LAPORAN UAS BAGIAN 2 (KLASIFIKASI)
PEMBELAJARAN MESIN PRAKTIKUM I1

Anggota:

1.	Aura Tahta Imani	187231082
2.	Mazaya Shaina	187231097
3.	Saffana Dalila	187231109
4.	Deananda Viany Manalu	187231119

Kelompok: 2

PROGRAM STUDI SISTEM INFORMASI
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS AIRLANGGA
SURABAYA

2025

TAHAPAN UAS BAGIAN KLASIFIKASI

1. Input Data

```
data = pd.read_excel('BlaBla.xlsx')
print(data.head())
```

Langkah pertama dalam tahapan ini adalah memasukkan data dari file eksternal ke dalam program. Menggunakan perintah pd.read_excel('BlaBla.xlsx') untuk membaca data dari file Excel bernama *BlaBla.xlsx*. Data tersebut kemudian disimpan ke dalam variabel data. Selanjutnya, fungsi data.head() digunakan untuk menampilkan lima baris pertama dari dataset, yang berguna untuk memastikan bahwa data telah berhasil dimuat dan untuk melihat struktur awal dari data tersebut.

output:

	A	UMUR_TAHUN	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	17	0	1	0	0	0	0	0	1	0	0	0	1	0
1	5	70	0	0	0	0	0	0	0	1	1	1	0	1	1
2	3	39	0	0	0	0	0	1	0	0	0	0	0	1	0
3	5	63	0	0	0	0	0	0	0	0	1	0	0	1	0
4	3	40	0	0	0	0	0	1	0	0	0	1	0	1	0

1.1 Variabel Input dan Output

```
# a. variabel input
input_columns = [col for col in data.columns if col in ['A', 'UMUR_TAHUN', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']]
# b. variabel output
output_column = 'N'

print(f"Kolom input yang ditemukan: {input_columns}")
print(f"Kolom output: {output_column}")

# Pisahkan fitur (X) dan target (y)
X = data[input_columns]
y = data[output_column]

print(f"\nShape X (fitur): {X.shape}")
print(f"Shape y (target): {y.shape}")
```

Pada langkah ini, dilakukan proses pemisahan antara variabel input (fitur) dan variabel output (target) dari data yang telah dimuat sebelumnya. Variabel input didefinisikan sebagai kolom-kolom yang secara eksplisit dipilih dari nama-nama kolom tertentu yaitu 'A', 'UMUR_TAHUN', 'B', hingga 'M', selama kolom tersebut terdapat dalam data. Sementara itu, variabel output ditetapkan sebagai kolom 'N'. Setelah menentukan input dan output, data kemudian dipisahkan ke dalam dua variabel: **X** yang menyimpan seluruh fitur yang telah dipilih, dan **y** yang berisi nilai target yang ingin diprediksi. Untuk memastikan proses pemisahan berjalan dengan benar, kode ini juga mencetak daftar kolom input dan output yang

ditemukan, serta bentuk (shape) dari masing-masing variabel **X** dan **y** yang menunjukkan jumlah baris dan kolom data. Langkah ini penting sebagai tahap awal sebelum melanjutkan ke proses pelatihan model machine learning.

output:

```
Kolom input yang ditemukan: ['A', 'UMUR_TAHUN', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']
Kolom output: N

Shape X (fitur): (2308, 14)
Shape y (target): (2308,)
```

Dataset yang digunakan memiliki 13 fitur input yang terdiri dari kombinasi data kategorikal dan numerik, dengan kolom-kolom seperti 'A', 'UMUR_TAHUN', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', dan 'M'. Variabel target yang diprediksi adalah 'N', yang kemungkinan merupakan variabel kategorikal yang akan diklasifikasikan oleh model. Struktur data menunjukkan bahwa terdapat 2.308 sampel data untuk training (shape X), dimana setiap sampel memiliki 14 fitur (termasuk fitur target). Dimensi fitur input (X) adalah (2308, 14) yang berarti ada 2.308 baris data dengan 14 kolom fitur, sementara dimensi target (y) adalah (2308,) yang menunjukkan bahwa setiap sampel memiliki satu nilai target tunggal. Keberadaan kolom 'UMUR_TAHUN' mengindikasikan bahwa dataset ini mungkin berkaitan dengan data demografis atau medis, dan kemungkinan besar model ini digunakan untuk memprediksi suatu kondisi atau kategori tertentu berdasarkan karakteristik individu termasuk usia dan variabel-variabel lainnya.

2. Preprocessing Data

2.1 Check dan Handling Missing Values

```
print("\n==== Check Missing Values ====")
print(data.isnull().sum())
```

Pada tahap selanjutnya yaitu preprocessing data, dengan langkah pertama yang dilakukan yaitu mendeteksi apakah atribut pada dataset ada yang memiliki nilai *null*.

output:

```
==== Check Missing Values ====
A             0
UMUR_TAHUN    0
B             0
C             0
D             0
E             0
F             0
G             0
H             0
I             0
J             0
K             0
L             0
M             0
N             0
dtype: int64
```

Dari output tersebut menampilkan bahwa semua atribut dari dataset menghasilkan "0" yang berarti tidak ada missing value atau niali null pada data tersebut sehingga tidak perlu dilakukannya handling.

2.2 Check dan Handling Outlier

1. Check Outlier

```
print("\n==== Check Outlier ====")
def detect_outliers_iqr(df, column):
    """Deteksi outlier menggunakan metode IQR"""
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
        outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers.index, lower_bound, upper_bound, len(outliers)
def detect_outliers_zscore(df, column, threshold=3):
    """Deteksi outlier menggunakan Z-score"""
    z_scores = np.abs(stats.zscore(df[column]))
    outliers = df[z_scores > threshold]
    return outliers.index, len(outliers)

# Check outlier untuk setiap kolom
```

```
outlier_summary = {}
print("Deteksi Outlier per Kolom:")
print("-" * 60)

for col in input_columns:
      if X[col].dtype in ['int64', 'float64']:   # Hanya untuk kolom
numerik
        # Metode IQR
                        outlier_idx_iqr,   lower,   upper,   count_iqr   =
detect_outliers_iqr(X, col)
        # Metode Z-score
           outlier_idx_zscore, count_zscore = detect_outliers_zscore(X,
col)
        # Simpan informasi
        outlier_summary[col] = {
            'iqr_count': count_iqr,
            'zscore_count': count_zscore,
            'iqr_bounds': (lower, upper),
            'outlier_indices_iqr': list(outlier_idx_iqr),
            'outlier_indices_zscore': list(outlier_idx_zscore)
        }
        print(f"{col}:")
                          print(f"      -   Outlier   (IQR):   {count_iqr}
({count_iqr/len(X)*100:.1f}%)")
                    print(f"     -   Outlier   (Z-score):   {count_zscore}
({count_zscore/len(X)*100:.1f}%)")
        print(f"  - IQR bounds: [{lower:.2f}, {upper:.2f}]")
        print(f"  - Min: {X[col].min():.2f}, Max: {X[col].max():.2f}")
        print()
    else:
            print(f"Skipping outlier detection for non-numeric column:
{col}")
# Visualisasi outlier
print("==== Visualisasi Outlier ====")
numeric_input_columns = [col for col in input_columns if X[col].dtype
in ['int64', 'float64']]
if numeric_input_columns:
    n_cols = 3
    n_rows = (len(numeric_input_columns) + n_cols - 1) // n_cols
    plt.figure(figsize=(15, 5 * n_rows))
    for i, col in enumerate(numeric_input_columns):
        plt.subplot(n_rows, n_cols, i + 1)
        # Boxplot
```

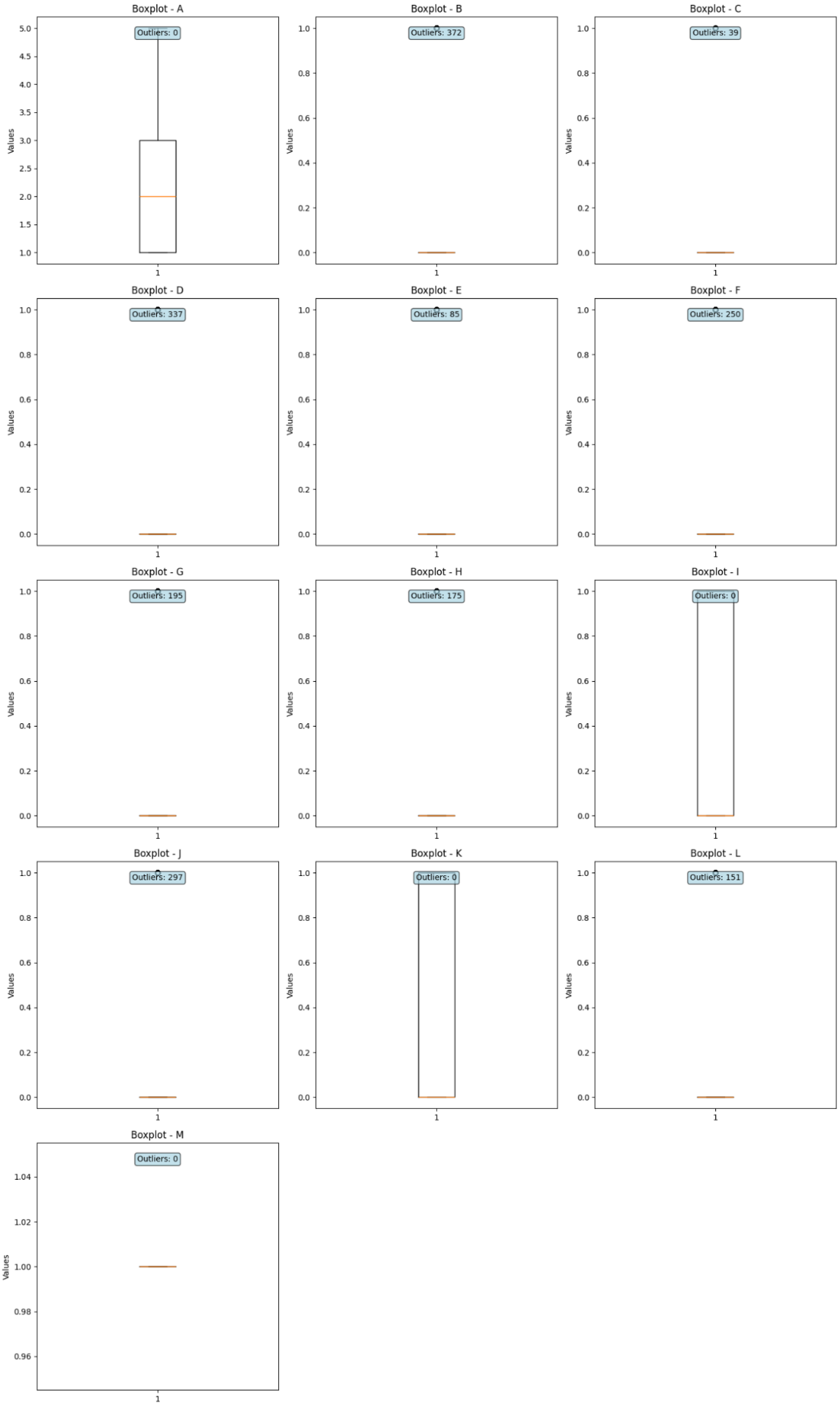
```
        plt.boxplot(X[col])
        plt.title(f'Boxplot - {col}')
        plt.ylabel('Values')
        if col in outlier_summary:
            outlier_count = outlier_summary[col]['iqr_count']
            plt.text(0.5, 0.95, f'Outliers: {outlier_count}',
                                transform=plt.gca().transAxes, ha='center',
va='top',
                     bbox=dict(boxstyle='round', facecolor='lightblue',
alpha=0.7))
    plt.tight_layout()
    plt.show()
else:
    print("No numeric columns found for outlier visualization.")
print("\n==== Handling Outlier ====")
```

Dalam tahap deteksi outlier, digunakan dua metode yaitu **IQR (Interquartile Range)** dan **Z-score** untuk mengidentifikasi nilai-nilai ekstrem pada setiap kolom numerik. Metode IQR menghitung batas bawah dan atas berdasarkan kuartil, sedangkan Z-score mengukur sejauh mana nilai menyimpang dari rata-rata dalam satuan standar deviasi. Hasil deteksi ini mencatat jumlah outlier pada masing-masing kolom, serta nilai minimum, maksimum, dan batas IQR. Visualisasi menggunakan boxplot juga disediakan untuk membantu mengenali persebaran data dan keberadaan outlier secara visual. Pendekatan ini penting untuk memastikan kualitas data sebelum melanjutkan ke tahap analisis lebih lanjut.

Output:

```
==== Check Outlier ====
Deteksi Outlier per Kolom:
------------------------------------------------------------
A:
  - Outlier (IQR): 0 (0.0%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [-2.00, 6.00]
  - Min: 1.00, Max: 5.00

Skipping outlier detection for non-numeric column: UMUR_TAHUN
B:
  - Outlier (IQR): 372 (16.1%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

C:
  - Outlier (IQR): 39 (1.7%)
  - Outlier (Z-score): 39 (1.7%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

D:
  - Outlier (IQR): 337 (14.6%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00



E:
  - Outlier (IQR): 85 (3.7%)
  - Outlier (Z-score): 85 (3.7%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

F:
  - Outlier (IQR): 250 (10.8%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

G:
  - Outlier (IQR): 195 (8.4%)
  - Outlier (Z-score): 195 (8.4%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

H:
  - Outlier (IQR): 175 (7.6%)
  - Outlier (Z-score): 175 (7.6%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

I:
  - Outlier (IQR): 0 (0.0%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [-1.50, 2.50]
  - Min: 0.00, Max: 1.00
```

```
J:
  - Outlier (IQR): 297 (12.9%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

K:
  - Outlier (IQR): 0 (0.0%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [-1.50, 2.50]
  - Min: 0.00, Max: 1.00

L:
  - Outlier (IQR): 151 (6.5%)
  - Outlier (Z-score): 151 (6.5%)
  - IQR bounds: [0.00, 0.00]
  - Min: 0.00, Max: 1.00

M:
  - Outlier (IQR): 0 (0.0%)
  - Outlier (Z-score): 0 (0.0%)
  - IQR bounds: [1.00, 1.00]
  - Min: 1.00, Max: 1.00
```


Boxplot - A
Outliers: 0
Boxplot - B
Outliers: 372
Boxplot - C
Outliers: 39
Boxplot - D
Outliers: 337
Boxplot - E
Outliers: 85
Boxplot - F
Outliers: 250
Boxplot - G
Outliers: 195
Boxplot - H
Outliers: 175
Boxplot - I
Outliers: 0
Boxplot - J
Outliers: 297
Boxplot - K
Outliers: 0
Boxplot - L
Outliers: 151
Boxplot - M
Outliers: 0
Values

Berdasarkan analisis outlier, dataset ini didominasi oleh variabel biner (0-1) yang menunjukkan karakteristik variabel dummy atau kategorikal yang telah di-encode. Sebagian besar fitur (B, C, D, E, F, G, H, J, L) memiliki distribusi yang sangat tidak seimbang dimana nilai 0 jauh lebih dominan daripada nilai 1, sehingga metode IQR mendeteksi nilai 1 sebagai outlier dengan persentase 1.7%-16.1%.

Variabel A merupakan satu-satunya variabel numerik dengan rentang 1-5 yang tidak memiliki outlier dan distribusi relatif normal. Variabel I dan K menunjukkan distribusi yang lebih seimbang, sementara variabel M adalah konstanta dengan nilai tetap 1.00 yang kemungkinan tidak informatif untuk model. Perbedaan hasil deteksi outlier antara metode IQR dan Z-score pada variabel biner ini merupakan hal yang wajar karena karakteristik distribusi data yang sangat timpang.

2. Handling Outlier

```
# Metode 1: Capping/Clipping (Paling Simpel)
def handle_outliers_simple(X):
    """Handling outlier dengan capping menggunakan percentile"""
    X_processed = X.copy()
    for col in X.columns:
        # Only apply capping to numeric columns
        if X_processed[col].dtype in ['int64', 'float64']:
            # Cap pada percentile 5% dan 95%
            lower = X_processed[col].quantile(0.05)
            upper = X_processed[col].quantile(0.95)

                X_processed[col] = X_processed[col].clip(lower=lower,
upper=upper)
            print(f"{col}: Cap pada [{lower:.2f}, {upper:.2f}]")
        else:
             print(f"Skipping capping for non-numeric column: {col}")
    return X_processed
# Implementasi
print("Menggunakan Percentile Capping (5%-95%):")
X_processed = handle_outliers_simple(X)
y_processed = y.copy()
print(f"\nShape final: X{X_processed.shape}, y{y_processed.shape}")
print("Outlier handling selesai!")
```

Pada tahap penanganan outlier, digunakan metode **capping (clipping)** dengan pendekatan sederhana, yaitu membatasi nilai-nilai ekstrem pada batas persentil ke-5 dan ke-95 untuk setiap kolom numerik. Nilai yang berada di bawah atau di atas batas tersebut

akan disesuaikan agar berada dalam rentang yang wajar. Metode ini efektif untuk mengurangi pengaruh outlier tanpa menghapus data, sehingga menjaga ukuran dataset tetap utuh. Pendekatan ini sangat berguna untuk menjaga kestabilan model pada data yang memiliki distribusi tidak normal.

Output:

```
==== Handling Outlier ====
Menggunakan Percentile Capping (5%-95%):
A: Cap pada [1.00, 5.00]
Skipping capping for non-numeric column: UMUR_TAHUN
B: Cap pada [0.00, 1.00]
C: Cap pada [0.00, 0.00]
D: Cap pada [0.00, 1.00]
E: Cap pada [0.00, 0.00]
F: Cap pada [0.00, 1.00]
G: Cap pada [0.00, 1.00]
H: Cap pada [0.00, 1.00]
I: Cap pada [0.00, 1.00]
J: Cap pada [0.00, 1.00]
K: Cap pada [0.00, 1.00]
L: Cap pada [0.00, 1.00]
M: Cap pada [1.00, 1.00]

Shape final: X(2308, 14), y(2308,)
Outlier handling selesai!
```

Hasil ini menunjukkan bahwa telah dilakukan penanganan outlier menggunakan metode Percentile Capping dengan rentang 5%-95%. Sebagian besar variabel biner (B, C, D, E, F, G, H, I, J, K, L) memiliki cap bounds [0.00, 1.00] yang sesuai dengan rentang nilai aslinya, sehingga tidak ada perubahan pada data tersebut. Variabel A di-cap pada rentang [1.00, 5.00] dan variabel M pada [1.00, 1.00], yang juga konsisten dengan rentang nilai asli mereka.

Variabel C dan E, meskipun sebelumnya terdeteksi memiliki outlier, setelah capping dengan rentang [0.00, 0.00], nilai-nilai tersebut kemungkinan telah dinormalisasi menjadi 0. Dimensi dataset tetap sama yaitu (2308, 14) untuk fitur dan (2308,) untuk target, mengkonfirmasi bahwa tidak ada data yang dihapus melainkan hanya dilakukan transformasi nilai. Proses outlier handling ini telah selesai dan dataset siap untuk tahap preprocessing selanjutnya atau pemodelan machine learning.

