terhadap prediksi kegagalan bayar kartu kredit nasabah. Berdasarkan korelasi matriks dan *Feature importance* diketahui bahwa jumlah kredit yang diberikan serta riwayat status pembayaran pada bulan-bulan sebelumnya menjadi fitur yang sangat berpengaruh dalam menentukan kegagalan bayar nasabah dalam membayar kartu kredit.

Dengan menggunakan Logistic Regression dan Random Forest, ditemukan bahwa kedua algoritma mampu mengklasifikasikan kegagalan bayar kartu kredit kredit berdasarkan perilaku penggunaan kartu kredit dengan tingkat evaluasi akurasi, presisi, *recall*, *F1 Score*, serta AUC yang cukup bagus, namun pada *random forest* terlihat baseline model masih overfitting. Setelah dilakukan pengembangan model dengan mencari kombinasi parameter terbaik, didapatkan hasil yang lebih maksimal untuk kedua algoritma model. Random Forest mampu menangani lebih baik kasus prediksi kegagalan bayar kartu kredit nasabah dengan tingkat akurasi 83,4%, *F1 Score* 82,8%, serta AUC 83,6%.

DAFTAR PUSTAKA

- Anas, A. (2020). Penerapan Algoritma *Fp - Growth* Dalam Menentukan Perilaku Konsumen Ghania Mart Muara Bulian. *Jurnal Ilmiah Media Sisfo*, 14(2), 120.
- Anas, A., & Delima, R. H. (2021). Perbandingan Kinerja Algoritma untuk Prediksi Penyakit Jantung dengan Teknik Data Mining. *Jurnal Ilmiah Media Sisfo*, 15(2).
- Arslan, E. S., & Al-Haq, M. A. A. (2022). Logistic regression for credit scoring: A review of the literature. *Expert Systems with Applications*, 180, 113-124.
- Basori, O. R., & Wahyuningsih, S. D. (2018). Analisis Penilaian Prinsip 5C dalam Pemberian Kredit terhadap Non Performing Loan guna Menilai Tingkat Kesehatan Bank pada PT BPR Harta Swadiri Pandaan. *Penelitian Manajemen Terapan (PENATARAN)*, 3(1), 54–63.
- Binarwati, L., Mukhlash, I., dan Soetrisno. (2017). Implementasi Algoritma Genetika untuk Optimalisasi Random Forest Dalam Proses Klasifikasi Penerimaan Tenaga Kerja Baru: Studi Kasus PT.XYZ. *Jurnal Sains dan Data Seni ITS*, 6(2), vol. 6, 78-82.
- Djuarni, W., & Ratnasari, R. (2022). Implementasi Prinsip 5C dalam Menentukan Kelayakan Pemberian Kredit pada Nasabah. *Jurnal Keuangan dan Perbankan Syariah*, 2(2), 99-113.
- Goel, E., & Abhilasha, E. Random Forest: A Review. *Int. J. Adv. Res. Comput. Sci. Softw. Eng*, 7(1), 251–257.
- Hasanli, H., & Rustamov, S. (2019). Sentiment Analysis of Azerbaijani twits Using Logistic Regression, Naive Bayes, and SVM. *IEEE 13th International Conference on Application of Information and Communication Technologies*, 1-7.
- Hendrian, S. (2018). Algoritma Klasifikasi Data Mining untuk Meprediksi Siswa Dalam Memperoleh Bantuan Dana Pendidikan. *Faktor Exacta* 11(3), 266-274.
- Jasmir, dkk. (2022). Klasifikasi Kelayakan Pemberian Kredit Pada Calon Debitur Menggunakan Naive Bayes. *JURIKOM (Jurnal Riset Komputer)*, 9(6), 1833–1839.
- Lin, L., Wang, F., Xie, X., & Zhong, S. (2017). Random forests-based extreme learning machine ensemble for multi- regime time series prediction. *Expert Systems with Applications*, 83, 164– 176.
- Marutho, D. (2019). Perbandingan Metode Naive Bayes, KNN, Decision Tree pada Laporan Water Level Jakarta. *Infokam*, (2).
- Otoritas Jasa Keuangan. (2019). *Buku II Perbankan*.

- Primajaya, A., & Sari, B. N. (2018). Random Forest Algorithm for Prediction of Precipitation. *Indonesian Journal of Artificial Intelligence and Data Mining (IJAIDM)*, 1(1), 27-31.
- Ratnawati, L., & Sulistyaningrum, D. R. (2019). Penerapan Random Forest untuk Mengukur Tingkat Keparahan Penyakit pada Daun Apel. *Jurnal Sains dan Data Seni ITS*, 8(2), 2337-3520.
- Rohman, A., & Rochcham, M. (2019). Komparasi Metode Klasifikasi Data Mining untuk Prediksi Kelulusan Mahasiswa. *Jurnal Neo Teknika*, 5(1).
- Sudarsono, B. G. dkk. (2021). Analisis Data Mining Data Netflix Menggunakan Aplikasi Rapid Miner. *Journal of Business and Audit Information Systems*, 4 (1), 13-21.
- Syukron, A., & Subekti, A. (2018). Penerapan Metode Random Over-Under Sampling dan Random Forest untuk Klasifikasi Penilaian Kredit. *Jurnal Informatika*, 5(2), 175-185.
- Undang-Undang Nomor 10 Tahun 1998 Tentang Perbankan.
- Utomo, D. P. & Mesran. (2020). Analisis Komparasi Metode Klasifikasi Data Mining dan Reduksi Atribut pada Dataset Penyakit Jantung. *Jurnal Media Informatika Budidarma*, 4(2), 437-444.
- Wibawa, A. P. dkk. (2018). Metode-metode Klasifikasi. *Prosiding Seminar Ilmu Komputer dan Teknologi Informasi*, 3(1).
- Widodo, A. M. dkk. (2021). Performansi K-NN, J48, Naive Bayes, dan Regresi Logistik Sebagai Algoritma Pengklasifikasi Diabetes. *Prosiding Seminar Nasional Sistem Informasi dan Teknologi*.

LAMPIRAN

Import Libraries

```
#importing required libraries for data analysis

import pandas as pd

import numpy as np

from numpy import math


# importing libraries for data visualization

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns
```

Import Data

```
df = pd.read_csv('/content/UCI_Credit_Card.csv')

df.head()
```

Preprocessing Data

```
df.info()

df.describe()
```

```
#check for any duplicates

len(df[df.duplicated()])
```

```
df.rename(columns={'default.payment.next.month' : 'IsDefaulter'}, inplace=True)

df.rename(columns={'PAY_0':'PAY_SEPT','PAY_2':'PAY_AUG','PAY_3':'PAY_JUL','PAY_4':'PAY_JU
N','PAY_5':'PAY_MAY','PAY_6':'PAY_APR'},inplace=True)
```

```

df.rename(columns={'BILL_AMT1':'BILL_AMT_SEPT','BILL_AMT2':'BILL_AMT_AUG','BILL_AMT3':'BILL_AMT_JUL','BILL_AMT4':'BILL_AMT_JUN','BILL_AMT5':'BILL_AMT_MAY','BILL_AMT6':'BILL_AMT_APR'}, inplace = True)

df.rename(columns={'PAY_AMT1':'PAY_AMT_SEPT','PAY_AMT2':'PAY_AMT_AUG','PAY_AMT3':'PAY_AMT_JUL','PAY_AMT4':'PAY_AMT_JUN','PAY_AMT5':'PAY_AMT_MAY','PAY_AMT6':'PAY_AMT_APR'},inplace=True)

# We tracked the past monthly payment records from April to September, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . . ; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

```

- EDA (Exploratory Data Analysis)

```

#cek defaulter

# (1=yes, 0=no)

plt.figure(figsize=(5,5))

sns.countplot(x = 'IsDefaulter', data = df)

```

```

#count plot for Sex and with respect to IsDefaulter

# Gender: 1 = male; 2 = female

fig, axes = plt.subplots(ncols=2,figsize=(10,5))

sns.countplot(x = 'SEX', ax = axes[0], data = df)

sns.countplot(x = 'SEX', hue = 'IsDefaulter',ax = axes[1], data = df)

```

```

# Education: 1 = graduate school; 2 = university; 3 = high school; 4 = others

df['EDUCATION'].value_counts()