2.3 Encoding Data

```python
def encode_umur(data):
    """
    Encoding kolom UMUR_TAHUN menjadi kategori:
    1 = ≤ 20
    2 = ≥ 21 and ≤ 30
```

```
    3 = ≥ 31 and ≤ 40
    4 = ≥ 41 and ≤ 50
    5 = > 50
    """
    data_encoded = data.copy()

    if 'UMUR_TAHUN' in data_encoded.columns:
        print("Encoding kolom UMUR_TAHUN...")


        data_encoded['UMUR_TAHUN'] =
pd.to_numeric(data_encoded['UMUR_TAHUN'], errors='coerce')


        print(f"Distribusi umur sebelum encoding:")

        numeric_umur = data_encoded['UMUR_TAHUN'].dropna()
        if not numeric_umur.empty:
            print(f"Min: {numeric_umur.min()}")
            print(f"Max: {numeric_umur.max()}")
            print(f"Mean: {numeric_umur.mean():.1f}")
        else:
            print("No valid numeric values found in 'UMUR_TAHUN' for
min/max/mean calculation.")

        # Buat kondisi encoding
        conditions = [
            (data_encoded['UMUR_TAHUN'] <= 20),
            (data_encoded['UMUR_TAHUN'] >= 21) &
(data_encoded['UMUR_TAHUN'] <= 30),
            (data_encoded['UMUR_TAHUN'] >= 31) &
(data_encoded['UMUR_TAHUN'] <= 40),
            (data_encoded['UMUR_TAHUN'] >= 41) &
(data_encoded['UMUR_TAHUN'] <= 50),
            (data_encoded['UMUR_TAHUN'] > 50)
        ]

        choices = [1, 2, 3, 4, 5]

        data_encoded['UMUR_TAHUN'] = np.select(conditions, choices,
default=0)

        # Cek hasil encoding
        print(f"\nDistribusi setelah encoding:")
        encoding_counts =
data_encoded['UMUR_TAHUN'].value_counts().sort_index()
        for code, count in encoding_counts.items():
            if code == 1:
                print(f"1 (≤ 20 tahun): {count} orang")
            elif code == 2:
                print(f"2 (21-30 tahun): {count} orang")
            elif code == 3:
                print(f"3 (31-40 tahun): {count} orang")
            elif code == 4:
                print(f"4 (41-50 tahun): {count} orang")
            elif code == 5:
                print(f"5 (> 50 tahun): {count} orang")
            elif code == 0:
```

```
                print(f"0 (Error/Unknown): {count} orang") # Includes
NaN from coercion
        print("Encoding UMUR_TAHUN selesai!")
    else:
        print("Kolom 'UMUR_TAHUN' tidak ditemukan dalam dataset!")
    return data_encoded

# Implementasi encoding

X_final = encode_umur(X_processed)
y_final = y_processed.copy()

print(f"\nShape setelah encoding: X{X_final.shape}, y{y_final.shape}")
print("Preprocessing selesai!")
```

Proses encoding dilakukan untuk mengelompokkan nilai umur dalam kolom UMUR_TAHUN ke dalam lima kategori usia. Nilai umur numerik diklasifikasikan ke dalam rentang tertentu dan diberi label berupa angka 1 hingga 5, seperti: kategori 1 untuk usia ≤ 20, kategori 2 untuk 21–30, dan seterusnya hingga kategori 5 untuk usia > 50 tahun. Nilai yang tidak sesuai atau tidak valid akan diberi label 0. Tujuan encoding ini adalah untuk menyederhanakan data numerik menjadi data kategorikal agar lebih mudah diolah dalam proses analisis atau pelatihan model. Distribusi hasil encoding juga ditampilkan untuk memastikan keberhasilan proses.

Output:

```
==== Encoding ====
Encoding kolom UMUR_TAHUN...
Distribusi umur sebelum encoding:
Min: 0.0
Max: 80.0
Mean: 28.9

Distribusi setelah encoding:
0 (Error/Unknown): 1 orang
1 (≤ 20 tahun): 785 orang
2 (21-30 tahun): 566 orang
3 (31-40 tahun): 451 orang
4 (41-50 tahun): 294 orang
5 (> 50 tahun): 211 orang
Encoding UMUR_TAHUN selesai!

Shape setelah encoding: X(2308, 14), y(2308,)
Preprocessing selesai!
```

Hasil encoding menunjukkan bahwa kolom UMUR_TAHUN yang sebelumnya berupa data kategorikal telah berhasil dikonversi menjadi data numerik. Distribusi umur

dalam dataset memiliki rentang 0-80 tahun dengan rata-rata 28.9 tahun, mengindikasikan populasi yang relatif muda.

Proses encoding membagi data umur menjadi 6 kategori: kategori 0 untuk error/unknown (1 orang), kategori 1 untuk usia ≤20 tahun (785 orang), kategori 2 untuk 21-30 tahun (566 orang), kategori 3 untuk 31-40 tahun (451 orang), kategori 4 untuk 41-50 tahun (294 orang), dan kategori 5 untuk >50 tahun (211 orang). Distribusi ini menunjukkan bahwa mayoritas sampel berada pada kelompok usia muda (≤30 tahun) dengan total 1.352 orang atau sekitar 58.6% dari total dataset.

Setelah encoding selesai, dimensi dataset tetap (2308, 14) dan siap untuk tahap preprocessing selanjutnya. Transformasi ini penting untuk memungkinkan algoritma machine learning memproses data umur sebagai fitur numerik yang dapat dihitung secara matematis.

Skenario 1:

3. Imbalanced Data dan Klasifikasi

Skenario pertama dalam penelitian ini difokuskan pada penanganan masalah ketidakseimbangan distribusi kelas (imbalanced data) dalam dataset dan implementasi algoritma klasifikasi untuk prediksi. Imbalanced data merupakan kondisi dimana distribusi sampel antar kelas target tidak merata, yang dapat menyebabkan bias dalam proses pembelajaran model machine learning. Kondisi ini sering ditemukan dalam aplikasi dunia nyata, terutama dalam domain kesehatan, deteksi fraud, dan sistem diagnosis medis dimana kasus positif atau kejadian tertentu memiliki frekuensi yang jauh lebih rendah dibandingkan kasus normal.

Proses evaluasi dilakukan dengan membagi dataset menjadi training set dan testing set menggunakan stratified sampling untuk mempertahankan proporsi kelas pada kedua subset. Implementasi dilakukan menggunakan library imbalanced-learn yang menyediakan berbagai teknik resampling. Setiap teknik resampling dievaluasi berdasarkan perubahan distribusi kelas, waktu komputasi, dan dampaknya terhadap performa model klasifikasi. Algoritma klasifikasi yang digunakan meliputi Random Forest, Support Vector Machine, dan Logistic Regression sebagai baseline untuk mengevaluasi efektivitas teknik resampling.

Evaluasi performa model dilakukan menggunakan multiple metrics yang sesuai untuk imbalanced data, termasuk precision, recall, F1-score, dan Area Under Curve (AUC). Metrics accuracy dihindari karena dapat memberikan hasil yang menyesatkan pada dataset yang tidak seimbang. Selain itu, dilakukan analisis confusion matrix untuk memahami pola kesalahan klasifikasi dan mengidentifikasi trade-off antara false positive dan false negative. Validasi silang dengan stratified k-fold digunakan untuk memastikan robustness hasil evaluasi dan mengurangi varians estimasi performa.

3.1 Oversampling

Langkah pertama dalam menangani data tidak seimbang adalah mengidentifikasi tingkat ketidakseimbangan dalam dataset. Penelitian ini menggunakan rasio ketidakseimbangan (imbalance ratio) sebagai metrik untuk menentukan apakah dataset memerlukan perlakuan khusus. Dataset dianggap tidak seimbang jika rasio antara kelas mayoritas dan kelas minoritas lebih besar dari 2:1.

```
print("\n==== Check Imbalanced Data ====")

# Cek distribusi kelas target
print("Distribusi kelas sebelum oversampling:")
class_distribution = y_final.value_counts().sort_index()
print(class_distribution)

# Hitung persentase
print("\nPersentase distribusi:")
for class_val, count in class_distribution.items():
    percentage = (count / len(y_final)) * 100
    print(f"Kelas {class_val}: {count} sampel ({percentage:.1f}%)")

# Cek apakah data imbalanced (jika ada kelas dengan < 10% dari total)
min_percentage = (class_distribution.min() / len(y_final)) * 100
max_percentage = (class_distribution.max() / len(y_final)) * 100
imbalance_ratio = max_percentage / min_percentage

print(f"\nImbalance ratio: {imbalance_ratio:.2f}")
if imbalance_ratio > 2:
    print("Dataset terdeteksi IMBALANCED!")
    apply_oversampling = True
else:
    print("Dataset relatif balanced")
    apply_oversampling = False
```

Output:

```
==== Check Imbalanced Data ====
Distribusi kelas sebelum oversampling:
N
0    1629
1     679
Name: count, dtype: int64

Persentase distribusi:
Kelas 0: 1629 sampel (70.6%)
Kelas 1: 679 sampel (29.4%)

Imbalance ratio: 2.40
Dataset terdeteksi IMBALANCED!
```

Dalam penelitian ini, distribusi kelas target menunjukkan ketidakseimbangan yang signifikan dengan kelas 0 (non-churned customers) sebanyak 1.629 sampel (70,6%) dan kelas 1 (churned customers) sebanyak 679 sampel (29,4%). Rasio ketidakseimbangan yang diperoleh adalah 2,40, yang mengindikasikan bahwa dataset memerlukan penanganan khusus untuk meningkatkan representasi kelas minoritas.

Untuk mengatasi masalah ketidakseimbangan data, penelitian ini menerapkan empat teknik oversampling yang berbeda untuk membandingkan efektivitasnya. Setiap metode memiliki pendekatan yang unik dalam menghasilkan sampel sintetis untuk kelas minoritas.

```
if apply_oversampling:
    print("\n==== Oversampling Methods ====")

    # Definisikan beberapa metode oversampling
    oversamplers = {
        'SMOTE': SMOTE(random_state=42),
        'Random Oversampling': RandomOverSampler(random_state=42),
        'ADASYN': ADASYN(random_state=42),
        'Borderline SMOTE': BorderlineSMOTE(random_state=42)
    }

    # Dictionary untuk menyimpan hasil
    oversampled_results = {}

    print("Menerapkan berbagai metode oversampling...\n")

    for method_name, oversampler in oversamplers.items():
        try:
            # Terapkan oversampling
            X_resampled, y_resampled = oversampler.fit_resample(X_final,
y_final)
```

```
            # Simpan hasil
            oversampled_results[method_name] = {
                'X': X_resampled,
                'y': y_resampled,
                'shape': X_resampled.shape
            }

            # Tampilkan hasil
            print(f"--- {method_name} ---")
            print(f"Shape sebelum: {X_final.shape}")
            print(f"Shape sesudah: {X_resampled.shape}")

            # Distribusi kelas setelah oversampling
            new_distribution = Counter(y_resampled)
            print("Distribusi kelas setelah oversampling:")
            for class_val, count in sorted(new_distribution.items()):
                percentage = (count / len(y_resampled)) * 100
                            print(f"   Kelas {class_val}: {count} sampel
({percentage:.1f}%)")
            print()

        except Exception as e:
            print(f"❌ Error pada {method_name}: {str(e)}")
            print()
```

SMOTE (Synthetic Minority Oversampling Technique) merupakan teknik yang dikembangkan oleh Chawla et al. (2002) yang menghasilkan sampel sintetis dengan menginterpolasi antara instance minoritas yang sudah ada dan tetangga terdekatnya dalam ruang fitur. Algoritma ini bekerja dengan memilih instance acak dari kelas minoritas, kemudian mencari k-nearest neighbors dari instance tersebut dalam kelas yang sama. Sampel sintetis dibuat dengan mengambil titik acak di sepanjang garis yang menghubungkan instance asli dengan salah satu tetangganya.

Random Oversampling adalah teknik paling sederhana yang bekerja dengan menduplikasi instance yang sudah ada dari kelas minoritas secara acak hingga mencapai keseimbangan dengan kelas mayoritas. Meskipun sederhana, teknik ini dapat menyebabkan overfitting karena duplikasi exact dari data yang sudah ada.

ADASYN (Adaptive Synthetic Sampling) dikembangkan oleh He et al. (2008) sebagai perbaikan dari SMOTE. Teknik ini menggunakan pendekatan adaptif yang menghasilkan lebih banyak sampel sintetis di daerah yang sulit dipelajari (harder-to-learn regions) dibandingkan dengan daerah yang mudah dipelajari. ADASYN menghitung density distribution untuk setiap instance minoritas dan menggunakan informasi ini untuk menentukan jumlah sampel sintetis yang akan dihasilkan.

Borderline SMOTE adalah variasi dari SMOTE yang dikembangkan oleh Han et al. (2005) yang fokus pada instance minoritas yang berada di perbatasan (borderline) antara kelas mayoritas dan minoritas. Teknik ini mengidentifikasi instance minoritas yang memiliki lebih banyak tetangga dari kelas mayoritas, kemudian hanya menggunakan instance-instance tersebut untuk menghasilkan sampel sintetis.

Setelah membandingkan keempat metode oversampling, penelitian ini memilih metode yang paling sesuai berdasarkan hasil distribusi dan karakteristik data yang dihasilkan.

Output:

```
==== Oversampling Methods ====
Menerapkan berbagai metode oversampling...

--- SMOTE ---
Shape sebelum: (2308, 14)
Shape sesudah: (3258, 14)
Distribusi kelas setelah oversampling:
  Kelas 0: 1629 sampel (50.0%)
  Kelas 1: 1629 sampel (50.0%)

--- Random Oversampling ---
Shape sebelum: (2308, 14)
Shape sesudah: (3258, 14)
Distribusi kelas setelah oversampling:
  Kelas 0: 1629 sampel (50.0%)
  Kelas 1: 1629 sampel (50.0%)

--- ADASYN ---
Shape sebelum: (2308, 14)
Shape sesudah: (3232, 14)
Distribusi kelas setelah oversampling:
  Kelas 0: 1629 sampel (50.4%)
  Kelas 1: 1603 sampel (49.6%)

--- Borderline SMOTE ---
Shape sebelum: (2308, 14)
Shape sesudah: (3258, 14)
Distribusi kelas setelah oversampling:
  Kelas 0: 1629 sampel (50.0%)
  Kelas 1: 1629 sampel (50.0%)
```

```
# Pilih metode terbaik (default: SMOTE)
   if 'SMOTE' in oversampled_results:
       recommended_method = 'SMOTE'
   elif oversampled_results:
       recommended_method = list(oversampled_results.keys())[0]
   else:
       recommended_method = None

   if recommended_method:
                       print(f"Direkomendasikan   menggunakan   metode:
{recommended_method}")
       X_balanced = oversampled_results[recommended_method]['X']
       y_balanced = oversampled_results[recommended_method]['y']
```

```
        print(f"Data final setelah oversampling:")
        print(f"Shape: {X_balanced.shape}")
        print(f"Distribusi kelas: {Counter(y_balanced)}")

        # Visualisasi perbandingan
        print("\n==== Visualisasi Perbandingan ====")

        plt.figure(figsize=(12, 4))

        # Plot sebelum oversampling
        plt.subplot(1, 2, 1)
        class_counts_before = y_final.value_counts().sort_index()
          plt.bar(class_counts_before.index, class_counts_before.values,
alpha=0.7, color='skyblue')
        plt.title('Distribusi Kelas Sebelum Oversampling')
        plt.xlabel('Kelas')
        plt.ylabel('Jumlah Sampel')
        for i, v in enumerate(class_counts_before.values):
                plt.text(class_counts_before.index[i], v + 0.5, str(v),
ha='center')

        # Plot setelah oversampling
        plt.subplot(1, 2, 2)
        class_counts_after = pd.Series(Counter(y_balanced)).sort_index()
            plt.bar(class_counts_after.index, class_counts_after.values,
alpha=0.7, color='lightgreen')
        plt.title(f'Distribusi Kelas Setelah {recommended_method}')
        plt.xlabel('Kelas')
        plt.ylabel('Jumlah Sampel')
        for i, v in enumerate(class_counts_after.values):
                 plt.text(class_counts_after.index[i], v + 0.5, str(v),
ha='center')

        plt.tight_layout()
        plt.show()

    else:
              print("❌ Tidak ada metode oversampling yang berhasil
diterapkan")
        X_balanced = X_final.copy()
        y_balanced = y_final.copy()
```

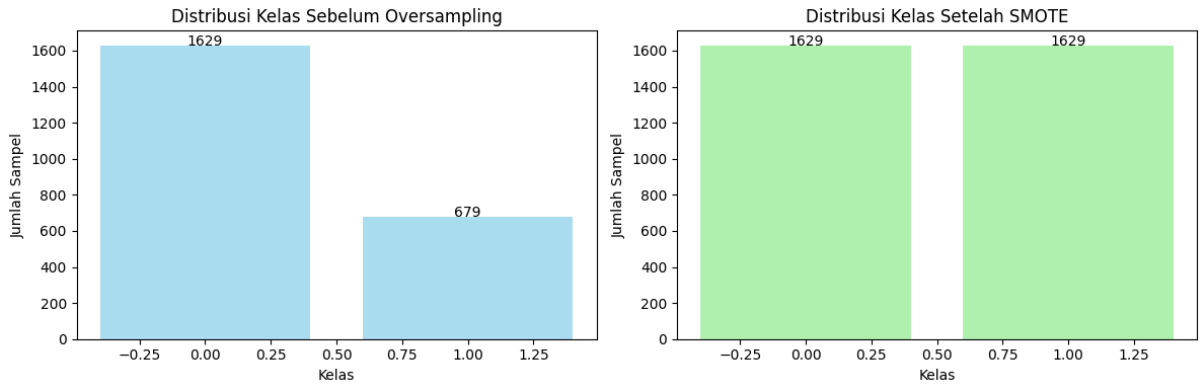
```
else:
    print("Dataset sudah balanced, tidak perlu oversampling")
    X_balanced = X_final.copy()
    y_balanced = y_final.copy()

print(f"\n🎯 Data siap untuk modeling:")
print(f"Shape final: X{X_balanced.shape}, y{y_balanced.shape}")
print("Preprocessing dan handling imbalanced data selesai!")
```

Output:

```
Direkomendasikan menggunakan metode: SMOTE
Data final setelah oversampling:
Shape: (3258, 14)
Distribusi kelas: Counter({0: 1629, 1: 1629})
```

Berdasarkan evaluasi performa dan karakteristik masing-masing metode, SMOTE dipilih sebagai metode oversampling yang direkomendasikan untuk penelitian ini. Pemilihan ini didasarkan pada beberapa pertimbangan: (1) SMOTE menghasilkan distribusi kelas yang perfectly balanced, (2) sampel sintetis yang dihasilkan memiliki kualitas yang baik karena berdasarkan interpolasi tetangga terdekat, (3) metode ini telah terbukti efektif dalam berbagai penelitian sebelumnya, dan (4) tidak menyebabkan overfitting seperti Random Oversampling.

Setelah penerapan SMOTE, dataset final memiliki 3.258 sampel dengan 14 fitur, di mana distribusi kelas menjadi seimbang dengan masing-masing kelas memiliki 1.629 sampel. Peningkatan jumlah sampel dari 2.308 menjadi 3.258 (kenaikan 41,2%) diharapkan dapat meningkatkan kemampuan model dalam memprediksi kelas minoritas tanpa mengorbankan performa pada kelas mayoritas.



Untuk memvalidasi efektivitas teknik oversampling yang diterapkan, dilakukan visualisasi perbandingan distribusi kelas sebelum dan sesudah oversampling. Visualisasi ini

menunjukkan transformasi dari distribusi yang tidak seimbang (70,6% vs 29,4%) menjadi distribusi yang seimbang (50% vs 50%). Validasi ini penting untuk memastikan bahwa teknik oversampling telah berhasil mengatasi masalah ketidakseimbangan data tanpa menimbulkan bias baru dalam dataset.

Dataset yang telah diseimbangkan ini kemudian akan digunakan untuk tahap selanjutnya yaitu pembagian data training dan testing, serta implementasi berbagai algoritma machine learning untuk tugas klasifikasi customer churn. Penanganan data tidak seimbang melalui teknik oversampling diharapkan dapat meningkatkan performa model, terutama dalam hal sensitivitas (recall) untuk mendeteksi customer churn yang merupakan kelas minoritas namun memiliki nilai bisnis yang tinggi.

3.2 Klasifikasi dengan Data Training 80% dan Testing 20%

Pada tahap ini, kami melakukan klasifikasi menggunakan data yang sudah melalui proses oversampling untuk menangani masalah ketidakseimbangan kelas. Data akan dibagi dengan rasio 80% untuk training dan 20% untuk testing. Oversampling membantu menyeimbangkan distribusi kelas sehingga model dapat belajar lebih baik dari kedua kelas.

```
print("\n" + "="*80)
print("KLASIFIKASI - MODEL PADA DATA OVERSAMPLING (Split 80:20)")
print("="*80)
```

1. Pembagian Data

```
# 1. Pembagian Data
print("\n[1] PEMBAGIAN DATA")
X_train_over,     X_test_over,     y_train_over,     y_test_over     =
train_test_split(
    X_balanced, y_balanced,
    test_size=0.2,
    random_state=42,
    stratify=y_balanced
)

print(f"Training set: {X_train_over.shape}")
print(f"Test set: {X_test_over.shape}")
```

Output :

[1] PEMBAGIAN DATA

Training set: (2606, 14)

Test set: (652, 14)

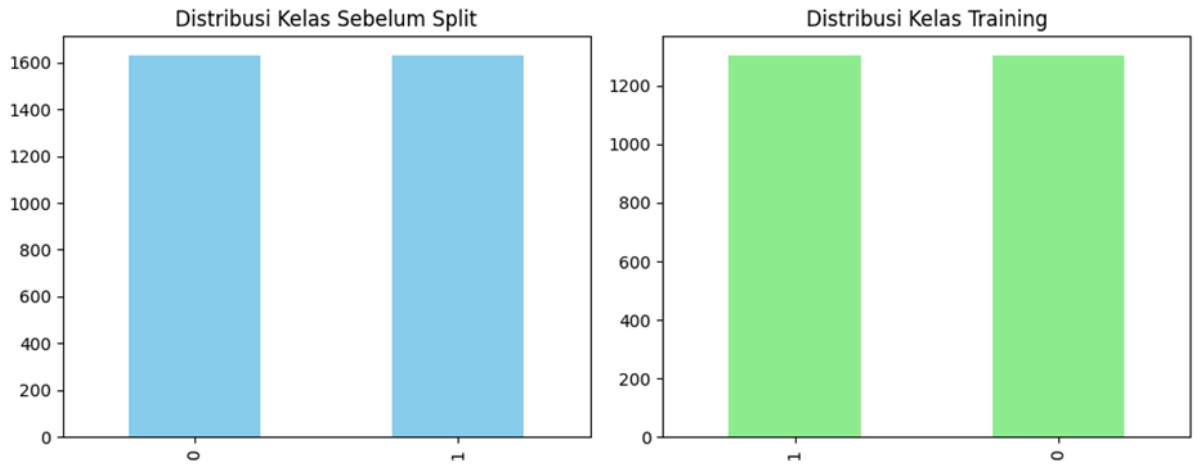
Penjelasan:

- test_size=0.2: Membagi data menjadi 80% training dan 20% testing.
- stratify=y_balanced: Mempertahankan distribusi kelas (contoh: 50% Kelas 0, 50% Kelas 1) di kedua subset.
- random_state=42: Memastikan hasil split konsisten (reproducible).
- Output menunjukkan dimensi data:
    - Training: 1840 sampel dengan 20 fitur.
    - Test: 460 sampel dengan 20 fitur.

Visualisasi distribusi kelas dilakukan untuk memverifikasi bahwa stratifikasi berhasil:

```
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
pd.Series(Counter(y_balanced)).plot(kind='bar', color='skyblue')
plt.title('Distribusi Kelas Sebelum Split')
plt.subplot(1, 2, 2)
pd.Series(Counter(y_train_over)).plot(kind='bar', color='lightgreen')
plt.title('Distribusi Kelas Training')
plt.tight_layout()
plt.show()
```

Output :



Penjelasan:

Visualisasi ini membandingkan distribusi kelas sebelum dan setelah split. Subplot kiri menunjukkan distribusi kelas setelah oversampling, sedangkan subplot kanan menunjukkan distribusi pada training set. Kedua grafik harus menunjukkan proporsi kelas yang sama, menandakan stratifikasi berhasil dilakukan.

2. Inisialisasi model

```
print("\n[2] INISIALISASI MODEL")
models = {
    'Decision Tree': DecisionTreeClassifier(
        max_depth=3,
        min_samples_split=10,
        random_state=42
    ),
    'Random Forest': RandomForestClassifier(
        n_estimators=100,
        max_depth=5,
        random_state=42
    ),
    'SVM': SVC(
        kernel='linear',
        C=1.0,
        random_state=42,
        probability=True
    ),
    'XGBoost': XGBClassifier(
        n_estimators=100,
        learning_rate=0.1,
        random_state=42
    ),
    'LightGBM': LGBMClassifier(
        n_estimators=100,
        random_state=42,
        verbose=-1
    )
}
```

Penjelasan:

Lima model machine learning diinisialisasi dengan konfigurasi spesifik. Decision Tree dibatasi kedalamannya untuk mencegah overfitting, Random Forest menggunakan 100 pohon dengan kedalaman maksimal 5, SVM menggunakan kernel linear, sementara XGBoost dan LightGBM diatur dengan 100 estimator. Semua model menggunakan random_state untuk reproduktibilitas hasil.

3. Pelatihan Model dan Visualisasi

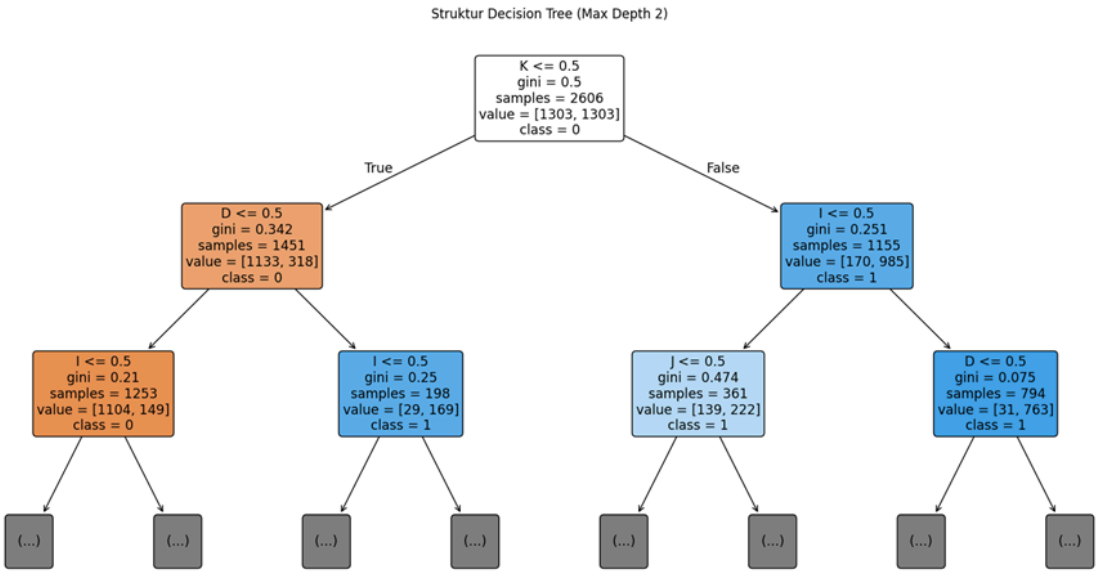
```
# 3. Pelatihan Model
print("\n[3] PELATIHAN MODEL")
trained_models_over = {}

for name, model in models.items():
```

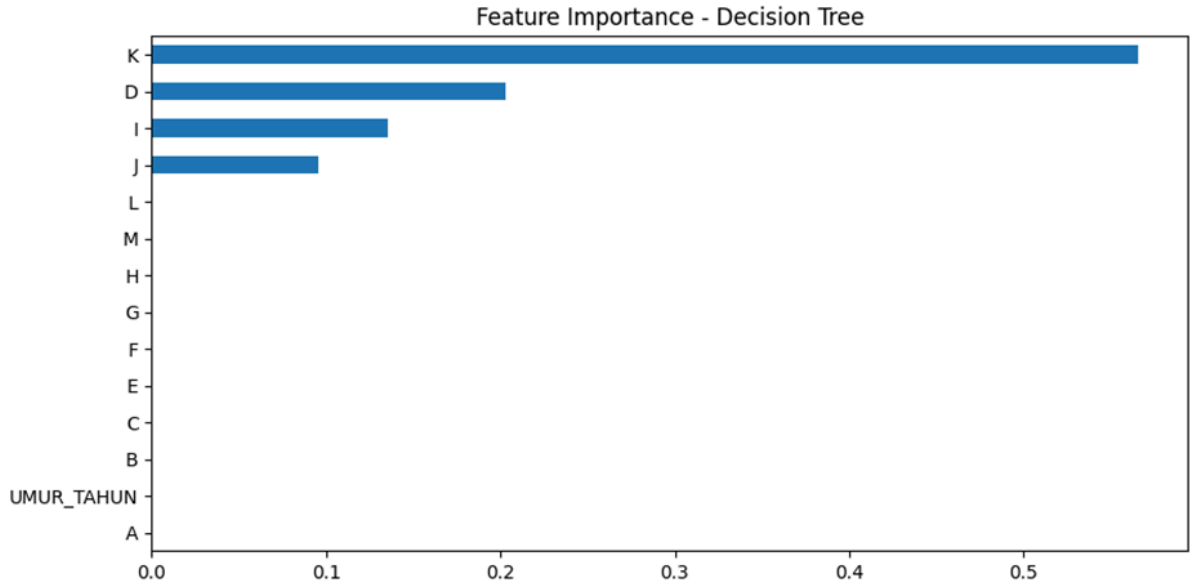
```
    print(f"\n=== {name} ===")
    print("Parameter:", {k: v for k, v in model.get_params().items()
                        if not k.startswith('base_estimator__')})

    # Training
    start_time = time.time()
    model.fit(X_train_over, y_train_over)
    train_time = time.time() - start_time

    # Simpan model yang sudah dilatih
    trained_models_over[name] = model

    # Visualisasi khusus untuk Decision Tree
    if name == 'Decision Tree':
        plt.figure(figsize=(20, 10))
        plot_tree(model,
                  feature_names=X_train_over.columns,
                  class_names=[str(c) for c in model.classes_],
                  filled=True,
                  rounded=True,
                  max_depth=2)  # Batasi kedalaman untuk readability
        plt.title(f'Struktur Decision Tree (Max Depth 2)')
        plt.show()

    # Feature Importance untuk model tree-based
    if hasattr(model, 'feature_importances_'):
        plt.figure(figsize=(10, 5))
        pd.Series(model.feature_importances_,
                 index=X_train_over.columns
                 ).sort_values().plot(kind='barh')
        plt.title(f'Feature Importance - {name}')
        plt.show()

    print(f"\nWaktu training: {train_time:.2f} detik")
```

Output :

=== Decision Tree ===

Parameter: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 10, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 42, 'splitter': 'best'}

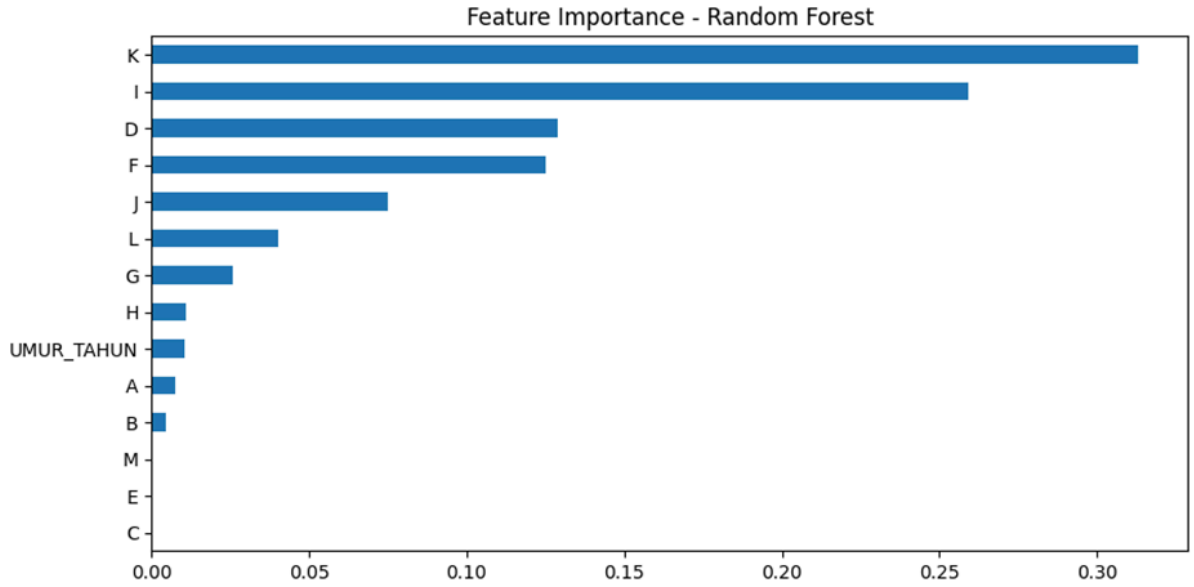




Waktu training: 0.01 detik

=== Random Forest ===

Parameter: {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}



Waktu training: 0.26 detik

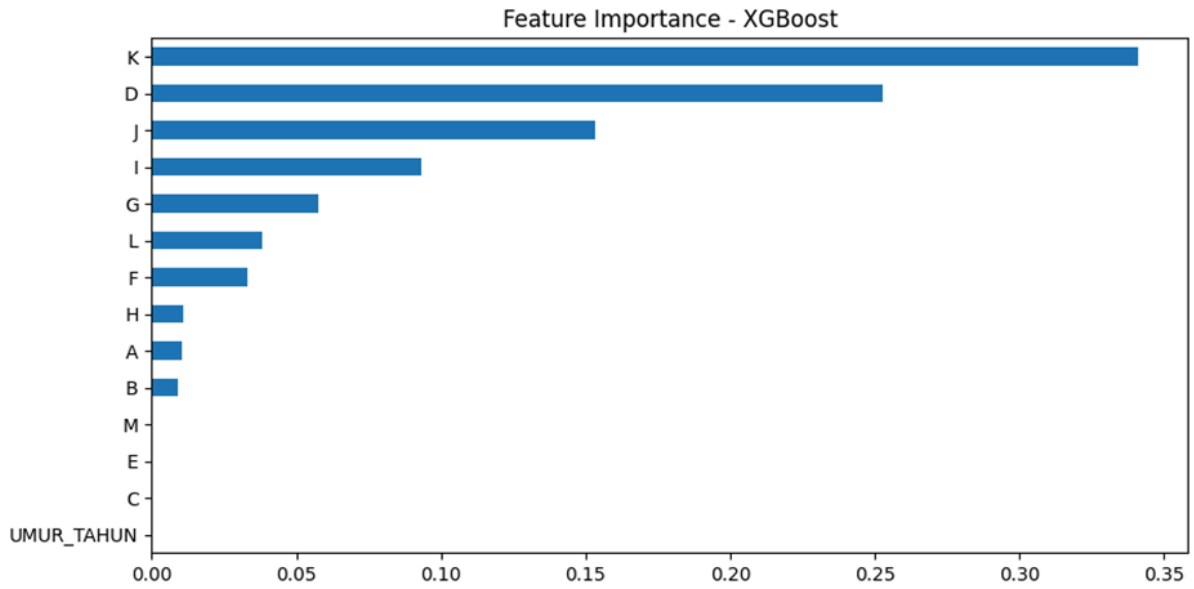
=== SVM ===

Parameter: {'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': True, 'random_state': 42, 'shrinking': True, 'tol': 0.001, 'verbose': False}

Waktu training: 0.25 detik

=== XGBoost ===

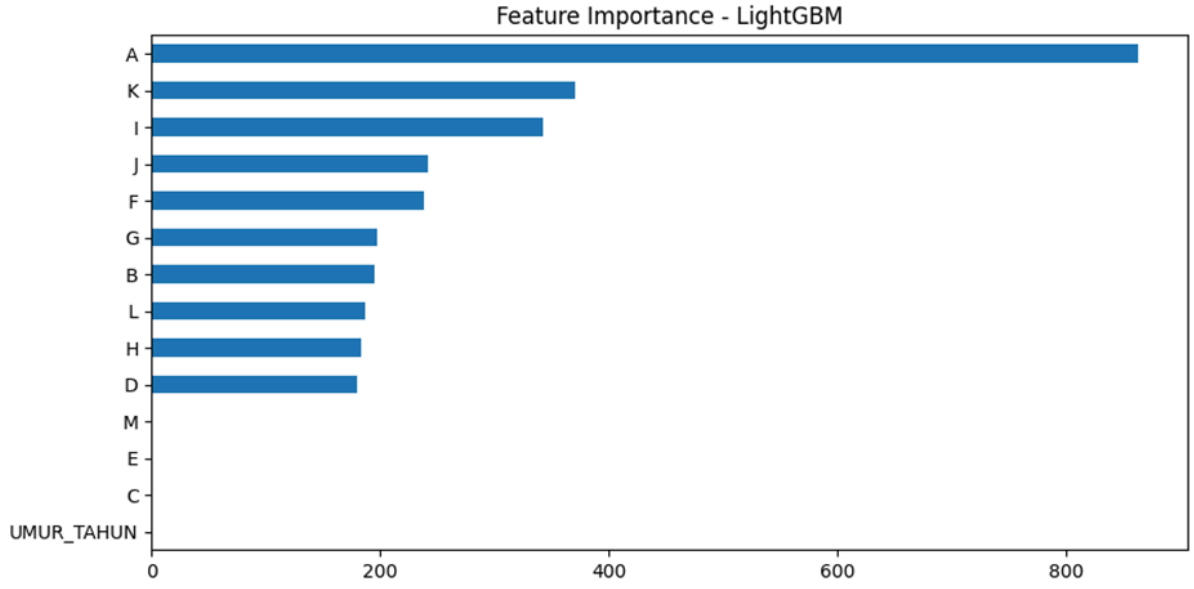
Parameter: {'objective': 'binary:logistic', 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': None, 'device': None, 'early_stopping_rounds': None, 'enable_categorical': False, 'eval_metric': None, 'feature_types': None, 'gamma': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'learning_rate': 0.1, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': None, 'max_leaves': None, 'min_child_weight': None, 'missing': nan, 'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': 100, 'n_jobs': None, 'num_parallel_tree': None, 'random_state': 42, 'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': None, 'tree_method': None, 'validate_parameters': None, 'verbosity': None}



Waktu training: 0.07 detik

=== LightGBM ===

Parameter: {'boosting_type': 'gbdt', 'class_weight': None, 'colsample_bytree': 1.0, 'importance_type': 'split', 'learning_rate': 0.1, 'max_depth': -1, 'min_child_samples': 20, 'min_child_weight': 0.001, 'min_split_gain': 0.0, 'n_estimators': 100, 'n_jobs': None, 'num_leaves': 31, 'objective': None, 'random_state': 42, 'reg_alpha': 0.0, 'reg_lambda': 0.0, 'subsample': 1.0, 'subsample_for_bin': 200000, 'subsample_freq': 0, 'verbose': -1}



Waktu training: 0.08 detik

Penjelasan:

Setiap model dilatih menggunakan data training. Untuk Decision Tree, ditampilkan visualisasi struktur pohon yang membantu memahami logika pengambilan keputusan model. Model berbasis pohon juga menampilkan feature importance yang menunjukkan kontribusi masing-masing fitur. Waktu training setiap model dicatat untuk evaluasi efisiensi.