```

```
#replace values with 5, 6 and 0 to Others
```

```
df.EDUCATION = df.EDUCATION.replace({5: 4, 6: 4, 0: 4})
```

```
#count plot for EDUCATION and with respect to IsDefaulter  
  
fig, axes = plt.subplots(ncols=2, figsize=(18,5))  
  
sns.countplot(x = 'EDUCATION', ax = axes[0], data = df)  
  
sns.countplot(x = 'EDUCATION', hue = 'IsDefaulter', ax = axes[1], data = df)
```

```
#Marriage  
  
#category wise values  
  
# 1 = married; 2 = single; 3 = others  
  
df['MARRIAGE'].value_counts()
```

```
#replace 0 with Others  
  
df.MARRIAGE = df.MARRIAGE.replace({0: 3})
```

```
#count plot for MARRIAGE and with respect to IsDefaulter  
  
fig, axes = plt.subplots(ncols=2, figsize=(10,5))  
  
sns.countplot(x = 'MARRIAGE', ax = axes[0], data = df)  
  
sns.countplot(x = 'MARRIAGE', hue = 'IsDefaulter', ax = axes[1], data = df)
```

```
#values count for Age with respect to IsDefaulter  
  
plt.figure(figsize=(20,8))  
  
sns.countplot(x = 'AGE', hue = 'IsDefaulter', data = df)
```

● Label Encoding

```
#label encoding
```

```
encoders_nums = {"SEX":{2:0,"Male":1}, "IsDefaulter":{"Yes":1, "No":0}} #female change from  
2 to 0  
  
df = df.replace(encoders_nums)  
df.head()
```

```
correlation_matrix = df.corr()  
  
# Membuat heatmap menggunakan seaborn  
  
plt.figure(figsize=(30, 20))  
  
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)  
  
plt.title('Heatmap Korelasi Antar Kolom pada df')  
  
plt.show()
```

- One Hot Encoding

```
#creat dummy variables  
  
df = pd.get_dummies(df, columns = ['EDUCATION', 'MARRIAGE'])  
  
df.shape
```

```
df.info()  
  
df.drop(['EDUCATION_4', 'MARRIAGE_3'], axis=1, inplace=True)  
  
df.shape
```

```
#creating dummy variables by droping firs variable  
  
df = pd.get_dummies(df, columns=['PAY_SEPT', 'PAY_AUG', 'PAY_JUL', 'PAY_JUN',  
'PAY_MAY', 'PAY_APR'], drop_first = True )  
  
df.info()
```

```
df.shape
```

```
df.head()
```

```
df.shape
```

- Handling imbalance class

```
importing SMOTE to handle class imbalance
```

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE()
```

```
# fit predictor and target variable
```

```
x_smote, y_smote = smote.fit_resample(df[(i for i in list(df.describe(include='all').columns) if i != 'IsDefaulter')], df['IsDefaulter'])
```

```
print('Original unbalanced dataset shape', len(df))
```

```
print('Resampled balanced dataset shape', len(y_smote))
```

```
#creating new dataframe from balanced dataset after SMOTE
```

```
balanced_df = pd.DataFrame(x_smote, columns=list(i for i in list(df.describe(include='all').columns) if i != 'IsDefaulter'))
```

```
#adding target variable to new created dataframe
```

```
balanced_df['IsDefaulter'] = y_smote
```

```
#check for class imbalance
```

```
plt.figure(figsize=(5,5))

sns.countplot(x='IsDefaulter', data=balanced_df)
```

```
#shape of balanced dataframe

balanced_df.shape
```

```
balanced_df.drop('ID', axis = 1, inplace = True)
```

```
balanced_df.head()
```

```
correlation_matrix = balanced_df.corr()

# Membuat heatmap menggunakan seaborn

plt.figure(figsize=(30, 20))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)

plt.title('Heatmap Korelasi Antar Kolom pada balanced_df')

plt.show()
```

```
#seperating dependant and independant variabales

X = balanced_df[[i for i in list(balanced_df.describe(include='all').columns) if i != 'IsDefaulter']] 

y = balanced_df['IsDefaulter']

print('X shape : ', X.shape)

print('y shape : ', y.shape)
```

- Data Transformation

```
#importing libraries for data transformation
```

```
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
  
X = scaler.fit_transform(X)
```

- Train Test Splitting

```
#importing libraries for splitting data into training and testing dataset  
  
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42, stratify=y)  
  
print('X train shape : ',X_train.shape)  
  
print('X test shape : ',X_test.shape)
```

Baseline Model

- Logistic Regression

```
from sklearn.linear_model import LogisticRegression  
  
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score,  
roc_auc_score, confusion_matrix, roc_curve, auc
```

```
logi = LogisticRegression()  
  
logi.fit(X_train, y_train)
```

```
y_pred_logi = logi.predict(X_test)  
  
y_train_pred_logi = logi.predict(X_train)
```

```
#getting all scores for logistic regression  
  
train_accuracy_logi = round(accuracy_score(y_train_pred_logi,y_train), 3)  
  
accuracy_logi = round(accuracy_score(y_pred_logi,y_test), 3)
```

```

precision_score_logi = round(precision_score(y_pred_logi,y_test), 3)

recall_score_logi = round(recall_score(y_pred_logi,y_test), 3)

f1_score_logi = round(f1_score(y_pred_logi,y_test), 3)

roc_score_logi = round(roc_auc_score(y_pred_logi,y_test), 3)

print("The accuracy on train data is ", train_accuracy_logi)

print("The accuracy on test data is ", accuracy_logi)

print("The precision on test data is ", precision_score_logi)

print("The recall on test data is ", recall_score_logi)

print("The f1 on test data is ", f1_score_logi)

print("The roc_score on test data is ", roc_score_logi)

```

```

# Get the confusion matrix

labels = ['Not Defaulter', 'Defaulter']

cm_logi = confusion_matrix(y_test, y_pred_logi )

print(cm_logi)

#plot confusion matrix

ax= plt.subplot()

sns.heatmap(cm_logi, annot=True, ax = ax)

# labels, title and ticks

ax.set_xlabel('Predicted labels')

ax.set_ylabel('True labels')

ax.set_title('Confusion Matrix - Logistic Regression')

ax.xaxis.set_ticklabels(labels)

```

```
ax.yaxis.set_ticklabels(labels)
```

- Random Forest Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
#fitting data into Random Forest Classifier  
rfc=RandomForestClassifier(n_estimators=50)  
rfc.fit(X_train, y_train)
```

```
#class prediction of y  
y_pred_rfc=rfc.predict(X_test)  
y_train_pred_rfc=rfc.predict(X_train)
```

```
len(y_pred_rfc)
```

```
#getting all scores for Random Forest Classifier  
train_accuracy_rfc = round(accuracy_score(y_train_pred_rfc,y_train), 3)  
accuracy_rfc = round(accuracy_score(y_pred_rfc,y_test), 3)  
precision_score_rfc = round(precision_score(y_pred_rfc,y_test), 3)  
recall_score_rfc = round(recall_score(y_pred_rfc,y_test), 3)  
f1_score_rfc = round(f1_score(y_pred_rfc,y_test), 3)  
roc_score_rfc = round(roc_auc_score(y_pred_rfc,y_test), 3)  
  
print("The accuracy on train data is ", train_accuracy_rfc)  
print("The accuracy on test data is ", accuracy_rfc)  
print("The precision on test data is ", precision_score_rfc)
```

```

print("The recall on test data is ", recall_score_rfc)

print("The f1 on test data is ", f1_score_rfc)

print("The roc_score on test data is ", roc_score_rfc)

```

```

# Get the confusion matrix for Random Forest Classifier

labels = ['Not Defaulter', 'Defaulter']

cm_rfc = confusion_matrix(y_test, y_pred_rfc )

print(cm_rfc)

#plot confusion matrix

ax= plt.subplot()

sns.heatmap(cm_rfc, annot=True, ax = ax)

# labels, title and ticks

ax.set_xlabel('Predicted labels')

ax.set_ylabel('True labels')

ax.set_title('Confusion Matrix - Random Forest Classifier')

ax.xaxis.set_ticklabels(labels)

ax.yaxis.set_ticklabels(labels)

```

- Model Comparison

```

all_classifiers = ['Logistic Regression', 'Random Forest']

all_train_accuracy = [train_accuracy_logi, train_accuracy_rfc]

all_test_accuracy = [accuracy_logi, accuracy_rfc]

all_precision_score = [precision_score_logi, precision_score_rfc]

all_recall_score = [recall_score_logi, recall_score_rfc]

```

```
all_f1_score = [f1_score_logi, f1_score_rfc]  
all_auc_score = [roc_score_logi, roc_score_rfc]
```

```
compare_df = pd.DataFrame({'Classifier': all_classifiers, 'Train Accuracy': all_train_accuracy, 'Test Accuracy': all_test_accuracy, 'Precision': all_precision_score, 'Recall': all_recall_score, 'F1 Score': all_f1_score, 'AUC': all_auc_score})  
  
compare_df.sort_values(by=['Test Accuracy'], ascending=False)
```