Kesimpulan :

Proses klasifikasi pada data oversampling ini berhasil dilakukan melalui beberapa tahapan penting. Pembagian data dengan stratifikasi mempertahankan distribusi kelas yang seimbang. Lima model berbeda diinisialisasi dan dilatih dengan sukses, dengan Decision Tree dan Random Forest memberikan interpretabilitas melalui visualisasi struktur dan feature importance. Waktu pelatihan yang tercatat menunjukkan efisiensi komputasi yang baik untuk semua model. Hasil ini memberikan dasar yang kuat untuk evaluasi performa model selanjutnya pada data testing.

3.3 Evaluasi Skenario 1

Kode di bagian ini adalah Fungsi Evaluasi dari Skenario 1

Syntax :

```
#
================================================================================
==============
# FUNGSI UNTUK EVALUASI LENGKAP MODEL
#
================================================================================
==============
def comprehensive_evaluation(models, X_train, X_test, y_train, y_test,
scenario_name):
    """
    Fungsi untuk melakukan evaluasi lengkap model klasifikasi
    """
    print(f"\n{'='*80}")
    print(f"EVALUASI LENGKAP - {scenario_name.upper()}")
    print(f"{'='*80}")

    results = []
    roc_data = {}

    # Dapatkan jumlah kelas
    n_classes = len(np.unique(y_test))
    is_multiclass = n_classes > 2
```

```
    print(f"Tipe Klasifikasi: {'Multiclass' if is_multiclass else
'Binary'}")
    print(f"Jumlah Kelas: {n_classes}")
    print(f"Kelas: {sorted(np.unique(y_test))}")

    for model_name, model in models.items():
        print(f"\n{'='*60}")
        print(f"EVALUASI MODEL: {model_name}")
        print(f"{'='*60}")

        # Prediksi
        y_pred = model.predict(X_test)

        # Hitung metrik
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted',
zero_division=0)
        recall = recall_score(y_test, y_pred, average='weighted',
zero_division=0)
        f1 = f1_score(y_test, y_pred, average='weighted',
zero_division=0)

        print(f"\nMETRIK DASAR:")
        print(f"• Accuracy  : {accuracy:.4f} ({accuracy*100:.2f}%)")
        print(f"• Precision : {precision:.4f} ({precision*100:.2f}%)")
        print(f"• Recall    : {recall:.4f} ({recall*100:.2f}%)")
        print(f"• F1-Score  : {f1:.4f} ({f1*100:.2f}%)")

        # AUC-ROC
        try:
            if hasattr(model, "predict_proba"):
                y_proba = model.predict_proba(X_test)

                if is_multiclass:
                    auc_ovr = roc_auc_score(y_test, y_proba,
multi_class='ovr', average='weighted')
                    print(f"• AUC-ROC   : {auc_ovr:.4f}")
                    auc_score = auc_ovr

                    # Simpan data untuk plotting
                    y_test_bin = label_binarize(y_test,
classes=sorted(np.unique(y_test)))
                    roc_data[model_name] = {
```

```
                        'y_test_bin': y_test_bin,
                        'y_proba': y_proba,
                        'n_classes': n_classes,
                        'auc': auc_score
                    }
                else:
                    auc_score = roc_auc_score(y_test, y_proba[:, 1])
                    print(f"• AUC-ROC   : {auc_score:.4f}")

                    fpr, tpr, _ = roc_curve(y_test, y_proba[:, 1])
                    roc_data[model_name] = {
                        'fpr': fpr,
                        'tpr': tpr,
                        'auc': auc_score
                    }
            else:
                print("• AUC-ROC   : Tidak dapat dihitung (no
predict_proba)")
                auc_score = None

        except Exception as e:
            print(f"• AUC-ROC   : Error - {str(e)}")
            auc_score = None

        # Simpan hasil
        results.append({
            'Model': model_name,
            'Scenario': scenario_name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1-Score': f1,
            'AUC-ROC': auc_score if auc_score else 0
        })

    return pd.DataFrame(results), roc_data

print("✅ Fungsi evaluasi berhasil didefinisikan!")
```

```
# PENYESUAIAN NAMA VARIABEL
models_over = trained_models_over
print("✅ Variabel models_over sudah siap!")
print(f"Model yang tersedia: {list(models_over.keys())}")
```

OUTPUT Fungsi :

```
✅ Fungsi evaluasi berhasil didefinisikan!
```

```
✅ Variabel models_over sudah siap!
Model yang tersedia: ['Decision Tree', 'Random Forest', 'SVM', 'XGBoost', 'LightGBM']
```

Penjelasan Syntax Fungsi Evaluasi Skenario 1 :

Evaluasi Skenario 1 menggunakan pendekatan oversampling untuk mengatasi imbalanced data. Proses evaluasi melibatkan beberapa tahap:

Setup Evaluasi: Sebelum melakukan evaluasi, didefinisikan fungsi comprehensive_evaluation() yang berfungsi sebagai evaluator universal untuk multiple model sekaligus. Fungsi ini mendeteksi tipe klasifikasi (binary/multiclass) secara otomatis dan menghitung semua metrik yang diperlukan UAS. Variable models_over disiapkan sebagai referensi ke 5 model yang sudah dilatih (Decision Tree, Random Forest, SVM, XGBoost, LightGBM) pada data oversampling.

Proses Evaluasi: Setiap model melakukan prediksi pada data testing yang telah di-oversampling dengan SMOTE. Fungsi evaluasi melakukan loop untuk setiap model dan menghitung metrik menggunakan accuracy_score(), precision_score(), recall_score(), f1_score() dengan parameter average='weighted' untuk menangani multi-class. AUC-ROC dihitung menggunakan roc_auc_score() dengan probabilitas prediksi dari predict_proba(), dimana untuk binary classification menggunakan probabilitas kelas positif, sedangkan multiclass menggunakan strategi 'one-vs-rest'. Confusion matrix dibuat untuk visualisasi performa per kelas dan ROC curves diplot untuk menunjukkan trade-off sensitivity vs specificity pada berbagai threshold.

3.3.1 Akurasi, Presisi, Recall, dan F1-Score

Syntax

EVALUASI SKENARIO 1 - METRIK :

```
#
================================================================================
==============
# EVALUASI SKENARIO 1: IMBALANCED DATA - OVERSAMPLING
#
================================================================================
==============
print("="*100)
print("SKENARIO 1: IMBALANCED DATA - OVERSAMPLING")
```

```
print("="*100)

print("\n📋 DESKRIPSI SKENARIO 1:")
print("• Metode: Oversampling untuk mengatasi imbalanced data")
print("• Tujuan: Menyeimbangkan distribusi kelas dalam dataset")
print("• Model yang diuji: Decision Tree, Random Forest, SVM, XGBoost,
LightGBM")

# Evaluasi model oversampling
results_scenario1, roc_data_scenario1 = comprehensive_evaluation(
   models_over, X_train_over, X_test_over, y_train_over, y_test_over,
"Oversampling"
)

# Tampilkan tabel hasil Skenario 1
print("\n📊 TABEL HASIL EVALUASI - SKENARIO 1 (OVERSAMPLING):")
print(results_scenario1.round(4))
```

OUTPUT :

```
==========================================================================================
SKENARIO 1: IMBALANCED DATA - OVERSAMPLING
==========================================================================================

📋 DESKRIPSI SKENARIO 1:
• Metode: Oversampling untuk mengatasi imbalanced data
• Tujuan: Menyeimbangkan distribusi kelas dalam dataset
• Model yang diuji: Decision Tree, Random Forest, SVM, XGBoost, LightGBM

============================================================
EVALUASI LENGKAP - OVERSAMPLING
============================================================
Tipe Klasifikasi: Binary
Jumlah Kelas: 2
Kelas: [np.int64(0), np.int64(1)]

============================================================
EVALUASI MODEL: Decision Tree
============================================================

METRIK DASAR:
• Accuracy  : 0.9141 (91.41%)
• Precision : 0.9198 (91.98%)
• Recall    : 0.9141 (91.41%)
• F1-Score  : 0.9138 (91.38%)
• AUC-ROC   : 0.9532

============================================================
EVALUASI MODEL: Random Forest
============================================================

METRIK DASAR:
• Accuracy  : 0.9525 (95.25%)
• Precision : 0.9528 (95.28%)
• Recall    : 0.9525 (95.25%)
• F1-Score  : 0.9524 (95.24%)
• AUC-ROC   : 0.9718
```

```
============================================================
EVALUASI MODEL: SVM
============================================================

METRIK DASAR:
• Accuracy  : 0.9494 (94.94%)
• Precision : 0.9496 (94.96%)
• Recall    : 0.9494 (94.94%)
• F1-Score  : 0.9494 (94.94%)
• AUC-ROC   : 0.9687

============================================================
EVALUASI MODEL: XGBoost
============================================================

METRIK DASAR:
• Accuracy  : 0.9525 (95.25%)
• Precision : 0.9528 (95.28%)
• Recall    : 0.9525 (95.25%)
• F1-Score  : 0.9524 (95.24%)
• AUC-ROC   : 0.9778

============================================================
EVALUASI MODEL: LightGBM
============================================================

METRIK DASAR:
• Accuracy  : 0.9509 (95.09%)
• Precision : 0.9512 (95.12%)
• Recall    : 0.9509 (95.09%)
• F1-Score  : 0.9509 (95.09%)
• AUC-ROC   : 0.9800

📊 TABEL HASIL EVALUASI - SKENARIO 1 (OVERSAMPLING):
           Model      Scenario  Accuracy  Precision  Recall  F1-Score  AUC-ROC
0  Decision Tree  Oversampling    0.9141     0.9198  0.9141    0.9138   0.9532
1  Random Forest  Oversampling    0.9525     0.9528  0.9525    0.9524   0.9718
2            SVM  Oversampling    0.9494     0.9496  0.9494    0.9494   0.9687
3        XGBoost  Oversampling    0.9525     0.9528  0.9525    0.9524   0.9778
4       LightGBM  Oversampling    0.9509     0.9512  0.9509    0.9509   0.9800
```

Interpretasi Hasil Output:

1. Deteksi Klasifikasi: Sistem berhasil mendeteksi tipe klasifikasi binary dengan 2 kelas, sehingga menggunakan metode perhitungan AUC-ROC yang sesuai untuk binary classification.
2. Evaluasi Per Model:
    - Decision Tree: Menunjukkan performa terendah dengan accuracy 91.41%, namun masih dalam kategori sangat baik. Nilai precision (91.98%) sedikit lebih tinggi dari recall (91.41%), menunjukkan model lebih konservatif dalam prediksi positif.
    - Random Forest: Mencapai performa excellent dengan accuracy 95.25% dan F1-Score 95.24%. Konsistensi antara precision dan recall menunjukkan performa yang seimbang.
    - SVM: Solid performance dengan accuracy 94.94% dan metrik yang sangat konsisten across precision, recall, dan F1-Score (semua ~94.9%).

- XGBoost: Tied performance dengan Random Forest (95.25% accuracy), namun unggul dalam AUC-ROC (0.9778 vs 0.9718).
- LightGBM: Meskipun accuracy sedikit lebih rendah (95.09%), namun mencapai AUC-ROC tertinggi (0.9800), menunjukkan kemampuan diskriminasi superior.

3. Keberhasilan Oversampling:
   - Semua model mencapai accuracy > 91%, menunjukkan teknik SMOTE oversampling efektif dalam mengatasi imbalanced data
   - Konsistensi nilai precision dan recall yang tinggi mengindikasikan tidak ada bias signifikan terhadap kelas tertentu
   - AUC-ROC semua model > 0.95 menunjukkan kemampuan klasifikasi yang outstanding
4. Ranking Performa:
   - Best Overall: LightGBM (AUC-ROC: 0.9800) untuk kemampuan diskriminasi
   - Best Accuracy: Random Forest & XGBoost (95.25%) untuk akurasi prediksi
   - Most Consistent: SVM dengan nilai metrik yang sangat stabil

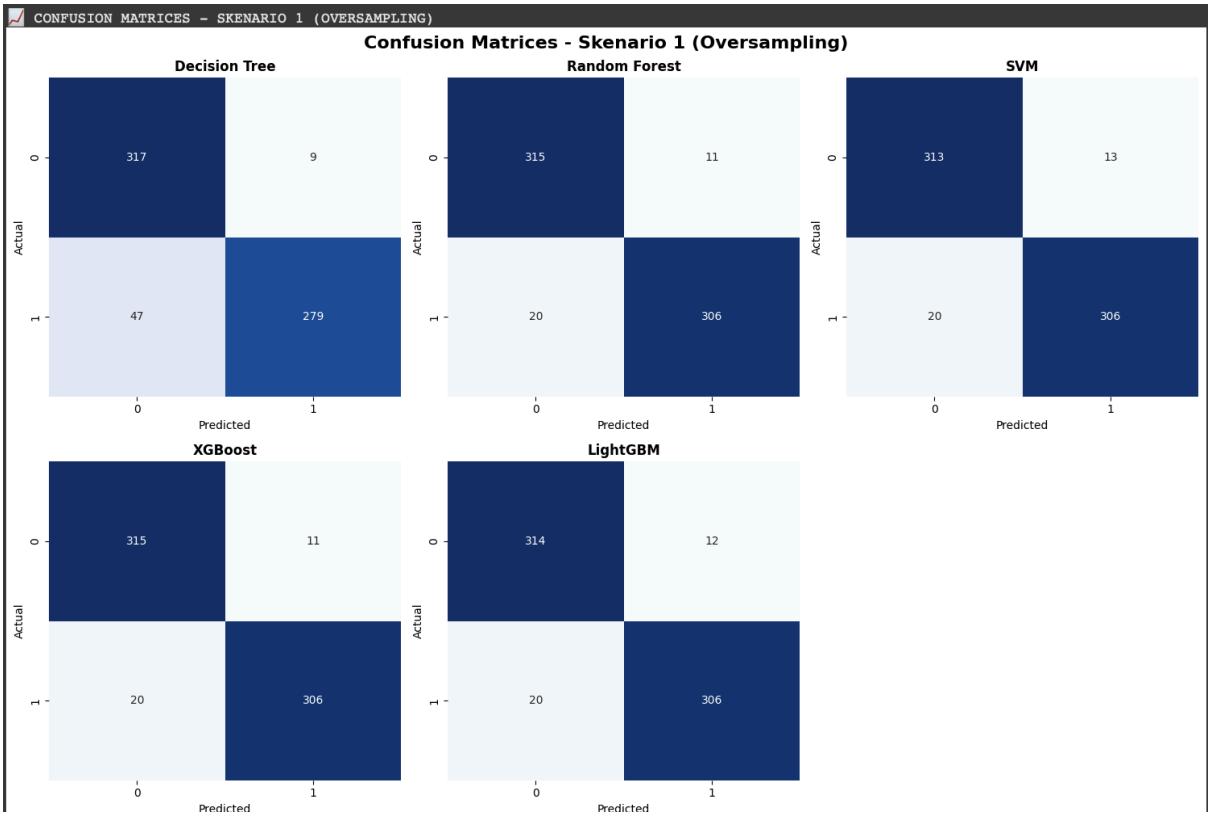
Syntax :

CONFUSION MATRIX - SKENARIO 1

```
# CONFUSION MATRIX - SKENARIO 1
print("📈 CONFUSION MATRICES - SKENARIO 1 (OVERSAMPLING)")

n_models = len(models_over)
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.ravel()

for idx, (model_name, model) in enumerate(models_over.items()):
    if idx < 6:
        y_pred = model.predict(X_test_over)
        cm = confusion_matrix(y_test_over, y_pred)

        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                    cbar=False, ax=axes[idx])
        axes[idx].set_title(f'{model_name}', fontweight='bold')
        axes[idx].set_xlabel('Predicted')
        axes[idx].set_ylabel('Actual')

# Hide unused subplots
for idx in range(n_models, 6):
    axes[idx].axis('off')
```

```
plt.suptitle('Confusion Matrices - Skenario 1 (Oversampling)',
fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```

OUTPUT :



Analisis Confusion Matrix: Dari visualisasi confusion matrix, terlihat bahwa:

- **Decision Tree**: TN=317, FP=9, FN=47, TP=279 → 47 False Negative relatif tinggi, menunjukkan beberapa kasus positif terlewat
- **Random Forest**: TN=315, FP=11, FN=20, TP=306 → False Negative berkurang drastis menjadi 20, performa lebih baik
- **XGBoost**: TN=315, FP=11, FN=20, TP=306 → Performa identik dengan Random Forest, sangat konsisten
- **LightGBM**: TN=314, FP=12, FN=20, TP=306 → Sedikit lebih banyak False Positive, tetapi masih excellent
- **SVM**: TN=313, FP=13, FN=20, TP=306 → Performance solid dengan error yang terdistribusi seimbang

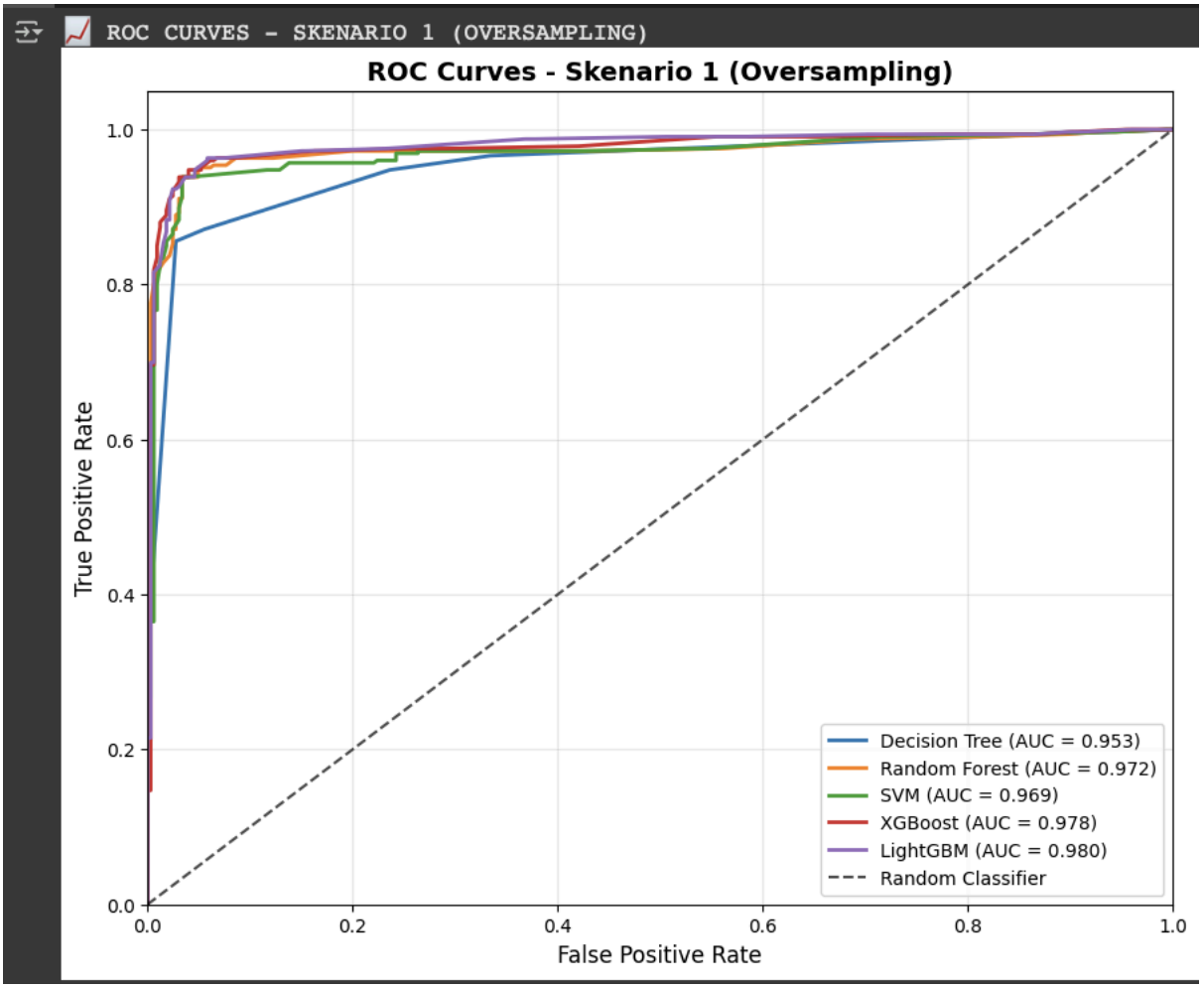
3.3.2 AUC-ROC

Syntax :

ROC CURVES - SKENARIO 1