- Combined ROC curve for all the models

```
#importing roc curve  
  
from sklearn.metrics import roc_curve  
  
  
#probabilty prediction of y for all model  
  
y_pred_proba_logi = logi.predict_proba(X_test)[:,1]  
  
y_pred_proba_rfc = rfc.predict_proba(X_test)[:,1]
```

```
fig = plt.figure(figsize=(10,8))  
  
  
  
fpr_logi, tpr_logi, _logi = roc_curve(y_test, y_pred_proba_logi)  
fpr_rfc, tpr_rfc, _rfc = roc_curve(y_test, y_pred_proba_rfc)  
  
  
  
plt.plot(fpr_logi, tpr_logi)  
plt.plot(fpr_rfc, tpr_rfc)  
  
  
plt.plot([0,1], [0,1], color='black', linestyle='--')
```

```
plt.xlabel("False Positive Rate", fontsize=12)  
plt.ylabel("True Positive Rate", fontsize=12)  
plt.title('Combined ROC Curve', fontsize=15)  
  
plt.legend(["Logistic", "Random Forest"], prop={'size':13}, loc='lower right' )
```

- Feature Importance on Random Forest Model

```
features = list(i for i in list(balanced_df.describe(include='all').columns) if i != 'IsDefaulter')  
  
feature_importances_rfc = rfc.feature_importances_  
  
feature_importances_rfc_df = pd.Series(feature_importances_rfc, index=features)  
  
feature_importances_rfc_df.sort_values(ascending=False)[0:15]
```

```
fig = plt.figure(figsize=(12,7))  
  
feature_importances_rfc_df.nlargest(15).plot(kind='bar')  
  
plt.xlabel("Features", fontsize=12)  
plt.ylabel("Coefficient", fontsize=12)  
plt.title('Feature Importance', fontsize=15)  
plt.show()
```

Cross Validation dan Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

- Logistic Regression

```
# penalty in Logistic Regression Classifier
```

```

penalties = ['l1','l2', 'elasticnet', 'none']

# hyperparameter C

C= [0.0001, 0.001, 0.1, 0.5, 0.75, 1, 1.25, 1.5, 5, 10]

# Hyperparameter Grid

param_dict = {'penalty':penalties,
              'max_iter' : [100, 1000, 2500, 5000],
              'C' : C }

```

```

# Create an instance of the Logistic Regression

logi = LogisticRegression()

# Grid search

logi_grid = GridSearchCV(estimator=logi,
                         param_grid = param_dict,
                         cv = 5, verbose=3, n_jobs = -1, scoring='roc_auc')

# fitting model

logi_grid.fit(X_train,y_train)

```

```

logi_grid.best_estimator_
logi_grid.best_params_
logi_optimal_model = logi_grid.best_estimator_

#class prediction of y on train and test

y_pred_logi_grid = logi_optimal_model.predict(X_test)

```

```

y_train_pred_logi_grid = logi_optimal_model.predict(X_train)

# Get the probabilities on train and test

y_pred_prob_logi_grid = logi_optimal_model.predict_proba(X_train)[:,1]

y_train_pred_prob_logi_grid = logi_optimal_model.predict_proba(X_test)[:,1]

```

```

train_accuracy_logi_grid = round(accuracy_score(y_train_pred_logi_grid,y_train), 3)

accuracy_logi_grid = round(accuracy_score(y_pred_logi_grid,y_test), 3)

precision_score_logi_grid = round(precision_score(y_pred_logi_grid, y_test), 3)

recall_score_logi_grid = round(recall_score(y_pred_logi_grid,y_test), 3)

f1_score_logi_grid = round(f1_score(y_pred_logi_grid,y_test), 3)

auc_logi_grid = round(roc_auc_score(y_pred_logi_grid,y_test), 3)

print("The accuracy on train data is ", train_accuracy_logi_grid)

print("The accuracy on test data is ", accuracy_logi_grid)

print("The precision on test data is ", precision_score_logi_grid)

print("The recall on test data is ", recall_score_logi_grid)

print("The f1 on test data is ", f1_score_logi_grid)

print("The auc on test data is ", auc_logi_grid)

```

```

# Get the confusion matrices for train and test

train_cm_logi_grid = confusion_matrix(y_train,y_train_pred_logi_grid)

test_cm_logi_grid = confusion_matrix(y_test,y_pred_logi_grid )