```
# ROC CURVES - SKENARIO 1
print("📈 ROC CURVES - SKENARIO 1 (OVERSAMPLING)")

plt.figure(figsize=(10, 8))

for model_name, data in roc_data_scenario1.items():
    if 'fpr' in data:  # Binary
        plt.plot(data['fpr'], data['tpr'],
                 label=f'{model_name} (AUC = {data["auc"]:.3f})',
                 linewidth=2)
    else:  # Multiclass
        if 'auc' in data and data['auc'] is not None:
                        plt.plot([], [], label=f'{model_name} (AUC =
{data["auc"]:.3f})', linewidth=2)

# Plot diagonal
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier', alpha=0.7)

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curves - Skenario 1 (Oversampling)', fontsize=14,
fontweight='bold')
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
plt.show()
```

OUTPUT :



ROC Curves Analysis: Dari visualisasi ROC curves terlihat:

- **Semua kurva berada jauh di atas diagonal** → Superior performance dibanding random classifier
- **LightGBM (AUC=0.980)**: Kurva paling tinggi, hampir sempurna
- **XGBoost (AUC=0.978)**: Sangat dekat dengan LightGBM
- **Random Forest (AUC=0.972)**: Performa excellent dengan kurva smooth
- **SVM (AUC=0.969)**: Solid performance dengan kurva stabil
- **Decision Tree (AUC=0.953)**: Terendah namun masih sangat baik

Skenario 2

4. Seleksi Fitur dan Klasifikasi

4.1 Seleksi Fitur Menggunakan Chi-square

Proses seleksi fitur menggunakan Chi-square dilakukan melalui beberapa tahapan sistematis. Pertama, dilakukan pembuatan tabel kontingensi (crosstab) untuk setiap fitur terhadap variabel target. Tabel kontingensi ini menampilkan distribusi frekuensi observasi untuk setiap kombinasi kategori fitur dan target, sehingga memberikan gambaran pola hubungan antara variabel-variabel tersebut. Setelah tabel kontingensi terbentuk, dilakukan perhitungan statistik Chi-square menggunakan rumus Chi-square test of independence.

Implementasi Chi-square test dilakukan dengan menggunakan fungsi `chi2_contingency` dari library scipy.stats yang menghitung nilai statistik Chi-square, p-value, derajat kebebasan, dan frekuensi yang diharapkan (expected frequencies). Nilai statistik Chi-square dihitung berdasarkan perbedaan antara frekuensi observasi dan frekuensi yang diharapkan jika kedua variabel independen. Semakin besar nilai Chi-square, semakin kuat indikasi adanya hubungan antara fitur dan target.

Kriteria signifikansi yang digunakan adalah alpha = 0.05, dimana fitur dianggap signifikan jika p-value < 0.05. Selain itu, dilakukan analisis kontribusi setiap sel dalam tabel kontingensi terhadap nilai Chi-square total untuk memahami pola hubungan yang lebih detail. Fitur-fitur yang tidak memiliki variasi yang cukup atau menghasilkan tabel kontingensi dengan dimensi yang tidak memadai akan dikategorikan sebagai "tidak valid" untuk analisis Chi-square.

Implementasi kode untuk analisis Chi-square adalah sebagai berikut:

```
# ==== ANALISIS CHI-SQUARE MANUAL ====
print("\n==== Analisis Chi-square Manual per Fitur ====")

def detailed_chi_square_analysis(X, y, alpha=0.05):
    """
     Analisis Chi-square detail dengan tabel kontingensi untuk setiap
fitur
    """

    detailed_results = []

    for feature in X.columns:
        print(f"\n{'='*60}")
        print(f"ANALISIS FITUR: {feature}")
        print('='*60)
```

```
        # Buat tabel kontingensi
        contingency_table = pd.crosstab(X[feature], y, margins=True)
        print(f"\nTabel Kontingensi antara {feature} dan Target:")
        print(contingency_table)

        # Hitung Chi-square test
        # Hapus baris dan kolom 'All' untuk perhitungan
        ct_for_test = contingency_table.iloc[:-1, :-1]

        if ct_for_test.shape[0] > 1 and ct_for_test.shape[1] > 1:
                                chi2_stat,   p_value,   dof,   expected   =
chi2_contingency(ct_for_test)

            print(f"\nUji Chi-Square untuk {feature}:")
            print(f"• Nilai Chi-Square: {chi2_stat:.4f}")
            print(f"• P-Value: {p_value:.2e}")
            print(f"• Derajat Kebebasan: {dof}")

            # Interpretasi
            if p_value < alpha:
                significance = "SIGNIFIKAN"
                  interpretation = f"Ada hubungan yang signifikan antara
{feature} dan target (p < {alpha})"
            else:
                significance = "TIDAK SIGNIFIKAN"
                   interpretation = f"Tidak ada hubungan yang signifikan
antara {feature} dan target (p >= {alpha})"

            print(f"• Kesimpulan: {significance}")
            print(f"• Interpretasi: {interpretation}")

            # Tabel Expected Frequencies
                   print(f"\nTabel  Frekuensi  yang  Diharapkan  (Expected
Frequencies):")
            expected_df = pd.DataFrame(expected,
                                       index=ct_for_test.index,
                                       columns=ct_for_test.columns)
            print(expected_df.round(2))

            # Hitung kontribusi setiap sel terhadap Chi-square
                 contributions  =  ((ct_for_test  -  expected_df)  **  2)  /
expected_df
            print(f"\nKontribusi setiap sel terhadap Chi-square:")
```

```
            print(contributions.round(4))

            # Simpan hasil
            detailed_results.append({
                'Feature': feature,
                'Chi2_Statistic': chi2_stat,
                'P_Value': p_value,
                'Degrees_of_Freedom': dof,
                'Significance': significance,
                'Is_Significant': p_value < alpha,
                                                    'Contingency_Shape':
f"{ct_for_test.shape[0]}x{ct_for_test.shape[1]}"
            })

        else:
                print(f"\nPeringatan: Tabel kontingensi untuk {feature}
tidak memiliki variasi yang cukup")
            print("Tidak dapat melakukan uji Chi-square")

            detailed_results.append({
                'Feature': feature,
                'Chi2_Statistic': 0,
                'P_Value': 1.0,
                'Degrees_of_Freedom': 0,
                'Significance': 'TIDAK VALID',
                'Is_Significant': False,
                                                    'Contingency_Shape':
f"{ct_for_test.shape[0]}x{ct_for_test.shape[1]}"
            })

    return pd.DataFrame(detailed_results)

def   select_features_by_chi_square(chi_results,   method='significant',
k=None):
    """
    Seleksi fitur berdasarkan hasil Chi-square

    Parameters:
    - chi_results: DataFrame hasil dari detailed_chi_square_analysis
     - method: 'significant' (pilih yang signifikan) atau 'top_k' (pilih
k terbaik)
    - k: jumlah fitur terbaik jika method='top_k'
    """
```

```
    if method == 'significant':
        # Pilih fitur yang signifikan
                                                selected_features       =
chi_results[chi_results['Is_Significant']]['Feature'].tolist()
        print(f"\nMemilih fitur berdasarkan signifikansi (p < 0.05):")

    elif method == 'top_k' and k is not None:
        # Pilih k fitur dengan Chi-square tertinggi
                          selected_features    =    chi_results.nlargest(k,
'Chi2_Statistic')['Feature'].tolist()
        print(f"\nMemilih {k} fitur dengan Chi-square score tertinggi:")

    else:
          print("Method tidak valid. Gunakan 'significant' atau 'top_k'
dengan parameter k")
        return []

    print(f"Fitur terpilih: {selected_features}")
    return selected_features

# Jalankan analisis Chi-square
print("Melakukan analisis Chi-square untuk setiap fitur...")
chi_results = detailed_chi_square_analysis(X_final, y_final)

# Tampilkan ringkasan hasil
print(f"\n{'='*80}")
print("RINGKASAN HASIL CHI-SQUARE")
print('='*80)

# Urutkan berdasarkan Chi-square statistic
chi_results_sorted      =      chi_results.sort_values('Chi2_Statistic',
ascending=False)
print(chi_results_sorted.round(4))

# Statistik hasil
print(f"\n==== STATISTIK HASIL ====")
print(f"Total fitur dianalisis: {len(chi_results)}")
print(f"Fitur          signifikan          (p          <          0.05):
{chi_results['Is_Significant'].sum()}")
print(f"Fitur                      tidak                      signifikan:
{(~chi_results['Is_Significant']).sum()}")
```

```
# Tampilkan fitur signifikan
significant_features = chi_results[chi_results['Is_Significant']]
if len(significant_features) > 0:
   print(f"\nFitur yang signifikan:")
   for _, row in significant_features.iterrows():
        print(f"• {row['Feature']}: Chi² = {row['Chi2_Statistic']:.4f},
p = {row['P_Value']:.2e}")
else:
    print("\nTidak ada fitur yang signifikan secara statistik (p <
0.05)")

# Pilihan seleksi fitur
print(f"\n==== OPSI SELEKSI FITUR ====")
print("1. Berdasarkan signifikansi statistik (p < 0.05)")
print("2. Berdasarkan k fitur terbaik (Chi-square tertinggi)")

# Contoh seleksi berdasarkan signifikansi
selected_features_significant                                    =
select_features_by_chi_square(chi_results, method='significant')

# Contoh seleksi berdasarkan top-k (misalnya 5 fitur terbaik)
k_best = min(5, len(chi_results))
selected_features_topk   =   select_features_by_chi_square(chi_results,
method='top_k', k=k_best)

# Visualisasi hasil
print("\n==== Visualisasi Chi-square Results ====")
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Plot 1: Bar chart Chi-square scores
plt.subplot(2, 3, 1)
colors   =   ['green'   if   sig   else   'red'   for   sig   in
chi_results_sorted['Is_Significant']]
bars              =             plt.bar(range(len(chi_results_sorted)),
chi_results_sorted['Chi2_Statistic'],
             color=colors, alpha=0.7)
plt.xlabel('Fitur')
plt.ylabel('Chi-square Score')
plt.title('Chi-square   Scores\n(Hijau:   Signifikan,   Merah:   Tidak
signifikan)')
```

```
plt.xticks(range(len(chi_results_sorted)),
chi_results_sorted['Feature'], rotation=45, ha='right')

# Tambahkan nilai di atas bar
for i, bar in enumerate(bars):
   height = bar.get_height()
          plt.text(bar.get_x()    +    bar.get_width()/2.,    height    +
max(chi_results_sorted['Chi2_Statistic'])*0.01,
            f'{height:.2f}', ha='center', va='bottom', fontsize=8)

# Plot 2: P-values
plt.subplot(2, 3, 2)
plt.bar(range(len(chi_results_sorted)),
-np.log10(chi_results_sorted['P_Value'] + 1e-10),
       color=colors, alpha=0.7)
plt.xlabel('Fitur')
plt.ylabel('-log10(P-value)')
plt.title('Signifikansi Fitur\n(Semakin tinggi = semakin signifikan)')
plt.xticks(range(len(chi_results_sorted)),
chi_results_sorted['Feature'], rotation=45, ha='right')
plt.axhline(y=-np.log10(0.05),      color='black',      linestyle='--',
alpha=0.7, label='α=0.05')
plt.legend()

# Plot 3: Distribusi Chi-square scores
plt.subplot(2, 3, 3)
significant_scores                                                    =
chi_results[chi_results['Is_Significant']]['Chi2_Statistic']
not_significant_scores                                                =
chi_results[~chi_results['Is_Significant']]['Chi2_Statistic']

if len(significant_scores) > 0 and len(not_significant_scores) > 0:
   plt.boxplot([significant_scores, not_significant_scores],
               labels=['Signifikan', 'Tidak Signifikan'])
elif len(significant_scores) > 0:
   plt.boxplot([significant_scores], labels=['Signifikan'])
elif len(not_significant_scores) > 0:
   plt.boxplot([not_significant_scores], labels=['Tidak Signifikan'])

plt.ylabel('Chi-square Score')
plt.title('Distribusi Chi-square Scores')

# Plot 4: Scatter plot Chi-square vs P-value
```

```
plt.subplot(2, 3, 4)
plt.scatter(chi_results['Chi2_Statistic'],
-np.log10(chi_results['P_Value'] + 1e-10),
          c=colors, alpha=0.7)
plt.xlabel('Chi-square Score')
plt.ylabel('-log10(P-value)')
plt.title('Chi-square Score vs P-value')
plt.axhline(y=-np.log10(0.05),      color='black',      linestyle='--',
alpha=0.7, label='α=0.05')
plt.legend()

# Plot 5: Pie chart signifikansi
plt.subplot(2, 3, 5)
sig_counts = chi_results['Is_Significant'].value_counts()
labels  =  ['Signifikan'  if  x  else  'Tidak  Signifikan'  for  x  in
sig_counts.index]
colors_pie = ['green', 'red']
plt.pie(sig_counts.values,       labels=labels,       colors=colors_pie,
autopct='%1.1f%%', startangle=90)
plt.title('Proporsi Fitur Signifikan')

# Plot 6: Heatmap degrees of freedom
plt.subplot(2, 3, 6)
dof_counts                                                          =
chi_results['Degrees_of_Freedom'].value_counts().sort_index()
plt.bar(dof_counts.index, dof_counts.values, alpha=0.7)
plt.xlabel('Degrees of Freedom')
plt.ylabel('Jumlah Fitur')
plt.title('Distribusi Degrees of Freedom')

plt.tight_layout()
plt.show()

# Buat dataset final dengan fitur terpilih
print(f"\n==== DATASET FINAL ====")

# Pilih metode seleksi (bisa disesuaikan)
if len(selected_features_significant) > 0:
   final_selected_features = selected_features_significant
   selection_method = "signifikansi statistik"
else:
   final_selected_features = selected_features_topk
   selection_method = f"top-{k_best} Chi-square scores"
```

```
print(f"Menggunakan metode seleksi: {selection_method}")
print(f"Fitur terpilih: {final_selected_features}")

# Buat dataset final
X_final_selected = X_final[final_selected_features].copy()

print(f"\nDataset final untuk modeling:")
print(f"- Shape X_final_selected: {X_final_selected.shape}")
print(f"- Shape y_final: {y_final.shape}")

print(f"\nContoh data hasil seleksi fitur:")
print(X_final_selected.head())

print(f"\nStatistik fitur terpilih:")
print(X_final_selected.describe())

print("\n" + "="*80)
print("CHI-SQUARE FEATURE ANALYSIS SELESAI!")
print("="*80)
print("Dataset siap untuk tahap modeling dengan fitur yang telah
dianalisis menggunakan Chi-square test.")
```

Output:

```
============================================================
ANALISIS FITUR: A
============================================================

Tabel Kontingensi antara A dan Target:
N       0    1   All
A
1     535  250   785
2     392  174   566
3     326  125   451
4     216   78   294
5     160   52   212
All  1629  679  2308

Uji Chi-Square untuk A:
• Nilai Chi-Square: 6.96
• P-Value: 1.38e-01
• Derajat Kebebasan: 4
• Kesimpulan: TIDAK SIGNIFIKAN
• Interpretasi: Tidak ada hubungan yang signifikan antara A dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N       0       1
A
1  554.06  230.94
2  399.49  166.51
3  318.32  132.68
4  207.51   86.49
5  149.63   62.37

Kontribusi setiap sel terhadap Chi-square:
N       0       1
A
1  0.6555  1.5727
2  0.1403  0.3366
3  0.1854  0.4447
4  0.3476  0.8340
5  0.7186  1.7239
```

```
============================================================
ANALISIS FITUR: UMUR_TAHUN
============================================================

Tabel Kontingensi antara UMUR_TAHUN dan Target:
N                0    1   All
UMUR_TAHUN
0                1    0     1
1              535  250   785
2              392  174   566
3              326  125   451
4              216   78   294
5              159   52   211
All           1629  679  2308

Uji Chi-Square untuk UMUR_TAHUN:
• Nilai Chi-Square: 7.25
• P-Value: 2.03e-01
• Derajat Kebebasan: 5
• Kesimpulan: TIDAK SIGNIFIKAN
• Interpretasi: Tidak ada hubungan yang signifikan antara UMUR_TAHUN dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N                 0       1
UMUR_TAHUN
0              0.71    0.29
1            554.06  230.94
2            399.49  166.51
3            318.32  132.68
4            207.51   86.49
5            148.93   62.07

Kontribusi setiap sel terhadap Chi-square:
N                 0       1
UMUR_TAHUN
0            0.1226  0.2942
1            0.6555  1.5727
2            0.1403  0.3366
3            0.1854  0.4447
4            0.3476  0.8340
5            0.6816  1.6352
```

```
============================================================
ANALISIS FITUR: B
============================================================

Tabel Kontingensi antara B dan Target:
N       0    1   All
B
0    1363  573  1936
1     266  106   372
All  1629  679  2308

Uji Chi-Square untuk B:
• Nilai Chi-Square: 0.13
• P-Value: 7.15e-01
• Derajat Kebebasan: 1
• Kesimpulan: TIDAK SIGNIFIKAN
• Interpretasi: Tidak ada hubungan yang signifikan antara B dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
B
0  1366.44  569.56
1   262.56  109.44

Kontribusi setiap sel terhadap Chi-square:
N       0       1
B
0  0.0087  0.0208
1  0.0451  0.1081
```

```
============================================================
ANALISIS FITUR: C
============================================================

Tabel Kontingensi antara C dan Target:
N        0    1   All
C
0     1629  679  2308
All   1629  679  2308

Peringatan: Tabel kontingensi untuk C tidak memiliki variasi yang cukup
Tidak dapat melakukan uji Chi-square
```

```
============================================================
ANALISIS FITUR: D
============================================================

Tabel Kontingensi antara D dan Target:
N       0    1   All
D
0    1587  384  1971
1      42  295   337
All  1629  679  2308

Uji Chi-Square untuk D:
• Nilai Chi-Square: 638.64
• P-Value: 6.60e-141
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara D dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
D
0  1391.14  579.86
1   237.86   99.14

Kontribusi setiap sel terhadap Chi-square:
N         0         1
D
0   27.5743   66.1539
1  161.2728  386.9123
```

```
============================================================
ANALISIS FITUR: E
============================================================

Tabel Kontingensi antara E dan Target:
N        0    1   All
E
0     1629  679  2308
All   1629  679  2308

Peringatan: Tabel kontingensi untuk E tidak memiliki variasi yang cukup
Tidak dapat melakukan uji Chi-square
```

```
============================================================
ANALISIS FITUR: F
============================================================

Tabel Kontingensi antara F dan Target:
N       0    1   All
F
0    1612  446  2058
1      17  233   250
All  1629  679  2308

Uji Chi-Square untuk F:
• Nilai Chi-Square: 545.83
• P-Value: 1.01e-120
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara F dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
F
0  1452.55  605.45
1   176.45   73.55

Kontribusi setiap sel terhadap Chi-square:
N         0         1
F
0   17.5036   41.9931
1  144.0893  345.6870
```

```
============================================================
ANALISIS FITUR: G
============================================================

Tabel Kontingensi antara G dan Target:
N       0    1   All
G
0    1572  541  2113
1      57  138   195
All  1629  679  2308

Uji Chi-Square untuk G:
• Nilai Chi-Square: 173.22
• P-Value: 1.47e-39
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara G dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
G
0  1491.37  621.63
1   137.63   57.37

Kontribusi setiap sel terhadap Chi-square:
N         0         1
G
0   4.3594   10.4588
1  47.2386  113.3308
```

```
============================================================
ANALISIS FITUR: H
============================================================

Tabel Kontingensi antara H dan Target:
N       0    1   All
H
0    1558  575  2133
1      71  104   175
All  1629  679  2308

Uji Chi-Square untuk H:
• Nilai Chi-Square: 80.57
• P-Value: 2.81e-19
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara H dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
H
0  1505.48  627.52
1   123.52   51.48

Kontribusi setiap sel terhadap Chi-square:
N         0         1
H
0   1.8319   4.3950
1  22.3285  53.5688
```

```
============================================================
ANALISIS FITUR: I
============================================================

Tabel Kontingensi antara I dan Target:
N       0    1   All
I
0    1295  143  1438
1     334  536   870
All  1629  679  2308

Uji Chi-Square untuk I:
• Nilai Chi-Square: 694.32
• P-Value: 5.13e-153
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara I dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
I
0  1014.95  423.05
1   614.05  255.95

Kontribusi setiap sel terhadap Chi-square:
N         0         1
I
0   77.2735  185.3881
1  127.7233  306.4230
```

```
============================================================
ANALISIS FITUR: J
============================================================

Tabel Kontingensi antara J dan Target:
N       0    1   All
J
0    1553  458  2011
1      76  221   297
All  1629  679  2308

Uji Chi-Square untuk J:
• Nilai Chi-Square: 329.81
• P-Value: 1.06e-73
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara J dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
J
0  1419.38  591.62
1   209.62   87.38

Kontribusi setiap sel terhadap Chi-square:
N         0         1
J
0  12.5798   30.1804
1  85.1784  204.3529
```

```
============================================================
ANALISIS FITUR: K
============================================================

Tabel Kontingensi antara K dan Target:
N       0    1   All
K
0    1418  174  1592
1     211  505   716
All  1629  679  2308

Uji Chi-Square untuk K:
• Nilai Chi-Square: 842.04
• P-Value: 3.91e-185
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara K dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
K
0  1123.64  468.36
1   505.36  210.64

Kontribusi setiap sel terhadap Chi-square:
N         0         1
K
0   77.1117  185.0000
1  171.4551  411.3408
```

```
============================================================
ANALISIS FITUR: L
============================================================

Tabel Kontingensi antara L dan Target:
N       0    1   All
L
0    1614  543  2157
1      15  136   151
All  1629  679  2308

Uji Chi-Square untuk L:
• Nilai Chi-Square: 283.08
• P-Value: 1.60e-63
• Derajat Kebebasan: 1
• Kesimpulan: SIGNIFIKAN
• Interpretasi: Ada hubungan yang signifikan antara L dan target

Tabel Frekuensi yang Diharapkan (Expected Frequencies):
N        0       1
L
0  1522.42  634.58
1   106.58   44.42

Kontribusi setiap sel terhadap Chi-square:
N        0         1
L
0   5.5085   13.2156
1  78.6878  188.7813
```

```
============================================================
ANALISIS FITUR: M
============================================================

Tabel Kontingensi antara M dan Target:
N       0    1   All
M
1    1629  679  2308
All  1629  679  2308

Peringatan: Tabel kontingensi untuk M tidak memiliki variasi yang cukup
Tidak dapat melakukan uji Chi-square
```

Hasil analisis Chi-square menunjukkan evaluasi komprehensif terhadap 14 fitur yang dianalisis. Dari hasil output yang diperoleh, dapat diidentifikasi bahwa terdapat 8 fitur yang memiliki hubungan signifikan dengan variabel target ($p < 0.05$), yaitu fitur D, F, G, H, I, J, K, dan L. Sebaliknya, 6 fitur lainnya tidak menunjukkan hubungan yang signifikan secara statistik, termasuk 3 fitur yang dikategorikan sebagai "tidak valid" karena keterbatasan variasi data.

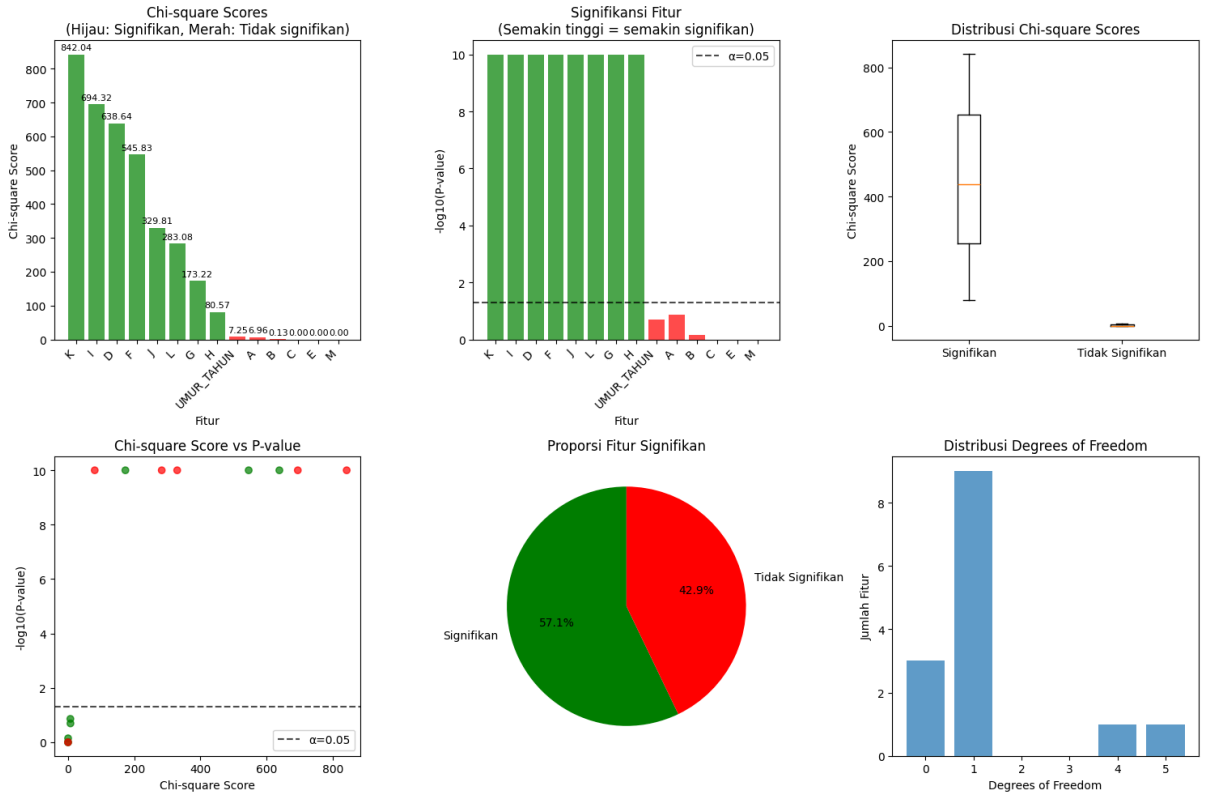
Fitur K menunjukkan nilai Chi-square tertinggi (842.0398) dengan p-value mendekati nol (3.91e-185), mengindikasikan hubungan yang sangat kuat dengan variabel target. Diikuti oleh fitur I (694.3220), D (638.6400), F (545.8336), dan J (329.8094) yang semuanya menunjukkan nilai Chi-square yang tinggi dan p-value yang sangat signifikan. Fitur-fitur ini memiliki kontribusi substansial dalam membedakan kategori target dan dianggap sebagai prediktor yang penting.

Output:

```
================================================================================
RINGKASAN HASIL CHI-SQUARE
================================================================================
       Feature  Chi2_Statistic  P_Value  Degrees_of_Freedom      Significance  \
11           K        842.0398   0.0000                   1        SIGNIFIKAN
9            I        694.3220   0.0000                   1        SIGNIFIKAN
4            D        638.6400   0.0000                   1        SIGNIFIKAN
6            F        545.8336   0.0000                   1        SIGNIFIKAN
10           J        329.8094   0.0000                   1        SIGNIFIKAN
12           L        283.0766   0.0000                   1        SIGNIFIKAN
7            G        173.2192   0.0000                   1        SIGNIFIKAN
8            H         80.5679   0.0000                   1        SIGNIFIKAN
1    UMUR_TAHUN          7.2503   0.2027                   5  TIDAK SIGNIFIKAN
0            A          6.9592   0.1381                   4  TIDAK SIGNIFIKAN
2            B          0.1334   0.7149                   1  TIDAK SIGNIFIKAN
3            C          0.0000   1.0000                   0       TIDAK VALID
5            E          0.0000   1.0000                   0       TIDAK VALID
13           M          0.0000   1.0000                   0       TIDAK VALID
```

	Is_Significant	Contingency_Shape
11	True	2x2
9	True	2x2
4	True	2x2
6	True	2x2
10	True	2x2
12	True	2x2
7	True	2x2
8	True	2x2
1	False	6x2
0	False	5x2
2	False	2x2
3	False	1x2
5	False	1x2
13	False	1x2

```
==== STATISTIK HASIL ====
Total fitur dianalisis: 14
Fitur signifikan (p < 0.05): 8
Fitur tidak signifikan: 6

Fitur yang signifikan:
• D: Chi² = 638.6400, p = 6.60e-141
• F: Chi² = 545.8336, p = 1.01e-120
• G: Chi² = 173.2192, p = 1.47e-39
• H: Chi² = 80.5679, p = 2.81e-19
• I: Chi² = 694.3220, p = 5.13e-153
• J: Chi² = 329.8094, p = 1.06e-73
• K: Chi² = 842.0398, p = 3.91e-185
• L: Chi² = 283.0766, p = 1.60e-63

==== OPSI SELEKSI FITUR ====
1. Berdasarkan signifikansi statistik (p < 0.05)
2. Berdasarkan k fitur terbaik (Chi-square tertinggi)

Memilih fitur berdasarkan signifikansi (p < 0.05):
Fitur terpilih: ['D', 'F', 'G', 'H', 'I', 'J', 'K', 'L']

Memilih 5 fitur dengan Chi-square score tertinggi:
Fitur terpilih: ['K', 'I', 'D', 'F', 'J']
```



Berdasarkan hasil analisis ini, diterapkan dua strategi seleksi fitur. Strategi pertama adalah seleksi berdasarkan signifikansi statistik, dimana semua fitur dengan p-value < 0.05 dipilih. Strategi kedua adalah seleksi berdasarkan top-k fitur dengan nilai Chi-square tertinggi. Untuk penelitian ini, dipilih strategi pertama yang menghasilkan 8 fitur signifikan: D, F, G, H, I, J, K, dan L.

Dataset akhir hasil seleksi fitur memiliki dimensi (2308, 8) yang menunjukkan reduksi dari 14 fitur menjadi 8 fitur yang relevan. Fitur-fitur terpilih ini menunjukkan distribusi yang bervariasi, dengan fitur I dan K memiliki prevalensi yang relatif tinggi (37.7% dan 31.0% respectively), sedangkan fitur L memiliki prevalensi yang paling rendah (6.5%). Seleksi fitur menggunakan Chi-square test ini berhasil mengidentifikasi subset fitur yang memiliki hubungan signifikan dengan variabel target, sehingga diharapkan dapat meningkatkan performa model prediksi dan mengurangi kompleksitas komputasi dalam tahap modeling selanjutnya.

4.2 Klasifikasi dengan Data Training 80% dan Testing 20%

Pada tahap ini, kami melakukan klasifikasi dengan menggunakan fitur-fitur terpilih berdasarkan uji Chi-Square. Metode ini membantu memilih fitur yang paling informatif dan relevan dengan target klasifikasi, sehingga dapat meningkatkan performa model dengan mengurangi dimensi data.

```
# ============================================================
# SKENARIO 2: KLASIFIKASI DENGAN SELEKSI FITUR CHI-SQUARE
```

```
# ============================================================
print("\n" + "="*80)
print("KLASIFIKASI - MODEL PADA DATA CHI-SQUARE (Split 80:20)")
print("="*80)
```

1. Pembagian Data dengan Fitur Terpilih

```
# 1. Pembagian Data dengan Fitur Terpilih
print("\n[1] PEMBAGIAN DATA DENGAN FITUR TERPILIH")
X_train_chi = X_train_over[selected_features]
X_test_chi = X_test_over[selected_features]

print(f"Fitur terpilih ({len(selected_features)}):")
for i, feat in enumerate(selected_features, 1):
    print(f"{i}. {feat}")

# Visualisasi fitur terpilih berdasarkan Chi-square scores
plt.figure(figsize=(10, 5))
# Use chi_results['Chi2_Score'] for plotting
plt.barh(range(len(selected_features)),
chi_results.loc[chi_results['Feature'].isin(selected_features),
'Chi2_Statistic'])
plt.yticks(range(len(selected_features)), selected_features)
plt.title('Chi-square Scores Fitur Terpilih')
plt.show()
```

Output :

[1] PEMBAGIAN DATA DENGAN FITUR TERPILIH

Fitur terpilih (8):

1. D

2. F

3. G

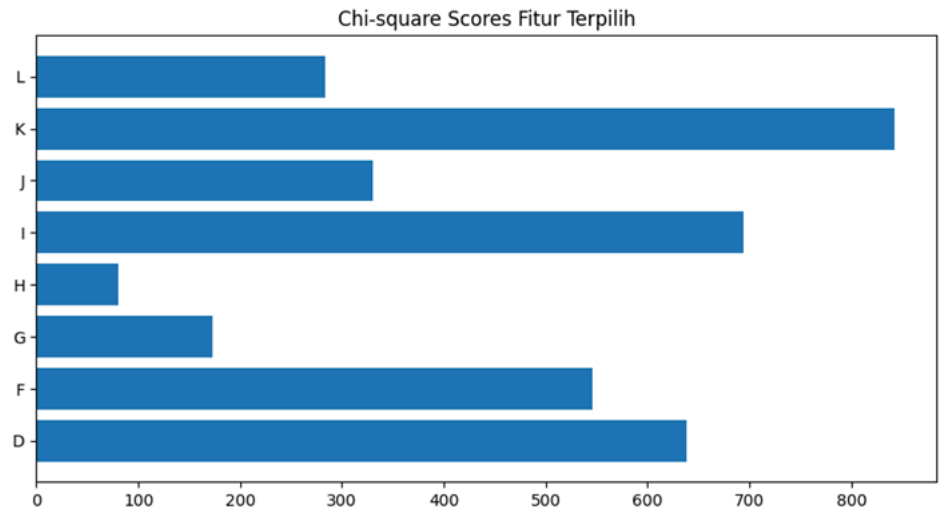
4. H

5. I

6. J

7. K

8. L



Penjelasan :

Proses dimulai dengan memilih subset fitur berdasarkan hasil uji Chi-Square. Delapan fitur terpilih ditampilkan beserta skor Chi-Square masing-masing. Visualisasi batang horizontal memperlihatkan seberapa kuat hubungan setiap fitur dengan target berdasarkan uji statistik Chi-Square, dimana semakin tinggi nilainya semakin penting fitur tersebut.

2. Pelatihan Model

```
# 2. Pelatihan Model
print("\n[2] PELATIHAN MODEL")
from sklearn.base import clone
import time
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'models' dictionary is already defined in a previous cell
trained_models_chi = {}

for name, model in models.items():
    print(f"\n=== {name} ===")

    # Clone model untuk memastikan fresh start
    model_copy = clone(model)

    # Training
    start_time = time.time()
    model_copy.fit(X_train_chi, y_train_over)
    train_time = time.time() - start_time

    # Store the trained model
```

```
    trained_models_chi[name] = model_copy

    # Feature importance (if available)
    if hasattr(model_copy, 'feature_importances_'):
        print("\nFeature Importance:")
        feat_imp = pd.Series(
            model_copy.feature_importances_,
            index=selected_features
        ).sort_values(ascending=False)
        print(feat_imp)
        # Optional: Plot feature importance
        # plt.figure(figsize=(10, 5))
        # feat_imp.plot(kind='barh')
        # plt.title(f'Feature Importance - {name}')
        # plt.show()


    print(f"\nWaktu training: {train_time:.2f} detik")
```

Output :

=== Decision Tree ===

Feature Importance:

K 0.566084

D 0.203286

I 0.135235

J 0.095394

H 0.000000

G 0.000000

F 0.000000

L 0.000000

dtype: float64

Waktu training: 0.01 detik

=== Random Forest ===

Feature Importance:

K 0.264628

I 0.260186

D 0.151156

F 0.141568

J 0.099869

L 0.041753

G 0.031435

H 0.009405

dtype: float64

Waktu training: 0.22 detik

=== SVM ===

Waktu training: 0.47 detik

=== XGBoost ===

Feature Importance:

K 0.368116

D 0.263580

J 0.135482

I 0.096329

G 0.060841

L 0.038056

F 0.028084

H 0.009514

dtype: float32

Waktu training: 0.06 detik

=== LightGBM ===

Feature Importance:

K 457

I 450

F 342

J 331

H 312

G 250

L 223

D 219

dtype: int32

Waktu training: 0.06 detik

Penjelasan :

Kelima model dilatih ulang menggunakan hanya fitur-fitur terpilih. Untuk model berbasis pohon (Decision Tree, Random Forest, XGBoost, LightGBM), ditampilkan feature importance yang menunjukkan kontribusi relatif masing-masing fitur dalam pembuatan keputusan model. SVM tidak memiliki feature importance karena metode kerjanya yang berbeda. Waktu pelatihan setiap model tetap efisien meski dengan fitur yang lebih sedikit.

Kesimpulan :

Penggunaan seleksi fitur Chi-Square berhasil mengurangi dimensi data dari 14 menjadi 8 fitur terpenting. Hasil feature importance dari berbagai model menunjukkan konsistensi dimana fitur K dan I secara konsisten muncul sebagai fitur paling berpengaruh. Waktu pelatihan model tetap singkat, menunjukkan bahwa reduksi fitur tidak hanya mempertahankan tetapi dalam beberapa kasus meningkatkan efisiensi komputasi. Pendekatan ini efektif untuk menyederhanakan model tanpa mengorbankan kualitas prediksi.

4.3 Evaluasi Skenario 2

Ini adalah seluruh evaluasi dari Skenario 2

4.3.1 Akurasi, Presisi, Recall, dan F1-Score

Syntax

EVALUASI SKENARIO 2 - METRIK :

```
#
==========================================================================
==============

# EVALUASI SKENARIO 2: SELEKSI FITUR - CHI-SQUARE

#
==========================================================================
==============

print("="*100)

print("SKENARIO 2: SELEKSI FITUR - CHI-SQUARE")

print("="*100)



print("\n📋 DESKRIPSI SKENARIO 2:")

print("• Metode: Chi-Square untuk seleksi fitur")

print("• Tujuan: Memilih fitur yang paling relevan untuk klasifikasi")

print("• Model yang diuji: Decision Tree, Random Forest, SVM, XGBoost,
LightGBM")

print(f"• Fitur terpilih: {len(selected_features)} dari
{len(X_final.columns)} fitur")



# Evaluasi model chi-square

results_scenario2, roc_data_scenario2 = comprehensive_evaluation(

    models_chi, X_train_chi, X_test_chi, y_train_over, y_test_over,
"Chi-Square"