```

```

train_cm_logi_grid

test_cm_logi_grid

```

- Random Forest

```
# Number of trees  
  
n_estimators = [100,150,200]  
  
  
# Maximum depth of trees  
  
max_depth = [10,20,30]  
  
  
# Minimum number of samples required to split a node  
  
min_samples_split = [50,100,150]  
  
  
# Minimum number of samples required at each leaf node  
  
min_samples_leaf = [40,50]  
  
  
# Hyperparameter Grid  
  
param_dict = {'n_estimators' : n_estimators,  
  
              'max_depth' : max_depth,  
  
              'min_samples_split' : min_samples_split,  
  
              'min_samples_leaf' : min_samples_leaf}
```

```
# Create an instance of the RandomForestClassifier  
  
rfc = RandomForestClassifier()  
  
  
# Grid search  
  
rfc_grid = GridSearchCV(estimator=rfc,  
  
                        param_grid = param_dict,
```

```
cv = 5, verbose=2, scoring='roc_auc')

# fitting model

rfc_grid.fit(X_train,y_train)
```

```
rfc_grid.best_estimator_
rfc_grid.best_params_
rfc_optimal_model = rfc_grid.best_estimator_
```

```
#class prediction of y on train and test

y_pred_rfc_grid=rfc_optimal_model.predict(X_test)

y_train_pred_rfc_grid=rfc_optimal_model.predict(X_train)

# Get the probabilities on train and test

y_pred_prob_rfc_grid = rfc_optimal_model.predict_proba(X_train)[:,1]

y_train_pred_prob_rfc_grid = rfc_optimal_model.predict_proba(X_test)[:,1]

len(y_pred_rfc_grid)
```

```
#getting all scores for Random Forest Classifier after CV and Hyperparameter Tunning

train_accuracy_rfc_grid = round(accuracy_score(y_train_pred_rfc_grid,y_train), 3)

accuracy_rfc_grid = round(accuracy_score(y_pred_rfc_grid,y_test), 3)

precision_score_rfc_grid = round(precision_score(y_pred_rfc_grid,y_test), 3)

recall_score_rfc_grid = round(recall_score(y_pred_rfc_grid,y_test), 3)

f1_score_rfc_grid = round(f1_score(y_pred_rfc_grid,y_test), 3)

auc_rfc_grid = round(roc_auc_score(y_pred_rfc_grid,y_test), 3)

print("The accuracy on train data is ", train_accuracy_rfc_grid)
```

```
print("The accuracy on test data is ", accuracy_rfc_grid)  
print("The precision on test data is ", precision_score_rfc_grid)  
print("The recall on test data is ", recall_score_rfc_grid)  
print("The f1 on test data is ", f1_score_rfc_grid)  
print("The auc on test data is ", auc_rfc_grid)
```

```
# Get the confusion matrices for train and test  
  
train_cm_rfc_grid = confusion_matrix(y_train,y_train_pred_rfc_grid)  
  
test_cm_rfc_grid = confusion_matrix(y_test,y_pred_rfc_grid )
```

```
train_cm_rfc_grid  
  
test_cm_rfc_grid
```

```
grid_classifiers = ['Optimal Logistic Regression', 'Optimal Random Forest']  
  
grid_train_accuracy = [train_accuracy_logi_grid, train_accuracy_rfc_grid]  
  
grid_test_accuracy = [accuracy_logi_grid, accuracy_rfc_grid]  
  
grid_precision_score = [precision_score_logi_grid, precision_score_rfc_grid]  
  
grid_recall_score = [recall_score_logi_grid, recall_score_rfc_grid]  
  
grid_f1_score = [f1_score_logi_grid, f1_score_rfc_grid]  
  
grid_auc_score = [auc_logi_grid, auc_rfc_grid]
```

```
grid_compare_df = pd.DataFrame({'Classifier':grid_classifiers, 'Train Accuracy':  
    grid_train_accuracy, 'Test Accuracy': grid_test_accuracy, 'Precision': grid_precision_score,  
    'Recall': grid_recall_score, 'F1 Score': grid_f1_score , 'AUC': grid_auc_score})  
  
all_comparision_df = pd.concat([compare_df, grid_compare_df]).reset_index()  
  
  
all_comparision_df.drop('index', axis=1, inplace=True)
```

```
all_comparision_df.sort_values('AUC', axis=0, ascending=False, inplace=True)
```

```
all_comparision_df
```