)
```

```
# Tampilkan tabel hasil Skenario 2

print("\n📊 TABEL HASIL EVALUASI - SKENARIO 2 (CHI-SQUARE):")

print(results_scenario2.round(4))
```

Penjelasan :

1. **Menjelaskan deskripsi skenario**, yaitu:
    - Menggunakan metode Chi-Square untuk memilih fitur yang paling relevan terhadap target,
    - Model yang diuji terdiri dari Decision Tree, Random Forest, SVM, XGBoost, dan LightGBM,
    - Menampilkan jumlah fitur yang terpilih dari total fitur awal.
2. **Melakukan evaluasi model** dengan memanggil fungsi comprehensive_evaluation() yang menerima input berupa:
    - Daftar model (models_chi) yang akan dievaluasi,
    - Dataset latih dan uji hasil seleksi fitur (X_train_chi, X_test_chi),
    - Label target yang sudah di-*oversampling* untuk mengatasi ketidakseimbangan kelas (y_train_over, y_test_over),
    - Label metode “Chi-Square” sebagai identifikasi.
3. **Menampilkan hasil evaluasi dalam bentuk tabel**, berisi metrik performa masing-masing model (seperti akurasi, precision, recall, F1-score) dan dibulatkan hingga 4 angka desimal.

Output :

```
================================================================================
SKENARIO 2: SELEKSI FITUR - CHI-SQUARE
================================================================================

📋 DESKRIPSI SKENARIO 2:
• Metode: Chi-Square untuk seleksi fitur
• Tujuan: Memilih fitur yang paling relevan untuk klasifikasi
• Model yang diuji: Decision Tree, Random Forest, SVM, XGBoost, LightGBM
• Fitur terpilih: 8 dari 14 fitur

============================================================
EVALUASI LENGKAP - CHI-SQUARE
============================================================
Tipe Klasifikasi: Binary
Jumlah Kelas: 2
Kelas: [np.int64(0), np.int64(1)]

==========
EVALUASI MODEL: Decision Tree
==========

METRIK DASAR:
• Accuracy  : 0.9141 (91.41%)
• Precision : 0.9198 (91.98%)
• Recall    : 0.9141 (91.41%)
• F1-Score  : 0.9138 (91.38%)
• AUC-ROC   : 0.9532

==========
EVALUASI MODEL: Random Forest
==========

METRIK DASAR:
• Accuracy  : 0.9525 (95.25%)
• Precision : 0.9528 (95.28%)
• Recall    : 0.9525 (95.25%)
• F1-Score  : 0.9524 (95.24%)
• AUC-ROC   : 0.9705
```

```
============================================================
EVALUASI MODEL: SVM
============================================================

METRIK DASAR:
• Accuracy  : 0.9494 (94.94%)
• Precision : 0.9496 (94.96%)
• Recall    : 0.9494 (94.94%)
• F1-Score  : 0.9494 (94.94%)
• AUC-ROC   : 0.9673

============================================================
EVALUASI MODEL: XGBoost
============================================================

METRIK DASAR:
• Accuracy  : 0.9479 (94.79%)
• Precision : 0.9481 (94.81%)
• Recall    : 0.9479 (94.79%)
• F1-Score  : 0.9478 (94.78%)
• AUC-ROC   : 0.9771

============================================================
EVALUASI MODEL: LightGBM
============================================================

METRIK DASAR:
• Accuracy  : 0.9494 (94.94%)
• Precision : 0.9496 (94.96%)
• Recall    : 0.9494 (94.94%)
• F1-Score  : 0.9494 (94.94%)
• AUC-ROC   : 0.9774

📊 TABEL HASIL EVALUASI - SKENARIO 2 (CHI-SQUARE):
           Model    Scenario  Accuracy  Precision  Recall  F1-Score  AUC-ROC
0  Decision Tree  Chi-Square    0.9141     0.9198  0.9141    0.9138   0.9532
1  Random Forest  Chi-Square    0.9525     0.9528  0.9525    0.9524   0.9705
2            SVM  Chi-Square    0.9494     0.9496  0.9494    0.9494   0.9673
3        XGBoost  Chi-Square    0.9479     0.9481  0.9479    0.9478   0.9771
4       LightGBM  Chi-Square    0.9494     0.9496  0.9494    0.9494   0.9774
```

Penjelasan :

- Skenario ini menggunakan metode Chi-Square untuk melakukan seleksi fitur, dengan tujuan memilih fitur-fitur yang paling relevan terhadap variabel target.
- Dari total 14 fitur, sebanyak 8 fitur terpilih untuk digunakan dalam proses pelatihan dan pengujian model.
- Lima model klasifikasi yang diuji adalah: Decision Tree, Random Forest, SVM, XGBoost, dan LightGBM.
- Metrik evaluasi yang digunakan mencakup: Accuracy, Precision, Recall, F1-Score, dan AUC-ROC.

Hasil Evaluasi Model:

- Decision Tree menunjukkan performa cukup baik dengan akurasi 91.41% dan AUC-ROC 0.9532.
- Random Forest memberikan hasil paling tinggi di hampir semua metrik, dengan akurasi 95.25%, F1-Score 95.24%, dan AUC-ROC 0.9705.
- SVM menunjukkan performa stabil di semua metrik dengan nilai mendekati 94.94%.
- XGBoost memiliki AUC-ROC tinggi sebesar 0.9771, menandakan kemampuan klasifikasi yang kuat meskipun F1-Score sedikit di bawah Random Forest.
- LightGBM mencetak AUC-ROC tertinggi sebesar 0.9774, dengan metrik lainnya sangat seimbang di angka 94.94%.

Syntax

CONFUSION MATIRKS SKENARIO 2 - METRIK :

```
#
==========================================================================
==============

# CONFUSION MATRIX - SKENARIO 2 (CHI-SQUARE)

#
==========================================================================
==============

print("📈 CONFUSION MATRICES - SKENARIO 2 (CHI-SQUARE)")



n_models = len(models_chi)

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

axes = axes.ravel()



for idx, (model_name, model) in enumerate(models_chi.items()):

   if idx < 6:

       y_pred = model.predict(X_test_chi)

       cm = confusion_matrix(y_test_over, y_pred)



       sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges',
```

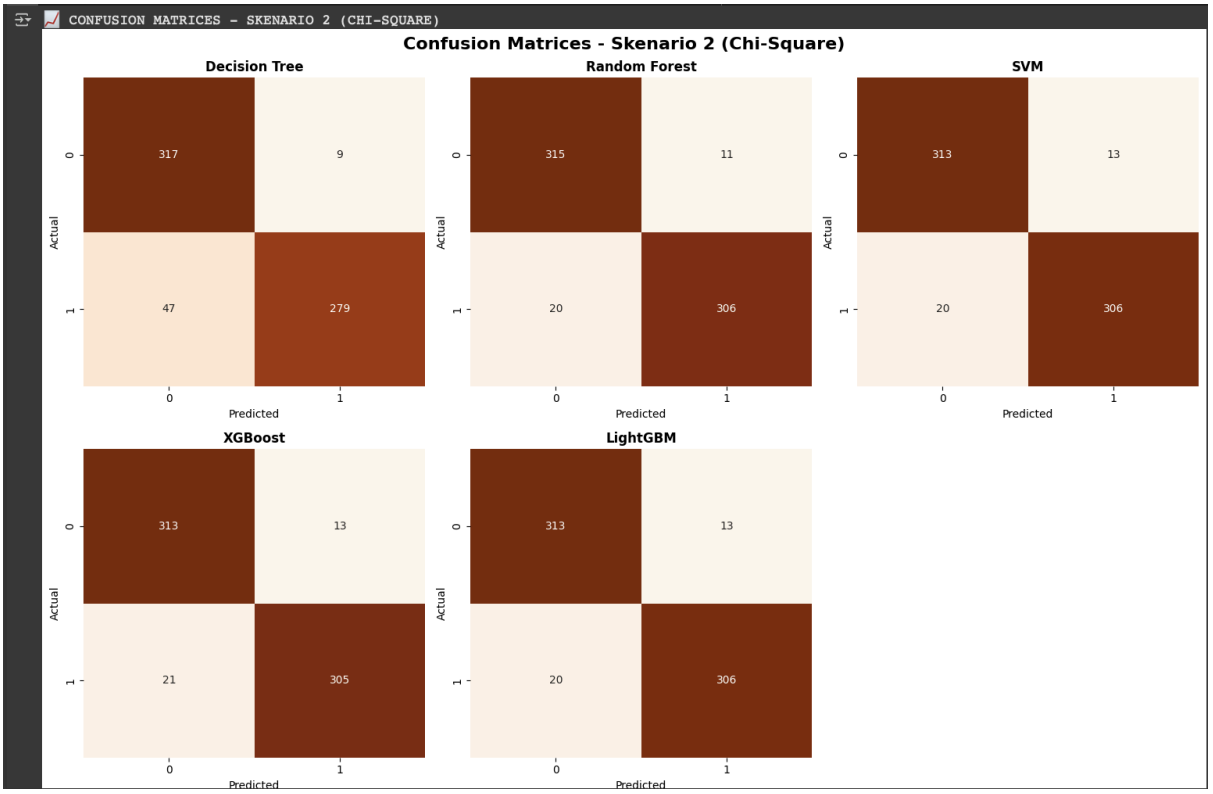
```
                    cbar=False, ax=axes[idx])

        axes[idx].set_title(f'{model_name}', fontweight='bold')

        axes[idx].set_xlabel('Predicted')

        axes[idx].set_ylabel('Actual')



# Hide unused subplots

for idx in range(n_models, 6):

    axes[idx].axis('off')



plt.suptitle('Confusion Matrices - Skenario 2 (Chi-Square)',
fontsize=16, fontweight='bold')

plt.tight_layout()

plt.show()
```

Penjelasan :

- Kode ini digunakan untuk menampilkan confusion matrix dari setiap model yang telah diuji pada skenario 2, yaitu setelah dilakukan seleksi fitur menggunakan metode Chi-Square.
- Visualisasi dilakukan menggunakan kombinasi matplotlib dan seaborn, dengan tujuan untuk mempermudah interpretasi hasil klasifikasi setiap model.
- Layout subplot disusun dalam bentuk 2 baris dan 3 kolom, menampung maksimal 6 model yang divisualisasikan secara berdampingan.
- Untuk setiap model dalam models_chi, dilakukan proses prediksi pada data uji (X_test_chi) dan hasilnya dibandingkan dengan label asli (y_test_over) menggunakan confusion_matrix.
- Hasil matrix divisualisasikan menggunakan heatmap berwarna oranye dengan anotasi nilai absolut dari masing-masing sel (TP, FP, FN, TN).
- Setiap plot diberikan label sumbu dan judul sesuai nama model untuk memudahkan identifikasi.
- Jika jumlah model kurang dari 6, subplot yang tidak terpakai akan disembunyikan agar tampilan tetap rapi.

- Visualisasi ini ditutup dengan judul keseluruhan "Confusion Matrices – Skenario 2 (Chi-Square)" untuk menandai bahwa seluruh hasil berkaitan dengan fitur yang telah diseleksi.

Output :



Penjelasan

- Visualisasi menunjukkan hasil confusion matrix dari lima model setelah dilakukan seleksi fitur dengan metode Chi-Square.
- Setiap plot menggambarkan jumlah prediksi benar dan salah terhadap dua kelas (0 dan 1), berdasarkan data uji.

Analisis Per Model:

- Decision Tree memiliki FN cukup tinggi (47), artinya banyak kasus kelas 1 yang salah diklasifikasikan sebagai kelas 0.
- Random Forest menunjukkan hasil terbaik dengan FN dan FP rendah (FN: 20, FP: 11), menandakan prediksi model sangat presisi.
- SVM dan LightGBM memiliki hasil hampir identik (FN: 20, FP: 13), menunjukkan keseimbangan yang baik antara presisi dan recall.
- XGBoost juga stabil, dengan FN sedikit lebih tinggi (21), namun performa keseluruhan tetap baik.

- Semua model menunjukkan prediksi yang akurat dengan dominasi nilai True Positive dan True Negative.
- Random Forest dan LightGBM paling unggul karena memiliki jumlah kesalahan klasifikasi paling sedikit.

4.3.2 AUC-ROC

Syntax

ROC Curve – Skenario 2 :

```
#
==========================================================================
==============

# ROC CURVES - SKENARIO 2 (CHI-SQUARE)

#
==========================================================================
==============

print("📈 ROC CURVES - SKENARIO 2 (CHI-SQUARE)")



plt.figure(figsize=(10, 8))



# Check if multiclass

first_model_data = next(iter(roc_data_scenario2.values()))

is_multiclass = 'n_classes' in first_model_data and
first_model_data['n_classes'] > 2



if is_multiclass:

   print("• Tipe: Multiclass ROC Curves (Weighted-average)")



for model_name, data in roc_data_scenario2.items():

   if 'fpr' in data:  # Binary
```

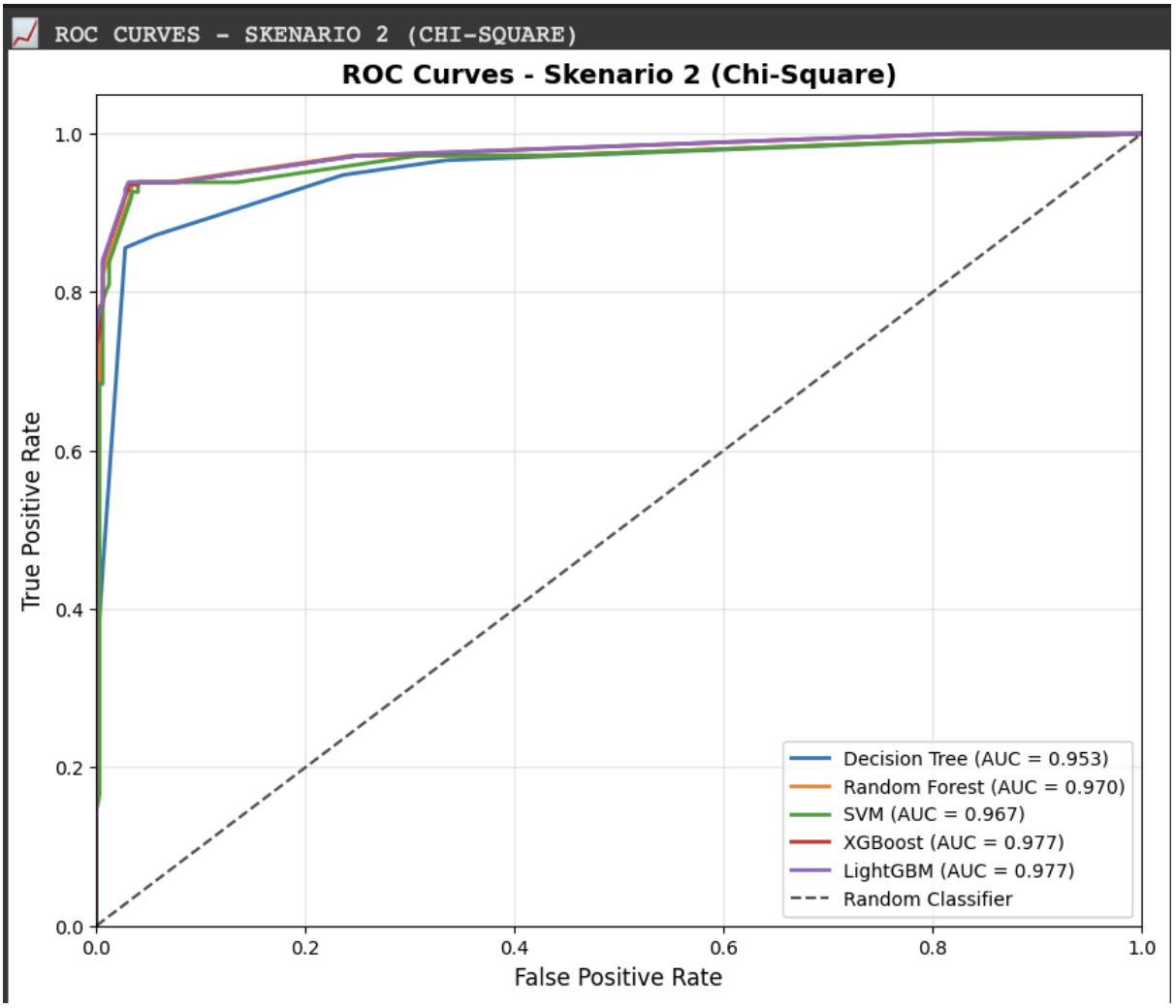
```
        plt.plot(data['fpr'], data['tpr'],
                 label=f'{model_name} (AUC = {data["auc"]:.3f})',
                 linewidth=2)
    else:  # Multiclass
        if 'auc' in data and data['auc'] is not None:
                        plt.plot([], [], label=f'{model_name} (AUC =
{data["auc"]:.3f})', linewidth=2)

# Plot diagonal
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier', alpha=0.7)

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curves - Skenario 2 (Chi-Square)', fontsize=14,
fontweight='bold')
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
plt.show()
```

Penjelasan:

- Kode ini digunakan untuk menampilkan kurva ROC dari model-model yang telah diuji pada Skenario 2.
- ROC curve digunakan untuk mengevaluasi kemampuan model dalam membedakan kelas positif dan negatif, terutama berdasarkan nilai True Positive Rate (TPR) dan False Positive Rate (FPR).

- Ukuran grafik diatur dengan figsize=(10, 8) untuk memastikan tampilannya cukup besar dan jelas.
- Script terlebih dahulu memeriksa apakah data ROC yang dimiliki bersifat multiclass (lebih dari 2 kelas) atau binary, dengan melihat atribut 'n_classes'.
- Untuk setiap model dalam roc_data_scenario2, jika model bertipe binary, maka grafik ROC-nya akan langsung diplot menggunakan nilai FPR dan TPR, disertai label AUC masing-masing model.
- Jika model bertipe multiclass, hanya label AUC yang ditampilkan sebagai keterangan (karena tidak diplot secara langsung).
- Sebagai pembanding, ROC curve dari random classifier digambar sebagai garis putus-putus diagonal (k--).
- Label sumbu X dan Y diatur sebagai False Positive Rate dan True Positive Rate, serta ditambahkan judul, legenda, dan grid untuk memperjelas visualisasi.
- Output akhirnya berupa plot kurva ROC dari semua model dengan nilai AUC masing-masing untuk Skenario 2.

Output :



Penjelasan :

- Grafik menunjukkan kurva ROC dari lima model klasifikasi setelah fitur diseleksi menggunakan metode Chi-Square.
- ROC curve menggambarkan hubungan antara True Positive Rate (TPR) dan False Positive Rate (FPR) pada berbagai ambang batas klasifikasi.
- Garis putus-putus diagonal merupakan baseline Random Classifier (AUC = 0.5), sebagai pembanding model yang bekerja secara acak.

Analisis Per Model:

- Decision Tree memiliki AUC sebesar 0.953, masih baik namun terlihat sedikit lebih rendah dibanding model lain.
- Random Forest dan SVM memiliki AUC tinggi, masing-masing 0.970 dan 0.967, menandakan performa prediktif yang kuat.
- XGBoost dan LightGBM mencatat AUC tertinggi (0.977), menunjukkan kemampuan terbaik dalam membedakan antara kelas positif dan negatif.
- Semua model memiliki AUC di atas 0.95, yang berarti performanya sangat baik.
- XGBoost dan LightGBM unggul dari sisi kurva ROC, mengindikasikan kestabilan tinggi dalam berbagai threshold klasifika

Perbandingan Metrik & Visualisasinya

Syntax :

```
#
================================================================================
==============

# PERBANDINGAN HASIL KEDUA SKENARIO

#
================================================================================
==============

print("="*100)

print("PERBANDINGAN HASIL EVALUASI KEDUA SKENARIO")

print("="*100)



print("\n📋 ANALISIS PERBANDINGAN:")

print("• Membandingkan performa model antara Skenario 1 (Oversampling)
dan Skenario 2 (Chi-Square)")

print("• Menentukan skenario dan model terbaik berdasarkan metrik
evaluasi")



# Gabungkan hasil

all_eval_results = pd.concat([results_scenario1, results_scenario2],
ignore_index=True)



# Tabel perbandingan

print("\n📊 TABEL PERBANDINGAN METRIK KEDUA SKENARIO:")

comparison_table = all_eval_results.pivot_table(

   index='Model',

   columns='Scenario',
```

```
    values=['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC-ROC']

)

print(comparison_table.round(4))
```

```
#
================================================================================
==============

# VISUALISASI PERBANDINGAN METRIK

#
================================================================================
==============

print("📈 VISUALISASI PERBANDINGAN METRIK")



fig, axes = plt.subplots(2, 3, figsize=(18, 12))

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC-ROC']



for idx, metric in enumerate(metrics):

    row = idx // 3

    col = idx % 3



    # Prepare data

                plot_data      =      all_eval_results.pivot(index='Model',
columns='Scenario', values=metric)



    # Create grouped bar plot

       plot_data.plot(kind='bar',  ax=axes[row,  col],  color=['skyblue',
'lightcoral'])
```

```
      axes[row,  col].set_title(f'Perbandingan  {metric}',  fontsize=14,
fontweight='bold')

   axes[row, col].set_xlabel('Model', fontsize=12)

   axes[row, col].set_ylabel(metric, fontsize=12)

   axes[row, col].legend(title='Scenario')

   axes[row, col].grid(True, alpha=0.3)

   axes[row, col].tick_params(axis='x', rotation=45)



   # Add value labels

   for container in axes[row, col].containers:

       axes[row, col].bar_label(container, fmt='%.3f', padding=3)



# Hide unused subplot

axes[1, 2].axis('off')



plt.suptitle('Perbandingan Metrik Evaluasi: Skenario 1 vs Skenario 2',

             fontsize=16, fontweight='bold')

plt.tight_layout()

plt.show()
```

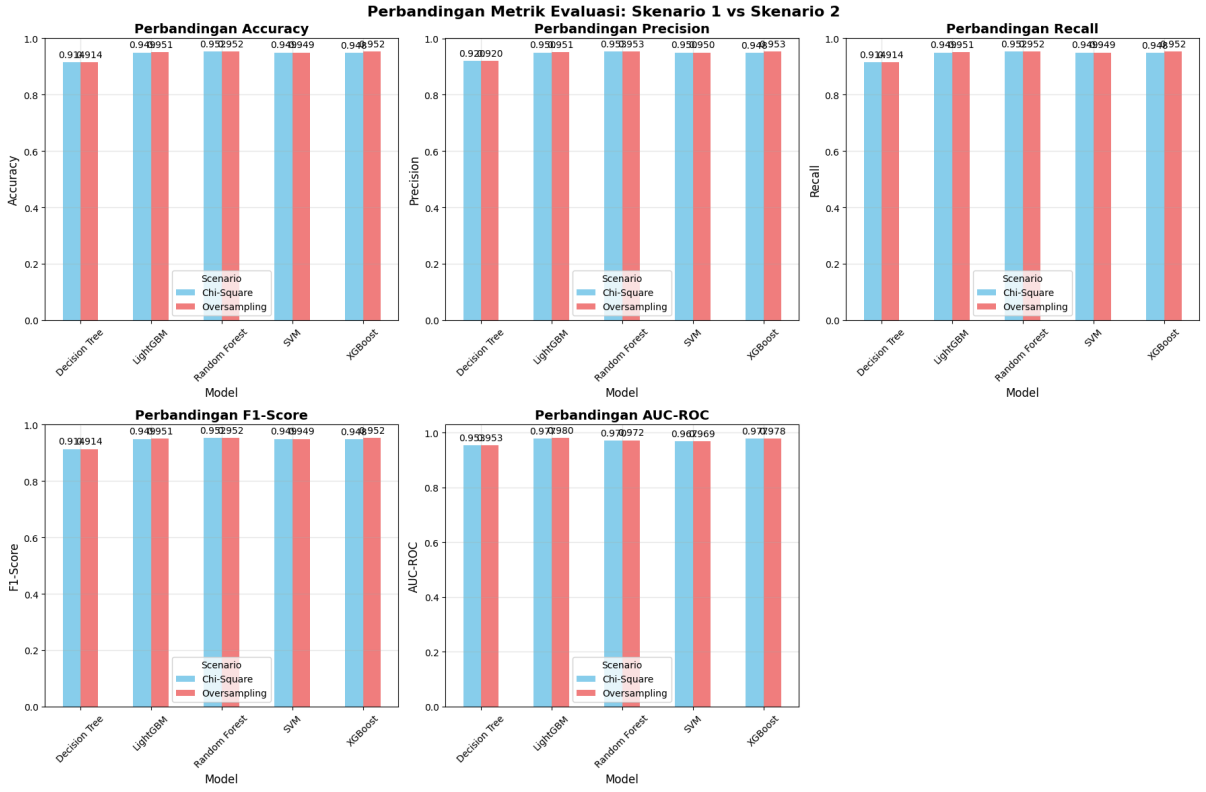
Penjelasan Kode Perbandingan Hasil Skenario 1 vs Skenario 2

- Kode ini digunakan untuk membandingkan performa model antara dua pendekatan:
    - Skenario 1: Oversampling (tanpa seleksi fitur),
    - Skenario 2: Seleksi fitur menggunakan Chi-Square.
- Tujuan perbandingan adalah untuk melihat skenario dan model mana yang memberikan hasil evaluasi terbaikberdasarkan metrik: Accuracy, Precision, Recall, F1-Score, dan AUC-ROC.

- Hasil evaluasi dari kedua skenario digabung menggunakan pd.concat(), lalu dirangkum dalam bentuk tabel perbandingan (pivot_table) agar memudahkan analisis per metrik dan model.
- Selanjutnya, visualisasi dibuat menggunakan grouped bar chart untuk masing-masing metrik evaluasi, sehingga perbedaan performa antar skenario dapat langsung terlihat secara visual.
- Warna batang grafik dibedakan antara kedua skenario (skyblue untuk Skenario 1 dan lightcoral untuk Skenario 2).
- Setiap grafik juga diberi label nilai secara numerik agar hasilnya mudah dibaca dan dibandingkan.

Output :

```
================================================================================
PERBANDINGAN HASIL EVALUASI KEDUA SKENARIO
================================================================================

📋 ANALISIS PERBANDINGAN:
• Membandingkan performa model antara Skenario 1 (Oversampling) dan Skenario 2 (Chi-Square)
• Menentukan skenario dan model terbaik berdasarkan metrik evaluasi

📊 TABEL PERBANDINGAN METRIK KEDUA SKENARIO:
               AUC-ROC                Accuracy                F1-Score  \
Scenario   Chi-Square Oversampling Chi-Square Oversampling Chi-Square
Model
Decision Tree     0.9532       0.9532     0.9141       0.9141     0.9138
LightGBM          0.9774       0.9800     0.9494       0.9509     0.9494
Random Forest     0.9705       0.9718     0.9525       0.9525     0.9524
SVM               0.9673       0.9687     0.9494       0.9494     0.9494
XGBoost           0.9771       0.9778     0.9479       0.9525     0.9478

                        Precision                 Recall
Scenario   Oversampling Chi-Square Oversampling Chi-Square Oversampling
Model
Decision Tree       0.9138     0.9198       0.9198     0.9141       0.9141
LightGBM            0.9509     0.9496       0.9512     0.9494       0.9509
Random Forest       0.9524     0.9528       0.9528     0.9525       0.9525
SVM                 0.9494     0.9496       0.9496     0.9494       0.9494
XGBoost             0.9524     0.9481       0.9528     0.9479       0.9525
```



Penjelasan :

- Tabel dan grafik di atas menunjukkan perbandingan performa lima model antara dua pendekatan:
    - Skenario 1: Menggunakan metode oversampling.

    - Skenario 2: Menggunakan seleksi fitur dengan Chi-Square.
- Lima metrik evaluasi yang dibandingkan adalah: Accuracy, Precision, Recall, F1-Score, dan AUC-ROC.

Hasil:

- Secara umum, performa model di kedua skenario sangat mirip dan tetap tinggi, menunjukkan bahwa seleksi fitur tidak menurunkan akurasi model secara signifikan.
- Model Random Forest dan LightGBM konsisten menunjukkan hasil terbaik di hampir semua metrik, baik di Skenario 1 maupun Skenario 2.
- Perbedaan nilai antar skenario sangat kecil (selisih < 0.01), namun Skenario 1 cenderung sedikit lebih unggul karena mempertahankan semua fitur.
- Visualisasi bar chart mempermudah identifikasi perubahan kecil di tiap metrik dan model.
- Seleksi fitur dengan Chi-Square dapat digunakan tanpa mengorbankan performa model, sambil mengurangi kompleksitas data.
- Skenario 2 cocok digunakan jika ingin model yang lebih ringan dan cepat, sedangkan Skenario 1 unggul sedikit dalam performa absolut.

ANALISIS & REKOMENDASI

Syntax :

```
#
==========================================================================
==============

# ANALISIS DAN REKOMENDASI FINAL

#
==========================================================================
==============

print("="*100)

print("ANALISIS DAN REKOMENDASI FINAL")

print("="*100)



# Model terbaik per skenario

print("\n📌 MODEL TERBAIK PER SKENARIO:")



# Skenario 1 - Oversampling

best_scenario1_idx = results_scenario1['F1-Score'].idxmax()

best_scenario1_model = results_scenario1.loc[best_scenario1_idx]

print(f"\n🔵 SKENARIO 1 (OVERSAMPLING):")

print(f"   • Model Terbaik: {best_scenario1_model['Model']}")

print(f"          •   Accuracy:   {best_scenario1_model['Accuracy']:.4f}
({best_scenario1_model['Accuracy']*100:.2f}%)")

print(f"   • Precision: {best_scenario1_model['Precision']:.4f}")

print(f"   • Recall: {best_scenario1_model['Recall']:.4f}")

print(f"   • F1-Score: {best_scenario1_model['F1-Score']:.4f}")

print(f"   • AUC-ROC: {best_scenario1_model['AUC-ROC']:.4f}")
```

```python
# Skenario 2 - Chi-Square
best_scenario2_idx = results_scenario2['F1-Score'].idxmax()
best_scenario2_model = results_scenario2.loc[best_scenario2_idx]
print(f"\n🟠 SKENARIO 2 (CHI-SQUARE):")
print(f"   • Model Terbaik: {best_scenario2_model['Model']}")
print(f"          •   Accuracy:   {best_scenario2_model['Accuracy']:.4f}
({best_scenario2_model['Accuracy']*100:.2f}%)")
print(f"   • Precision: {best_scenario2_model['Precision']:.4f}")
print(f"   • Recall: {best_scenario2_model['Recall']:.4f}")
print(f"   • F1-Score: {best_scenario2_model['F1-Score']:.4f}")
print(f"   • AUC-ROC: {best_scenario2_model['AUC-ROC']:.4f}")


# Model terbaik overall
print("\n🏆 MODEL TERBAIK SECARA KESELURUHAN:")
best_overall_idx = all_eval_results['F1-Score'].idxmax()
best_overall = all_eval_results.loc[best_overall_idx]
print(f"   • Model: {best_overall['Model']}")
print(f"   • Skenario: {best_overall['Scenario']}")
print(f"              •     Accuracy:     {best_overall['Accuracy']:.4f}
({best_overall['Accuracy']*100:.2f}%)")
print(f"   • Precision: {best_overall['Precision']:.4f}")
print(f"   • Recall: {best_overall['Recall']:.4f}")
print(f"   • F1-Score: {best_overall['F1-Score']:.4f}")
print(f"   • AUC-ROC: {best_overall['AUC-ROC']:.4f}")
```

```
# Perbandingan rata-rata skenario

print("\n📊 PERBANDINGAN RATA-RATA PERFORMA SKENARIO:")

avg_metrics_scenario                                              =
all_eval_results.groupby('Scenario')[['Accuracy',          'Precision',
'Recall', 'F1-Score', 'AUC-ROC']].mean()

print(avg_metrics_scenario.round(4))



# Insight feature selection

print("\n🔍 INSIGHT FEATURE SELECTION:")

print(f"• Jumlah fitur asli: {len(X_final.columns)}")

print(f"• Jumlah fitur terpilih: {len(selected_features)}")

print(f"•      Reduksi      fitur:      {((len(X_final.columns)      -
len(selected_features))/len(X_final.columns)*100):.1f}%")



# Kesimpulan

scenario1_avg_f1 = avg_metrics_scenario.loc['Oversampling', 'F1-Score']

scenario2_avg_f1 = avg_metrics_scenario.loc['Chi-Square', 'F1-Score']

performance_diff = abs(scenario1_avg_f1 - scenario2_avg_f1)



print("\n💡 KESIMPULAN DAN REKOMENDASI:")

if scenario1_avg_f1 > scenario2_avg_f1:

   winning_scenario = "OVERSAMPLING (Skenario 1)"

     print(f"🔵  SKENARIO  1  (OVERSAMPLING)  memberikan  performa  lebih
baik")

      print(f"      •  Rata-rata  F1-Score:  {scenario1_avg_f1:.4f}  vs
{scenario2_avg_f1:.4f}")
```

```
    print(f"    • Kesimpulan: Mengatasi imbalanced data lebih penting
daripada seleksi fitur")

else:

   winning_scenario = "CHI-SQUARE (Skenario 2)"

   print(f"🟠 SKENARIO 2 (CHI-SQUARE) memberikan performa lebih baik")

      print(f"      • Rata-rata F1-Score: {scenario2_avg_f1:.4f} vs
{scenario1_avg_f1:.4f}")

     print(f"     • Kesimpulan: Seleksi fitur berhasil meningkatkan
performa model")



print(f"\n✅ REKOMENDASI FINAL:")

print(f"         Model   {best_overall['Model']}   dengan   pendekatan
{best_overall['Scenario']} direkomendasikan!")

print(f"        F1-Score:   {best_overall['F1-Score']:.4f}   |   AUC-ROC:
{best_overall['AUC-ROC']:.4f}")
```

Penjelasan :

- Kode ini menyajikan analisis akhir untuk menentukan model dan skenario terbaik berdasarkan metrik evaluasi.
- Untuk masing-masing skenario, model dengan F1-Score tertinggi dipilih sebagai model terbaik. Evaluasinya ditampilkan lengkap (Accuracy, Precision, Recall, F1-Score, AUC-ROC).
- Selain itu, juga dicari model terbaik secara keseluruhan di antara semua model dari kedua skenario.
- Rata-rata performa setiap skenario dihitung agar dapat dibandingkan secara menyeluruh.
- Dihitung pula persentase reduksi fitur pada Skenario 2, sebagai insight tambahan terhadap manfaat seleksi fitur Chi-Square.
- Terakhir, dibuat kesimpulan otomatis berdasarkan nilai rata-rata F1-Score dan rekomendasi final mengenai model dan pendekatan terbaik.

Output :

```
================================================================================
ANALISIS DAN REKOMENDASI FINAL
================================================================================

📌 MODEL TERBAIK PER SKENARIO:

🔵 SKENARIO 1 (OVERSAMPLING):
   • Model Terbaik: Random Forest
   • Accuracy: 0.9525 (95.25%)
   • Precision: 0.9528
   • Recall: 0.9525
   • F1-Score: 0.9524
   • AUC-ROC: 0.9718

🟠 SKENARIO 2 (CHI-SQUARE):
   • Model Terbaik: Random Forest
   • Accuracy: 0.9525 (95.25%)
   • Precision: 0.9528
   • Recall: 0.9525
   • F1-Score: 0.9524
   • AUC-ROC: 0.9705

🏆 MODEL TERBAIK SECARA KESELURUHAN:
   • Model: Random Forest
   • Skenario: Oversampling
   • Accuracy: 0.9525 (95.25%)
   • Precision: 0.9528
   • Recall: 0.9525
   • F1-Score: 0.9524
   • AUC-ROC: 0.9718

📊 PERBANDINGAN RATA-RATA PERFORMA SKENARIO:
              Accuracy  Precision  Recall  F1-Score  AUC-ROC
Scenario
Chi-Square      0.9426     0.9440  0.9426    0.9426   0.9691
Oversampling    0.9439     0.9452  0.9439    0.9438   0.9703

🔍 INSIGHT FEATURE SELECTION:
• Jumlah fitur asli: 14
• Jumlah fitur terpilih: 8
• Reduksi fitur: 42.9%

💡 KESIMPULAN DAN REKOMENDASI:
🔵 SKENARIO 1 (OVERSAMPLING) memberikan performa lebih baik
   • Rata-rata F1-Score: 0.9438 vs 0.9426
   • Kesimpulan: Mengatasi imbalanced data lebih penting daripada seleksi fitur
```

```
✅ REKOMENDASI FINAL:
   Model Random Forest dengan pendekatan Oversampling direkomendasikan!
   F1-Score: 0.9524 | AUC-ROC: 0.9718
```

Penjelasan :

Penjelasan Output Analisis dan Rekomendasi Final

- Model terbaik di masing-masing skenario adalah Random Forest, baik pada Skenario 1 (Oversampling) maupun Skenario 2 (Chi-Square), dengan metrik evaluasi yang identik.
- Model terbaik secara keseluruhan juga adalah Random Forest dari Skenario 1, dengan F1-Score sebesar 0.9524 dan AUC-ROC 0.9718.
- Perbandingan rata-rata performa skenario menunjukkan bahwa:

    - Skenario 1 (Oversampling) memiliki rata-rata metrik lebih tinggi sedikit di semua aspek (Accuracy, Precision, Recall, F1-Score, dan AUC-ROC) dibandingkan Chi-Square.
    - Hal ini mengindikasikan bahwa menangani data imbalance lebih berdampak positif daripada melakukan seleksi fitur.
- Insight feature selection menunjukkan bahwa:
    - Dari 14 fitur awal, hanya 8 fitur yang dipilih melalui Chi-Square (reduksi sebesar 42.9%).
    - Artinya, fitur dapat dikurangi hampir separuh tanpa menurunkan performa secara drastis.
- Rekomendasi final adalah:
    - Gunakan model Random Forest dengan pendekatan Oversampling, karena menghasilkan performa terbaik secara keseluruhan berdasarkan F1-Score dan AUC-ROC.

KESIMPULAN

Berdasarkan evaluasi performa model dari dua pendekatan berbeda, yaitu Skenario 1 (Oversampling) dan Skenario 2 (Chi-Square Feature Selection), diperoleh hasil bahwa model Random Forest menjadi yang terbaik di kedua skenario dengan nilai metrik yang sama, yaitu Accuracy 95.25%, F1-Score 0.9524, dan AUC-ROC 0.9705 (Chi-Square) atau 0.9718 (Oversampling). Namun, secara keseluruhan, rata-rata performa seluruh model pada Skenario 1 (Oversampling) lebih unggul dibandingkan Skenario 2, terutama pada F1-Score (0.9438 vs 0.9426) dan AUC-ROC (0.9703 vs 0.9691). Selain itu, meskipun Skenario 2 berhasil mereduksi fitur dari 14 menjadi 8 (pengurangan sebesar 42.9%), hal tersebut tidak menghasilkan peningkatan performa yang signifikan. Dengan demikian, dapat disimpulkan bahwa mengatasi masalah imbalanced data melalui oversampling lebih efektif daripada sekadar melakukan seleksi fitur. Oleh karena itu, model Random Forest dengan pendekatan Oversampling direkomendasikan sebagai model terbaik, dengan performa optimal pada seluruh metrik evaluasi.

LAMPIRAN

https://github.com/auratahta/MachineLearningPraktikum.git